PROCEEDINGS

SEKE 2007

The 19th International Conference on Software Engineering & Knowledge Engineering

Sponsored by

Knowledge Systems Institute Graduate School, USA

Technical Program

July 9-11, 2007

Hyatt Harborside Hotel, Boston, Massachusetts, USA

Organized by

Knowledge Systems Institute Graduate School

ISBN 1-891706-20-9 (paper)

Additional Copies can be ordered from:
Knowledge Systems Institute Graduate School
3420 Main Street
Skokie, IL 60076, USA
Tel:+1-847-679-3135
Fax:+1-847-679-3166
Email:office@ksi.edu
http://www.ksi.edu

Proceedings preparation, editing and printing are sponsored by
Knowledge Systems Institute Graduate School

Printed by Knowledge Systems Institute Graduate School

SEKE 2007 Foreword

PROGRAM CHAIR'S MESSAGE

Welcome to Boston. The International Conference on Software Engineering and Knowledge Engineering has entered its nineteenth year. For the past eighteen years, the Conference on Software Engineering and Knowledge Engineering has provided a unique, centralized, forum for academic and industrial researchers and practitioners to discuss the application of either software engineering methods in knowledge engineering or knowledge-based techniques in software engineering. Preference is given to papers that emphasize the transference of methods between both engineering disciplines; however, outstanding papers on software engineering or knowledge engineering alone have also been presented.

This year's technical program consists of the following four tracks organized by a great team of Vice Program chairs:

Track:	**Co-Chairs:**
Service-Oriented	George Spanoudakis, City University, UK
Data	Taghi M. Khoshgoftaar, Florida Atlantic University, USA
Applications	Jerry Gao, San Jose State University, USA
Agents	Du Zhang, California State University, USA

The SEKE2007 Program Committee selected papers for publication in the proceedings and presentation at the Conference based upon a rigorous review process of the full papers. The acceptance rate for full papers is 44% and for short papers is 23%. This year, authors from 39 countries will present papers at the conference.

I appreciate having had the opportunity to serve as the program chair for this Conference, and am very grateful for the outstanding efforts provided by the Program Committee Co-Chairs, who are listed above. The Program Committee members and reviewers provided excellent support in promptly reviewing the manuscripts. We are grateful to the authors and sessions chairs for their time and efforts to make SEKE2007 a success. As always, Dr. S. K. Chang of the Knowledge Systems Institute, USA, provided excellent guidance throughout the effort. Last but not least, we all owe a debt of gratitude the heroic efforts of Mr. Daniel Li, of the Knowledge Systems Institute.

Daniel E. Cooke
SEKE2007 Program Chair

The 19th International Conference on Software Engineering & Knowledge Engineering (SEKE 2007)

July 9-11, 2007
Hyatt Harborside Hotel, Boston, Massachusetts, USA

Organizers & Committee

Steering Committee

Vic Basili, *University of Maryland, USA*
Bruce Buchanan, *University of Pittsburgh, USA*
Shi-Kuo Chang, *University of Pittsburgh, USA*
C. V. Ramamoorthy, *University of California, Berkeley, USA*

Conference General Chair

Shi-Kuo Chang, *University of Pittsburgh, USA*

Program Chair

Daniel Cooke, *Texas Tech University*

Program Co-Chairs

George Spanoudakis, *City University, UK*
Taghi M. Khoshgoftaar, *Florida Atlantic University, USA*
Jerry Gao, *San Jose State University, USA*
Du Zhang, *California State University, USA*

Program Committee

Public Relations

Industry Advisory Committee

Proceedings Cover Design

Conference Secretariat

Table of Contents

Note: (S) means short paper.

Keynote: A View on Software Specification Research and Education Advancement

Joseph E. Urban
National Science Foundation and Arizona State University, U.S.A.

The impact of software engineering research has been the steady improvement of approaches to software development and maintenance. Software requirements analysis, specification, and design methodologies have been formulated to aid in the development of reliable software. This talk will cover advancing software methodologies through research in the front-end aspects of software development. Advancement of these formalisms is necessary for improving software productivity, reliability, and development team dynamics. For integration with existing approaches, the incorporation of components with improved software development techniques and tools is needed for use with new programming languages. Exploring computer languages that are to be effective in the early stages of development is essential for major improvements in software systems. A more effective software engineering workforce will be addressed through upper level undergraduate and graduate education.

About Professor Joseph E. Urban

Dr. Joseph E. Urban currently serves as a program director in the U.S. National Science Foundation on an Intergovernmental Personnel Act mobility assignment within the Division of Computing and Communication Foundations of the Directorate for Computer & Information Science & Engineering. He is a professor of computer science at Arizona State University. He has worked at the University of Miami, the University of Louisiana at Lafayette, and part-time at the University of South Carolina while with the U. S. Army Signal Center.

Professor Joseph E. Urban has published over one hundred conference and journal papers. He has supervised the development of eight software specification languages. His research areas include software engineering, executable specification languages, prototyping software systems, web based software tools, engineering education, computer languages, data engineering, and distributed computing.

Dr. Joseph E. Urban earned a B.S. degree from the Florida Institute of Technology, an M.S. degree from the University of Iowa, and a Ph.D. degree from the University of Louisiana at Lafayette, all in Computer Science. He has received the Computer Society's Meritorious and Distinguished Service Awards, a Distinguished Professor Award while at the University of Louisiana at Lafayette, and an Association for Computing Machinery Doctoral Forum Award for one of the four best Ph.D. dissertations produced during the 1977-1978 academic year.

Constructing Self-Adaptive Systems with Polymorphic Software Architecture

Xiaoxing Ma, Yu Zhou, Jian Pan, Ping Yu and Jian Lu
State Key Laboratory for Novel Software Technology, Nanjing University, China
E-mail: xxm@nju.edu.cn

Abstract

Facing changing environment and user requirements, modern distributed software systems often have to evolve accordingly. We propose an extended object-oriented programming model for dynamically self-adaptive distributed software systems. With this model every component of a system is explicitly situated in an active architectural context, which is naturally used to regulate and facilitate potential runtime reconfiguration. The architecture context is explicitly implemented with a distributed shared object, whose state changes and polymorphic substitutions realize the anticipated and unanticipated architectural reconfigurations respectively. Thus the adaptation behavior specified at the architectural level can be automatically carried out. A prototypical supporting system is developed for the model.

1 Introduction

In an open, dynamic and uncertain environment such as the Internet and some pervasive computing settings, the network links, bandwidth, available resources and services, etc. are changing constantly. At the same time, applications working in such an environment are also facing an open group of users with different preferences, and even the requirement of the same user can evolve over time. To keep their service satisfactory, the application systems must be able to adapt themselves accordingly, at runtime and without significant disturbance to their operation [18].

To develop such self-adaptive systems is difficult [12]. In classic software development methodologies the support for system adaptability is limited. To make a system adaptable, developers have to foresee potential changes and treat these changes as a part of the requirement, explicitly or implicitly. Software development has been essentially regarded as a transformation from requirement model to implementation, and only the *result* of the transformation is finally put into operation. The decisions and deliberations directing the transformation process become implicit and dispersed in the implementation, although they can be documented somewhere else. This loss of information is a deep reason why to evolve the implemented system is hard and error-prone despite of the great flexibility of modern runtime system modification techniques such as computational reflection [17, 7], dynamic library loading and hot component deployment.

Contrastingly, an self-adaptive software system implementation itself has to stretch across a long semantic distance from abstract user-oriented policies up in the problem domain to concrete functional components down in the solution domain. Thus it is desirable to take a "high-order" view of software development and system adaptation: A system implementation put into operation is not only the result of the transformation process, but also some facilities of the *transformation* itself and the vulnerable part of requirement/environment specifications. With the built-in transformation facilities, the adaptation of the system is derived "on-the-fly" from the changes of corresponding part of requirement/evironment settings, which in turn are figured out at runtime with probed context information, user-specified adaptation polices, and possible direct human indications.

The problem is how to organize such transformation facilities. Here we do not mean full automation of software development, but a reflective computing [17] framework limited to the scope of required autonomy. Software architecture, which *"involves the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on these patterns"* [22], is a good candidate for the central abstraction of the framework. As the high-level design decisions that bridge the problem domain and the solution domain, software architecture embodies some most important information that should be retained in the implementation to support and regulate future adaptation and evolution.

To make the system self-adaptive, There must be a mechanism that causally connect the software architecture specification to the running environment – i.e. changes specified at the architectural level are always realized by the implementation "on-the-fly" and runtime status of the implementation (and related context) are always reflected up to the architectural level. Although substantial research on software architecture description languages (ADLs) exists [19], additional work is needed to causally connect software architecture to implementation.

In this paper a programming model directly reifying software architecture as an explicit object is proposed. An intrinsic causal connection between this object and the system implementation is established by dynamically reinterpreting the references between the component objects under current architectural configurations. Once the architecture is treated as a first class object, the architectural adaptation can be naturally expressed as the dynamic behavior of this object. Further, with such an reification, the software architecture becomes polymorphic, i.e., inheritance and polymorphism mechanisms can be applied on the architecture classes and objects. We can extend and refine the architecture class, and dynamically upgrade the software architecture by replacing the architecture object polymorphically. In the new architecture class, new reconfiguration behavior can be defined, and thus the whole application system can do some dynamically reconfiguration beyond the anticipation of their original developers. Despite the logically uniform view of the architecture object, it's physical implementation can be distritbuted.

The rest of the paper is organized as follows: Section 2 describes the reification of software architecture as runtime object and how to use it to support system adaptation. Section 3 gives a prototypical supporting system and a demo application. Related work is briefly discussed in Section 4 before we conclude the paper.

2 Software architecture as runtime object

A considerable amount of research efforts have been done to bridge the gap between architecture specification and implementation – architecture refinement [20], automatic generation of implementation from architecture specification [19], explicit maintaining of dynamic architectures [9, 15], architecture retrieval from working system [24], to name a few. However, software architectures were mainly viewed as design specifications rather than materialized and operational/functional entities in the final running systems. Although software architecture specifications can help the development and management of dynamic system adaptations [14, 16], as the upper part of Figure 1 shows, difficult and ad-hoc efforts are needed to maintain the consistency between a software architecture specification and the working system implementation.

To further ease the understanding, expressing and realization of dynamic adaptation at implementation level, the software architecture should be directly implemented at this level. Encoding the architecture specification into a data structure is not enough – it must be synchronized with the system's current configuration. But if this synchronization was done "by force", we just repeated the problem. More intrinsic mechanism is needed to make the reified software architecture (which is a runtime object) causally connected to the system implementation.

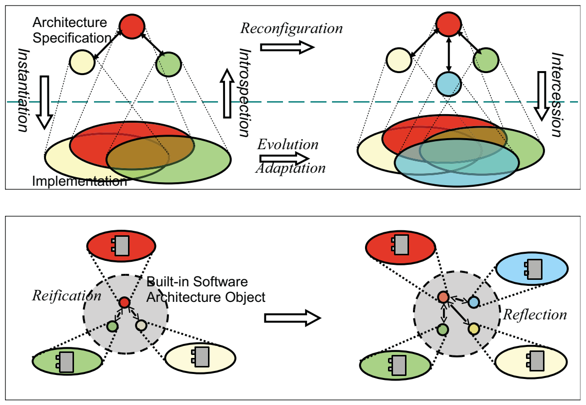


Figure 1. Software architecture reification

2.1 An intrinsic approach

Such a mechanism must be deep into the programming model. As we know, what glue running entities together in the object-oriented programming model are object references and associated method invocations, usually expressed like "`o.m()`". These references and method invocations are dispersed into the entities (i.e. object) and finally realize the overall structure and coordination logic of the system. With dynamic adaptation in mind, we can find following problems of this paradigm: First, eventually the reference "`o`" must refer to some concrete object before any method invocation "`m`" can be carried out. During the process of determining the referred object the system coordination structure is gradually consolidated with a loss of organizational and architectural information. Consider following scenarios: in a startup company, employees directly report to Bill the boss ("`o`" points to Bill). With the growth of the company, Tom, a project manager, is hired for project development affairs. The reference "`o`" should be redirected to Tom. But in the original architecture decision the one who is responsible for hearing report is not Bill, nor Tom, even not the boss role nor the project manager role, but the *current* role in charge of project development according to the *current* organization or architecture of the company. In the two scenarios the reference is eventually fixed to a specific value and the underlying architecture information is lost. Secondly, these references are distributed and hidden irregularly in the program entities (objects), which hinder the understanding and reconfiguration of the architecture. Also, the reusability of the entities is decreased because of the dispersed references. Software component models [23] and their supporting middleware[5] are helpful here as they eliminate the direct object references cross components, but themselves do not directly support the architectural issues at the implementation level [6].

With these considerations, a dynamic software architecture-oriented programming model (illustrated by the lower part of Figure 1) is proposed, which features:

- *Built-in runtime software architecture object* The software architecture concerns are separated from interacting component objects, and expressed explicitly as a first class object in the final implementation. The cross-component references are dynamically interpreted according to this architecture object. In other words, the references are "functions" over the current software architecture configuration. In this way the change of the architecture object will immediately affects the interaction between the components. Naturally anticipated dynamic reconfigurations are implemented as the behavior of the architecture object.

- *Unanticipated dynamic reconfiguration support* Once the software architecture is reified as an object, inheritance and typing mechanisms of object oriented programming model can be applied to architectural evolutions of the system. In addition to the planned reconfiguration just mentioned, some unanticipated reconfiguration can be implemented as new behavior of an architecture object whose class inherits the original's. With the help of dynamic library or class loading, the system's architecture object can be polymorphically replaced with the new one, and then the new reconfiguration behavior eventually carried out.

- *Distributed shared object implementation* The above discussion assumes a centralized architecture object, which is convenient for the developer to express the coordination logic structurally. However, the underlying implementation in the open network environment must be distributed flexibly. We adopt a distributed shared object mechanism: the dynamic architecture object is co-implemented with a group of coordinated sub-objects located at every node involved. Each sub-object provides a logically unified architectural context for the local component.

2.2 Application model

In our framework an application consists of a set of functional components, optional connectors, an architecture object, and a mapping between the component/connector instances and the internals of the architecture object.

Component objects A component implements some business functionality. It serves via its *provided interfaces* in condition that it is served via its *required interfaces*. Practically it can be a CCM component, an Enterprise JavaBean (EJB) or a Web Service.

Connector objects Connectors focus on non-functional aspects such as communication, security, reliability, logging, etc. They are optional in our framework for its implementation-oriented nature. Connectors are implemented as interceptors syntactically transparent to the components and do not affect the business logic of the application.

Architecture Object The architecture object implements the application's structural organization and related behavior. It's this object through which the components are finally connected together. It's also the locus where dynamic adaptation capabilities are realized. We will discuss how to define an architecture class shortly.

Mapping Component objects must be mapped to the component roles in the architecture object to get the necessary architectural context. Each required interface is fulfilled indirectly by a provided interface or interfaces (a multiplexer connector may be employed) of other components under the management of the architecture object. In practice the mapping can be defined with a graphical tool (cf. Figure 3 in Section 3). Syntactical type checking and even behavioral compliance checking can be included here.

2.3 Architecture class definition

The behavior of an architecture object is defined by its class. Architecture classes reify the concept of software architectural styles [22]. All architecture classes must inherit form a system class `RTArchitecture` directly or indirectly. `RTArchitecture` provides some basic functions for the development of specific architecture class, including: 1) basic architecture topology, which is merely a canonical programming-level representation of software architecture specification in ACME [10]; 2) redirection of the cross-component reference according to current architecture topology; 3) supports for the distributed implementation of the architecture object. Here some consistency insurance mechanisms from basic synchronization to two-phase commit protocol is needed. 4) basic reconfiguration activities, including addition/deletion of component roles and links between them, replacing of the component for a role.

An architecture class library can be provided by the development environment to support the reuse of common architectural styles. Developers derive their own architecture class from an existing class to best fit their application on hand. For example, a class of simple Master/Slave style can be declared rather straightforwardly as follows:

```
public class MSArch extends RTArchitecture
      implements ISlave, IMaster {
//methods declared in ISlave
//for slaves to pull jobs from the master
    public Object invokeOnMaster(Method m,
                                Object[] params)
        throws Exception{...};
//methods declared in IMaster -- omitted
    ... ...
//implementations for dynamic reconfiguration
    public void addSlave(SLAVE T){...};
    public void removeSlave(SLAVE T){...};
    ... ...
//Constructors
    public MSArch(ArchConfig ac){...};
    ... ...
}
```

The architecture class provides an interface for each of its players. The mapping tool will generate dynamic proxies with the method defined in this interface to fulfil the required interfaces of the associated component object. In this example a weakly typed method `invokeOnMaster` is provided to redirect calls from slaves to master to the proper component mapped to the master player. Here we assume the slaves pull jobs from the master. If the master need to push jobs to slaves, then `invokeOnSlaves` should be defined for `IMaster`. And also a multiplexer connector should be used to resolve the mismatching during the mapping process.

Suppose a Web-based application using this architecture. The component mapped to the Master is responsible for accepting user requests and presenting process results but distributing real work to the components mapped to Slaves. It's often required to dynamically add or delete Slaves to match the current work load. Since the object defines the architecture, dynamic reconfigurations are treated as the object's behavior. They are implemented as modification methods of the class. In `MSArch`, insertion and removal of Slaves are defined with methods `addSlave()` and `removeSlave()`. The implementation of these method is mainly changing the topology with the facilities provided in `RTArchitecture`. Remember the object implementation can be physically distributed, thus it often has to use the two-phase commitment support to ensure atomicity.

2.4 Unanticipated reconfigurations

It's natural to implement dynamic reconfigurations as the behavior of the architecture object. But these reconfigurations must be foreseen by the original architecture designer. There are some reconfiguration requirements gradually discovered after the system is put into operation. Common solutions for these unanticipated reconfigurations require shutdown of the working system to upgrade it. For certain applications, the stop of service is unacceptable or too expensive.

Our approach also provides a reasonable support for unanticipated dynamic reconfigurations. A new subclass of the original architecture class will be defined to implement new reconfiguration behavior. As to the Master/Slave application discussed above, with more and more slaves added in, the master itself is overloaded, which is beyond the anticipation of the original system architect. Now the architecture should be evolved to a new style of Extend Master/Slave which also support multiple Masters. Therefore a new architecture class `EMSArch` is defined, in which additional Masters can be added in, and related interactions are adjusted accordingly:

```
public class EMSArch extends MSArch
      implements IEMaster {
//New reconfiguration behavior
    public void addMaster(MASTER M){...};
    public void removeMaster(MASTER M){...};

//Redefined behavior. Some load balancing
//can be implemented here.
    public Object invokeOnMaster(Method m,
                                Object[] params)
        throws Exception{...};

//methods defined in IEMaster to support
//coordination among masters -- omitted
    ......
}
```

Hence the new behaviors of adding and removing Masters are defined. At the same time, to make a smooth switching from the old architecture to the new one, the new architecture object should be able to act as the old one to untouched parts of the working system, but with new semantics of the new architecture. So some behaviors defined in the old architecture class must also be redefined in the new class. With this new class, a new architecture object can be instanced and initialized with the current state of the old architecture object. With the polymorphic substitution of the new architecture object for the old one, the system is dynamically upgraded and then new reconfiguration behavior can be carried out.

2.5 Driving the adaptation

To make the system self-adaptive, two more facilities are needed. One is a set of sensors deployed to monitor and report the current status of the environment and the system itself, such as network bandwidth, response delay, processor workloads, current user preferences, etc. This information is called *context*, which is a topic extensively researched in the field of context-aware computing [8]. The other is a decision-making mechanism that drives architectural reconfiguration commands according to the specified polices, current system settings and gathered context information. It involves related user requirements, such as an online commerce system should be "responsive", the domain knowledge, such as to be "responsive" the user experienced latency should not be more than 2 seconds, the architecture properties and available reconfiguration behaviors, such as for a system with the aforementioned Master-Slave architecture style generally the response time can be reduced with the addition of new slaves.

In current practices these facilities are often implemented in *ad hoc* ways. To build a reusable and flexible framework we are currently using an ontological approach. A spectrum of OWL [2] ontologies are designed to represent the low-level context information, architecture configurations, behaviors and properties, user requirements, and related domain knowledge respectively. Then the decision mechanism is built on the standard and customized reasoning on these ontologies. Contrasting to hard-coded decision components for self-adaptation, this multi-ontology mechanism enables knowledge learned afterward to be naturally included in at runtime. Together with the unanticipated architecture reconfigurations discussed earlier, it provides a reasonable support for online evolution of self-adaptive systems. For the limited space, a full discussion about this mechanism is left to a subsequent paper.

2.6 Discussion

It's worthy to note that the dynamic reconfiguration must not compromise the consistency of the system. Under our approach, the consistency constraints that only depend on architectural states should be regarded as architecture class invariants and respected in the implementation of adaptation behaviors. In fact a variety of formal models from graph grammar [16] to process algebra[14, 4] and corresponding model checking techniques can be incorporated in. The built-in runtime object provide a ready basis for the checking.

The constraints depended on the states of the components are more troublesome and often application specific. Developers can handle these issues with derived architecture class with specific consistency checking and recovering mechanisms. we are currently exploring techniques from code-level dynamic software updating [11] to component-level blocking/quietening[14, 26] for a solution best fit for our reified software architecture.

3 A prototypical supporting system

To enable software development and evolution with foresaid ideas and techniques, a prototypical supporting system is developed. As shown in Figure 2, the system is built around the reified software architecture. Integrating several well-known open source systems and self-programmed components, it makes an initial step toward an integrated environment for the visual development, deployment, monitoring, adaptation of self-adaptable and online evolvable software systems.

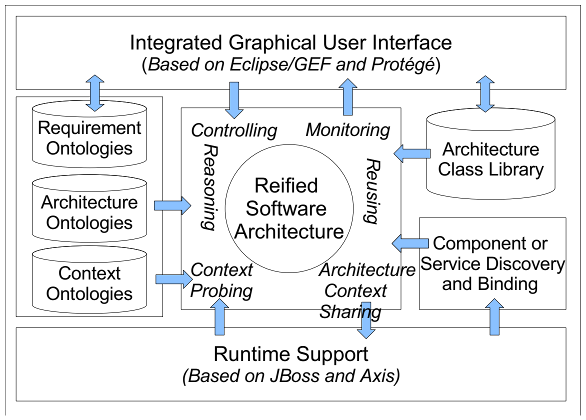

Integrated Graphical User Interface
(Based on Eclipse/GEF and Protégé)
Requirement Ontologies
Architecture Ontologies
Context Ontologies
Controlling
Monitoring
Reasoning
Reusing
Reified Software Architecture
Context Probing
Architecture Context Sharing
Architecture Class Library
Component or Service Discovery and Binding
Runtime Support
(Based on JBoss and Axis)

Figure 2. Conceptual structure of Artemis-MAC

The system is programmed in Java, and closely integrated with the Eclipse platform. Its main functions include support for

- *a distributed component framework*. The component model used is essentially EJB, but with explicitly specified required interfaces. Web Services can also be used. Discovery and binding of EJBs or Web Services are supported. The component framework are mainly implemented with some modifications upon the JBoss AS and the Axis package.
- *architecture-centric component composition*. Through the graphical architecture editing tool implemented with the Graphical Editing Framework(cf. Figure 3), developers can graphically specify the concrete configuration of the application architecture with a chosen architecture class, and mapping the EJBs/Services discovered to the players in the architecture. The system will automatically generate the distributed shared architecture object.
- *dynamic self-adaptation and online evolution*. A Jena based reasoning engine is used to infer and trigger proper reconfiguration actions (including architecture object upgrading) with user-specified rules, context information and architecture properties expressed in OWL ontologies. These ontologies are developed with the OWL editor Protégé [13]. Containers for context probes and monitors are also provided. The lower part of Figure 3 shows a visual monitor for response time.

Figure 3 shows a demo application with the Master-Slaves architecture discussed above running in the support-

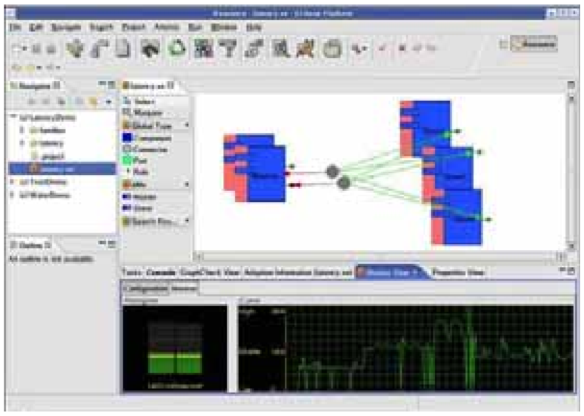


Figure 3. Artemis-MAC User Interface

ing environment. Suppose there are various independent ticket-booking services scattered over the network offering different kinds of tickets, such as plane, train, bus, et al. Because of the similarity of their business logic, these services can be abstractly unified and integrated in a comprehensive service – those independent services are mapped to the slaves and the comprehensive service mapped to the master. When requests come, they will be handled by the master and dispatched to the slave services. After processing the results would be synthesized and returned to users by the master. In this way, users can find out possible trip routes once for all, instead of checking out those independent sites one by one. In developing such an application, it's easy to predict that latterly more services could be added in, and sub-optimal services could be dropped out. So the addition/removal of slaves are implemented as the behavior of the built-in architecture object. But as the comprehensive service become more and more popular, the master becomes the potential performance bottleneck. As discussed before, an upgraded architecture class is derived and corresponding architecture object is instantiated and initialized with the current object. Also, associated ontologies are enriched with the knowledge that upgrading the architecture object is needed to add new masters to further reduce response time. Then later, with the new object and enriched ontologies, upgrading of the architecture and addition of a secondary master are triggered when needed. The performance monitor in the lower part of Figure 3 shows this adaptation restores the response time to an acceptable level.

4 Related work

To support self-adaptation of software systems, Garlan and others built a runtime software architecture based framework called Rainbow [9]. Rainbow uses accurate and up-to-date architectural information to help the monitoring and adaptation of a running system. However in Rainbow the runtime architecture representation is alien to the running application system, and thus the modification to the architecture representation can not affect the running application system itself. Ad-hoc mechanisms must be developed to causally connect the architecture representation to the running application system. In our approach the architecture object is built-in as an integral part of the running application system, and the causal connection between the object and the system becomes more intrinsic. In this way we hope that the architectural concerns can be expressed more explicitly and systematically with the built-in architecture object, the coupling between component computation and architectural coordination can be further decreased, and dynamic adaptations, esp. unanticipated dynamic evolution can be understood and implemented more naturally.

To bridge the gap between software architecture and system implementation, Aldrich designed ArchJava [1], an extension to Java, that supports direct expressing of architectural structure at programming level. Implementation's conformance to architecture concerns is ensured by the type system. But ArchJava provides little support for dynamic software architectures.

The ArchWare European project [21, 3] aims at an integrated set of architecture-centric languages and tools for the model-driven engineering of evolvable software systems. An ADL based on high-order typed π-calculus plays a central role in their approach. One of its novelty is to separate functionality and evolution. While such kind of separations do favor developers at the modeling level, we believe an intrinsic connection mechanism as discussed in this paper is also necessary at the implementation level. The strength of ArchWare on the formal modeling and reasoning of software architecture and architecture adaptation is also attractive to us that we are exploring the possibility of using it in our system.

5 Conclusions

There is an increasing need for the self-adaptation and online evolution of software systems[18, 25]. Ad hoc solutions are not satisfactory to software engineers. In this paper a polymorphic software architecture based approach has been presented as a first step toward a systematic and disciplined software development method for self-adaptive and online-evolvable software systems. A prototypical supporting system and a demo application have also been developed.

Our plan for further research includes applying the approach to a more realistic application, developing a practical and reusable consistency-keeping mechanism for runtime system adaptation, and enriching the ontologies for practical use and optimizing the engines for better runtime reasoning.

Acknowledgements

The research is co-supported by NSFC (60403014), JSNSF (BK2006712), China 863 Program (2006AA01Z159) and 973 Program (2002CB312002).

References

[1] J. Aldrich. *Using Types to Enforce Architectural Structure*. PhD thesis, University of Washington, August 2003.

[2] G. Antoniou and F. van Harmelen. Web ontology language: Owl. In S. Staab and R. Studer, editors, *Handbook on Ontologies*, pages 67–92. Springer-Verlag, 2004.

[3] D. Balasubramaniam, R. Morrison, G. Kirby, K. Mickan, B. Warboys, I. Robertson, B. Snowdon, R. M. Greenwood, and W. Seet. A software architecture approach for structuring autonomic systems. In *DEAS '05: Proceedings of the 2005 workshop on Design and evolution of autonomic application software*, pages 1–7, New York, NY, USA, 2005. ACM Press.

[4] M. Bernardo, P. Ciancarini, and L. Donatiello. Architecting families of software systems with process algebras. *ACM Trans. Softw. Eng. Methodol.*, 11(4):386–426, 2002.

[5] P. A. Bernstein. Middleware: A model for distributed system services. *Communications of the ACM*, 39(2):86–98, February 1996.

[6] G. S. Blair, L. Blair, V. Issarny, P. Tuma, and A. Zarras. The role of software architecture in constraining adaptation incomponent-based middleware platforms. In *Middleware '00: IFIP/ACM International Conference on Distributed systems platforms*, pages 164–184, Secaucus, NJ, USA, 2000. Springer-Verlag New York, Inc.

[7] F. Demers and J. Malenfant. Reflection in logic, functional and object-oriented programming: a short comparative study. In *Proceedings of the IJCAI'95 Workshop on Reflection and Metalevel Architectures and Their Applications in AI*, pages 29–38, 1995.

[8] A. K. Dey, G. D. Abowd, and D. Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction (HCI) Journal*, 16 (2-4):97–166, 2001.

[9] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[10] D. Garlan, R. T. Monroe, and D. Wile. Acme: Architectural description of component-based systems. In G. T. Leavens and M. Sitaraman, editors, *Foundations of Component-Based Systems*, pages 47–68. Cambridge University Press, 2000.

[11] M. Hicks and S. Nettles. Dynamic software updating. *ACM Trans. Program. Lang. Syst.*, 27(6):1049–1096, 2005.

[12] J. O. Kephart. Research challenges of autonomic computing. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 15–22, 2005.

[13] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen. The protégé owl plugin: An open development environment for semantic web applications. In *ISWC '04 Proceedings of the Third International Semantic Web Conference*, volume 3298 of *LNCS*, pages 229–243, 2004.

[14] J. Krammer and J. Magee. Analysing dynamic change in distributed software architectures. *IEE Proceedings-Software*, 145(5), 1998 1998.

[15] L. Lan, G. Huang, L. Ma, M. Wang, H. Mei, L. Zhang, and Y. Chen. Architecture based deployment of large-scale component based systems: the tool and principles. In *CBSE '05: Proceedings of the8th International SIGSOFT Symposium on Component-based Software Engineering*, pages 123–138. Springer, 2005.

[16] D. Le Metayer. Describing software architecture styles using graph grammars. *IEEE Transactions on Software Engineering*, 24(7):521–533, July 1998.

[17] P. Maes. Concepts and experiments in compuational reflection. In *Proc. of OOPSLA'87*. ACM, Oct. 1987.

[18] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng. Composing adaptive software. *Computer*, 37(7):56–64, 2004.

[19] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transaction on Software Engineering*, 26(1):70–93, January 2000.

[20] M. Moriconi, X. Qian, and R. A. Riemenschneider. Correct architecture refinement. *IEEE Transactions on Software Engineering*, 21(4):356–372, April 1995.

[21] F. Oquendo, B. Warboys, R. Morrison, R. Dindeleux, F. Gallo, H. Garavel, and C. Occhipinti. Archware: Architecting evolvable software. In *Proceedings of the 1st European Workshop on Software Architecture, LNCS*, volume 3047, pages 257–271, 2004.

[22] M. Shaw and D. Garlan. *Software Architecture: Perspective on an emerging discipline*. Prentice Hall, 1996.

[23] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 2nd edition, 2002.

[24] A. van Deursen, C. Hofmeister, R. Koschke, L. Moonen, and C. Riva. Symphony: view-driven software architecture reconstruction. In *WICSA'04: Proceedings of the Fourth Working IEEE/IFIP Conference on Software Architecture*, pages 122–132, June 2004.

[25] P. Yu, X. Ma, and J. Lu. Dynamic software architecture oriented service composition and evolution. In *CIT '05: Proceedings of the The Fifth International Conference on Computer and Information Technology*, pages 1123–1129, Washington, DC, USA, 2005. IEEE Computer Society.

[26] J. Zhang and B. H. C. Cheng. Model-based development of dynamically adaptive software. In *ICSE '06: Proceeding of the 28th international conference on Software engineering*, pages 371–380, New York, NY, USA, 2006. ACM Press.

[27] Y. Zhou, J. Pan, X. Ma, et al. Applying ontology in architecture-based self-management applications. In *SAC'07: Proceedings of the 22nd Annual ACM Symposium on Applied Computing*, Seoul, Korea, March 2007. ACM.

Odyssey-MDA: A transformational approach to component models

Natanael E. N. Maia [1], Ana Paula Terra Bacelo [1, 2], Cláudia M. Werner[1]
[1]*COPPE/UFRJ – Programa de Engenharia de Sistemas e Computação*
Caixa Postal 68.511 – CEP. 21945-970 – Rio de Janeiro – RJ - Brazil
[2]*Pontifícia Universidade Católica do Rio Grande do Sul – PUCRS-FACIN*
Av. Ipiranga, 6681 – Prédio 32 – CEP. 90619-900 – Porto Alegre – RS - Brazil
{ ntmaia, anablois, werner}@cos.ufrj.br

Abstract

This paper presents a MDA based approach to support the developer in obtaining platform specific component models. These models are obtained through the definition and execution of transformations which can be defined by the composition of small generic transformations (built-ins) or implemented and incorporated by an extension mechanism (plug-ins). To support the use of this approach, a machine for model transformation execution was implemented. An example of its use for the EJB platform is presented.

1. Introduction

Software house organizations have recognized the importance of reuse politics during the systems development to decrease costs, optimize resources and increase the productivity and software quality. Component-based development (CBD) has been considered a software reuse approach through which it is possible to obtain such benefits. The CBD area aims at the building of systems based on reusable software components [11].

New technologies are continuously being introduced and some of them become popular. A challenge for software development teams is to choose technologies for specific applications. Several software houses adopt new technologies for different reasons: a) their clients require the use of new technologies; b) these technologies may solve problems for these clients, and c) some technologies become obsolete and a support is no longer a viable option. As a consequence, the components of the obsolete technologies need to be migrated to a new technology or to a new version of the existent technology [5].

A possible solution to this problem is proposed by OMG, through the Model Driven Architecture (MDA) framework [7]. The goal of MDA is to propose an approach: a) to specify a system independently of the implementation platform; b) to specify the possible platforms for the systems that are under development; c) to choose the platform for a system; and d) to transform the initial specifications to obtain the system specification in the chosen platform. Therefore, the developers invest more time in the requirements modeling activities and spend less time with their implementation issues.

This paper describes an approach for the definition and execution of transformations on component models. The approach allows independent component models to be transformed into specific models in a chosen platform. The transformations may be defined by the composition of small generic transformations (built-ins) or implemented and incorporated through extension mechanisms (plug-ins). The approach allows the definition and execution of bidirectional transformations. Besides, it is independent of the development environment, using XMI format

The paper is organized as follows: Section 2 describes some standards adopted in this work. Section 3 presents the approach for the definition and execution of transformations on component models - Odyssey-MDA. Section 4 describes the implementation of the approach by the Odyssey-MDA transformation machine. Section 5 presents an example of transformation using the Enterprise JavaBeans platform. Section 6 presents related works and lastly, section 7 concludes the paper and presents the main contributions of Odyssey-MDA and future works.

2. Adopted Standards

The proposed approach is based on Meta-Object Facility (MOF) [8], XML Metadata Interchange (XMI)

[13] and Java Metadata Interface (JMI) [4] standards. The goal is to use these standards to create a more flexible approach to import, transform and export these models, and to be extended in order to implement and incorporate new transformations. The MOF standard specifies abstract languages to describe other languages and, in this case, it is described as a meta-meta-model (M3 level). The abstract languages based on MOF are known as meta-model (M2). An example of these abstract languages is the UML meta-model, where each UML model is part of M1 level.

The XMI standard allows the interchange of data amidst UML based modeling tools with the repository of the MOF based models into heterogeneous and distributed environments.

The JMI specification allows the construction of interfaces in the Java Language for a specific MOF meta-model (M2), in order to make the manipulation of the instances of the meta-model possible. For example, the UML meta-model is described as MOF. Through JMI it is possible to manipulate the elements of UML (class, interface, attribute …) as Java objects, by using the existing interfaces. This method may be adopted not only by UML, but also by any MOF based meta-model. MDR [6] is a repository that implements MOF and JMI standards allowing the conveyance of these MOF meta-models and the maintenance of the instances of the meta-models. The instances maintained into the repository are systematically manipulated, using the interfaces that were created by the JMI standard.

3. Odyssey-MDA Approach

The Odyssey-MDA approach allows the developer to transform models that are independent of a platform into specific models of a particular platform (e.g. EJB). The first models are known as PIM (Platform Independent Models) and the latter as PSM (Platform Specific Model). Through a PSM it is possible to obtain the implementation of components in a specific language (e.g. EJB components implemented in Java). The transformation process is composed of the following steps:

a) Definition of the Platforms and Transformations – The developer defines the necessary platforms and transformations to create a PSM based on the transformation of a PIM.
b) The initial modeling of the components – The components modeling is performed in an independent way, where these components will be implemented in the future.
c) Marking the internal components model – The elements of the internal model (class model) are prepared with a marking mechanism to guide the execution of the consecutive transformations.
d) Choice of the platform and execution of transformations – The developer chooses the platform previously defined. Transformations are applied on the model previously marked to obtain the PSM for the chosen platform.
e) Implementation development – The source-code of the PSM can be built, using the programming language of the platform.

Figure 1 shows a typical scenario of the use of the Odyssey-MDA approach. The developer builds the initial model in a CASE tool or other modeling environment. Through a CASE Tool, a modeling environment or a reuse environment, the developer must perform the marking of the elements of the components internal model (class model), using stereotypes and tagged-values. This is important for the selection of the most adequate mappings aiming at the transformation of these elements into other elements of the output model. The possible set of stereotypes and tagged-values is defined according to the goal of the transformation to be executed and it can be based on

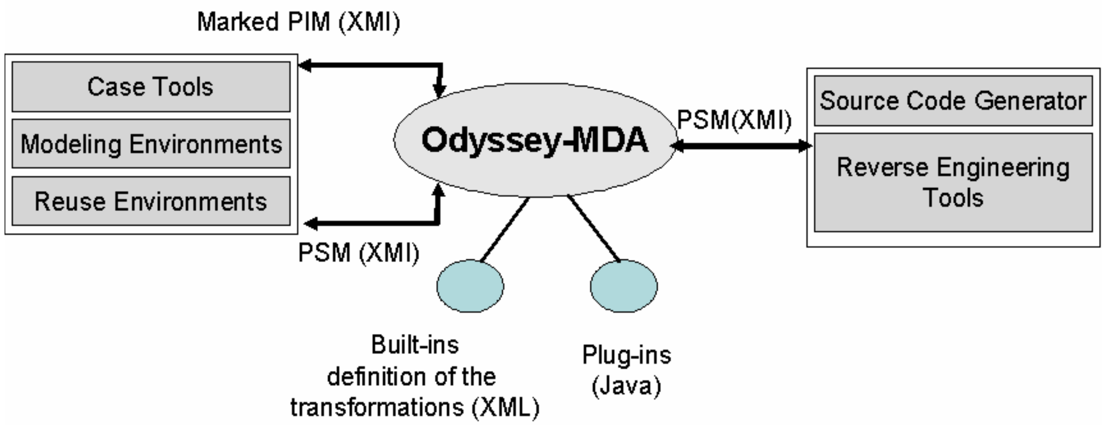


Figure 1: Typical Scenario of the Odyssey-MDA Approach

some UML profile such as UML Profile For EDOC [2]. The developer must export the original model (marked PIM), in XMI format and import it into the Odyssey-MDA transformation machine. In Odyssey-MDA, the transformations can be executed on the original model, resulting in an output model (PSM). The generated PSM can also be exported in XMI format for future importation in the initial CASE tool or transfer to some source code generator.

The Odyssey-MDA approach allows the definition and execution of bidirectional transformations. In this case, the developer can build a PSM based on a PIM (forward transformation), to make refinements and updates on the PSM and propagate these updates on the original PIM (reverse transformation). This propagation is extensive, once the original elements of the output model are updated by the transformations and it is not modifiable. Other ways to solve the conflicts related to the synchronization are not currently supported by the Odyssey-MDA approach [3].

Another reverse transformation scenario is the extraction of platform independent information of a PSM model, or a source code, for a PIM creation. This scenario may be useful in cases where a component, originally modeled and implemented in the specific technology, needs to be migrated to another technology.

4. Odyssey-MDA – Prototype

Each defined transformation is composed by a XML declarative specification and a set of mechanisms. The declarative specification is responsible for the definition of the mappings among the input and output models. These mappings are defined through *finders* for the selection of the elements to be transformed and by the attribution of a mechanism responsible for the execution of the transformation of these elements.

Finders are responsible for the selection of the output model elements. The elements may be selected by name, stereotype and tagged-values. For example, all Classes that have <<input>> stereotype or all classes with value “true” in the “persistent” tagged-value. The mappings are specific for each element to be mapped. To map subtypes of classified in UML meta-model (e.g: class, interface), the classifier-map mapping is proposed. To map subtypes of features (e.g. attributes and methods) the feature-map sub-mapping is proposed. To map a classifier into a feature, or vice-versa, the sub-mapping classifier-feature-map is proposed.

The Odyssey-MDA transformation machine proposes an infra-structure of generic mechanisms, known as built-ins, which executes simple transformations on UML elements. Table 1 presents the proposed built-ins and the transformations that are executed by each one. For example, the Classes of a PIM model may be mapped into Classes, interfaces, attributes and methods of the PSM model, as proposed by 6, 7, 8 and 9 built-ins of Table 1, respectively.

As mentioned before, the built-ins are generic. In this sense, there must be a way to configure the manipulated elements for these built-ins and, as a consequence, to control the behavior of the transformation. For instance, to transform a class into an interface (ClassInterface), their attributes, which are defined in the UML meta-model, need to be informed for the built-in through a set of properties. Table 2 presents the possible properties for the configuration of the built-ins.

When some textual transformation about the name of the elements is necessary, the nameTransformation property may be used to make such transformations (updates) using regular expressions. For example, it is possible to transform the name “attribute” into “getAttribute” (forward transformation), or the name “setAttribute” into “attribute” (reverse transformation). In section 5 an example of these properties is presented. Each mechanism implements the Transformation interface to allow the execution of the bidirectional transformations (PIM<->PSM). The Transformation interface defines the transformations in both directions, through the transformationLeftToRight and transformationRightToLeft operations. For instance, considering the ClassInterface mechanism,

Table 1. Transformations of the Generic Mechanisms (built-ins)

Built-in	***Transformation***	***Built-in***	***Transformation***
1. ComponentComponent	Component↔ Component	9. ClassOperation	Class ↔ Operation
2. ComponentClass	Component ↔ Class	10. InterfaceInterface	Interface ↔ Interface
3. ComponentInterface	Component ↔ Interface	11. InterfaceAttribute	Interface ↔ Attribute
4. ComponentAttribute	Component ↔ Attribute	12. InterfaceOperation	Interface ↔ Operation
5. ComponentOperation	Component ↔ Operation	13. AttributeAttribute	Attribute ↔ Attribute
6. ClassClass	Class ↔ Class	14. AttributeOperation	Attribute ↔ Operation
7. ClassInterface	Class ↔ Interface	15. OperationOperation	Class ↔ Operation
8. ClassAttribute	Class ↔ Attribute		

Table 2. Possible properties for the configurations of the built-ins

Manipulated Element	***Properties***
Component	isRoot, isLeaf, isAbstract, isActive
Class	isRoot, isLeaf, isAbstract, isActive
Interface	isRoot, isLeaf, isAbstract
Attribute	ownerscope, changeability
Operation	ownerscope, isQuery, concurrency, isRoot, isLeaf, isAbstract, parameters
All elements	name, nameTransformation, stereotype, taggedValues, visibility, isSpecification

the transformationLeftToRight operation gets a class as parameter and returns the Interface built from this class. On the other hand, the reverse operation transformationRightToLeft gets an Interface and returns a class.

The built-ins available in the transformation machine have been enough for the definition of the transformations among supported types so far. However, there may be some cases where the developer needs some transformation which is not available. In these cases, he may develop his own mechanism (plug-in) which must implement the Transformation interface and manipulate the necessary elements through JMI interfaces.

5. Example

An example of transformation is presented to illustrate the functionalities of Odyssey-MDA. In this example, an independent class model of a component (PIM) is received and a dependent class model (PSM) that represents the components for the EJB (Enterprise JavaBeans) platform is built.

Figure 2 presents the result of the transformation executed on the PIM model received as input. The Client class was previously marked with <<Entity>> stereotype and, in accordance to the defined standards of the EJB platform, the initial class was transformed into the ClientBean class and the Client and ClientHome interfaces of the PSM output model. The updates from the ClientBean class may be propagated, through the reverse transformation, to the Client class of the initial PIM. In this case, the transformationLeftToRight operation of the Transformation interface must be executed.

Figures 3 and 4 illustrate the declarative area of the transformation, i.e., a XML document with the declaration of the mapping among elements, the finders (Figure 3) and the configuration properties (Figure 4). The specification of the transformation mapping (transformation-map) is composed by mappings of the classifier-map types. Each mapping has finders which are responsible for selecting the elements into input and output models, according to the side attribute. Figure 3 presents the specification of a classifier-map (Entity-EntityBean) that uses the ClassClass generic mechanism. This mechanism transforms all identified classifiers by finder, which is responsible for selecting the marked elements with the <<Entity>> stereotype in the PIM model. Figure 3 also illustrates other examples of mappings such as feature-map. This mapping is responsible for performing the copy of the attributes of Entity element into the EntityBean and to build the get and set methods for each attribute of the input element.

Figure 5 shows the graphic interface of the Odyssey-MDA transformation machine. The tree presented on the left hand side of the interface shows the PIM model, initially developed and marked in a CASE Tool and imported via XMI mechanism. After the choice of the desired transformation, the user may select the

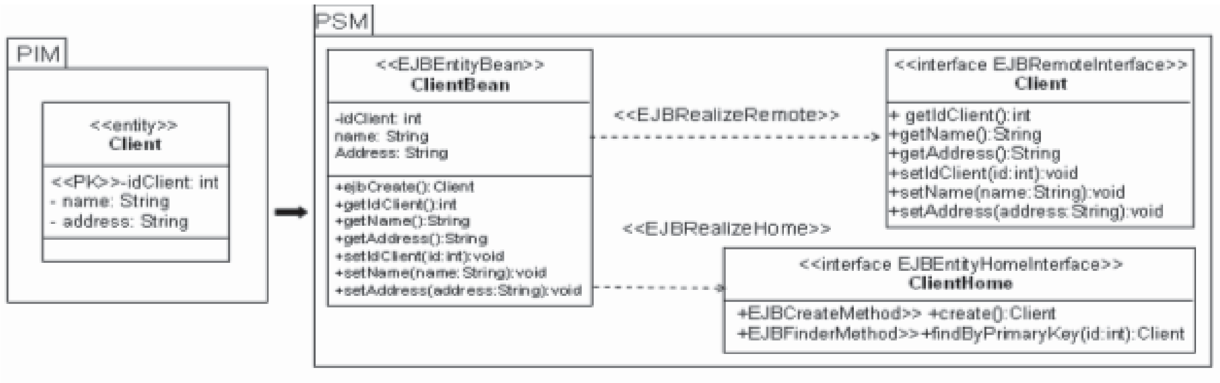
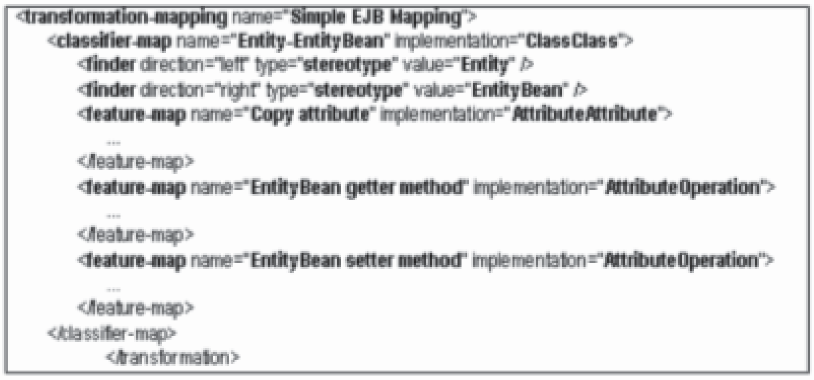
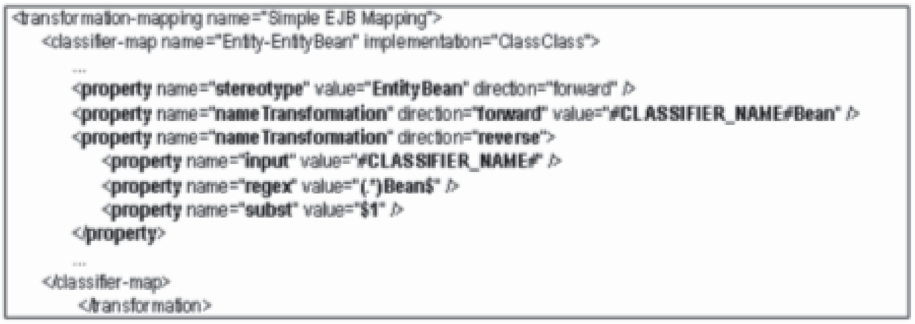


Figure 2. The result of the transformation executed on the PIM



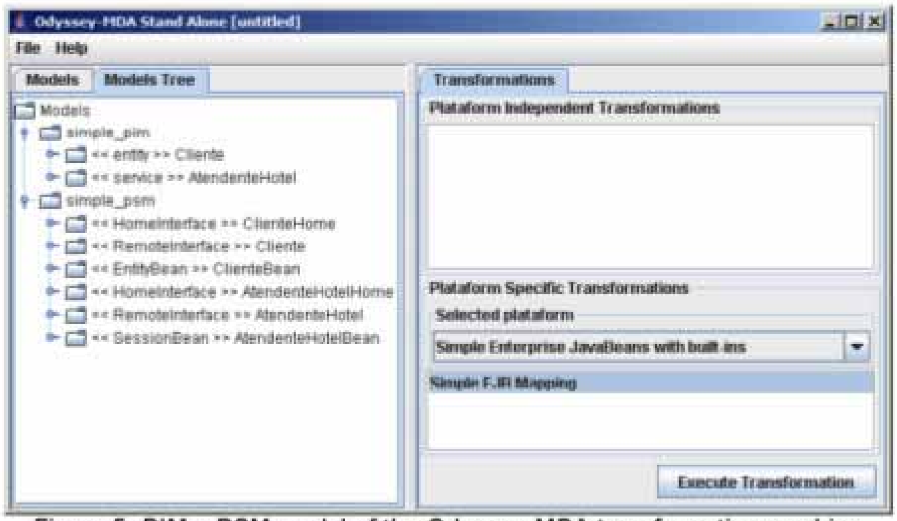
```
<transformation-mapping name="Simple EJB Mapping">
    <classifier-map name="Entity-EntityBean" implementation="ClassClass">
        <finder direction="left" type="stereotype" value="Entity" />
        <finder direction="right" type="stereotype" value="EntityBean" />
        <feature-map name="Copy attribute" implementation="AttributeAttribute">
            ...
        </feature-map>
        <feature-map name="EntityBean getter method" implementation="AttributeOperation">
            ...
        </feature-map>
        <feature-map name="EntityBean setter method" implementation="AttributeOperation">
            ...
        </feature-map>
    </classifier-map>
            </transformation>
```

Figure 3. Part of the specification of a classifier-map that uses the ClassClass generic Mechanism



```
<transformation-mapping name="Simple EJB Mapping">
    <classifier-map name="Entity-EntityBean" implementation="ClassClass">
        ...
        <property name="stereotype" value="EntityBean" direction="forward" />
        <property name="nameTransformation" direction="forward" value="#CLASSIFIER_NAME#Bean" />
        <property name="nameTransformation" direction="reverse">
            <property name="input" value="#CLASSIFIER_NAME#" />
            <property name="regex" value="(.*)Bean$" />
            <property name="subst" value="$1" />
        </property>
        ...
    </classifier-map>
        </transformation>
```

Figure 4. Part of the EJB mapping specification (configuration properties of the ClassClass built-in)



```
Models
  simple_pim
    << entity >> Cliente
    << service >> AtendenteHotel
  simple_psm
    << HomeInterface >> ClienteHome
    << RemoteInterface >> Cliente
    << EntityBean >> ClienteBean
    << HomeInterface >> AtendenteHotelHome
    << RemoteInterface >> AtendenteHotel
    << SessionBean >> AtendenteHotelBean
```

Selected plataform: Simple Enterprise JavaBeans with built-ins

Simple EJB Mapping

Odyssey-MDA Stand Alone [untitled]
File
Help
Models
Models Tree
Transformations
Plataform Independent Transformations
Plataform Specific Transformations
Execute Transformation

Figure 5. PIM e PSM model of the Odyssey-MDA transformation machine

input and output models and the direction of the transformation (forward or reverse). The transformation may be executed and the configurations and the resulting PSM model may be exported in a XMI format to a source code Tool Generator such as Odyssey-MDA-codegen [10] or some other CASE Tool.

6. Related Works

Considering the existing approaches related to the model transformation, Model Transformation Framework [9] and UML Model Transformation [12] are the most relevant for this research.

MTF is composed by a set of tools that support the developer during the comparison, consistency checking and implementation of transformations among UML models. The infra-structure of storage and manipulation of models is performed on EMF (Eclipse Modeling Framework). Odyssey-MDA is different from MTF since it manipulates models developed into any CASE tool or in any developing environment that uses MOF/JMI/XMI standards. UMT is a tool that transforms and generates source-code to UML/XMI models. The transformations in UMT are defined in XSLT/Java and manipulate models through XMI simplified representations (XMI Light). Besides, UMT does not support bi-directional transformation definitions, as proposed by Odyssey-MDA.

Czarnecki and Helsen [1] propose a classification of other existing approaches to models transformation, organized in five groups: direct manipulation, relational, graph-based transformations, structure directed and hybrids. Odyssey-MDA can be considered a hybrid approach because it combines the characteristics of structure directed and direct manipulations, by using built-ins and plug-ins, respectively.

Besides MTF and UMT, other approaches were analyzed, such as: Atlas Transformation Language [15], UMLx [16], AndroMDA [17] and OptimalJ [14].

7. Final Considerations

This paper presented the Odyssey-MDA which allows that components model created independently of any platform be transformed into specific models for a platform. The approach is supported by the Odyssey-MDA transformation machine, developed in Java with the following characteristics: (1) independent of the development environment, using XMI format; (2) the possibility of the execution of bi-directional transformations; and (3) extension facilities through generic mechanisms (built-ins) and the possibility of the definition of new transformations (plug-ins).

Some limitations of Odyssey-MDA include: (1) the restriction of UML model transformations; (2) the use of a language which is not totally compatible with QVT proposed by OMG to define the transformations; (3) lastly, there is the impossibility to define transformations of behavior models and, as a consequence, to create a source-code based on PIM.

Currently, Odyssey-MDA prototype uses UML models but it might be possible to transform any model based on MOF. The generated EJB may be used with other source-code and reverse engineering tools, representing an agile solution for software development. A case study to evaluate Odyysey-MDA in an industrial setting is being planned.

References:

[1] Czarnecki, K., Helsen, S. (2003) "Classification of Model Transformation Approaches". Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, USA, 2003.

[2] EDOC (2004) UML Profile For Enterprise Distributed Object Computing (EDOC) specification, OMG.

[3] Gardner, T., Griffin, C., Koehler, J., Hauser, R. 2003.A Review of OMG MOF 2.0 Query / Views / Transformations. OMG Document.

[4] JCP (2006) Java Metadata Interface (JMI) API 1.0 Specification,Available at: http://jcp.org/en/jsr/detail?id=040, 23/09/2006.

[5] Kleppe, A., Warmer, J., Bast, W. (2003) MDA Explained: The Model Driven Architecture – Practice and Promise. Boston, MA, Addison-Wesley, 2003.

[6] Matula, M. (2003) NetBeans Metadata Repository, Available at http://mdr.netbeans.org, 23/09/2005.

[7] MDA (2003) Model Driven Architecture, OMG, Available at http://www.omg.org/mda, 23/09/2005.

[8] MOF (2002) Meta Object Facility (MOF) specification, version 1.4, OMG.

[9] MTF (2004) Model Transformation Framework, Available at http://www.alphaworks.ibm.com/tech/mtf, 23/09/2005.

[10] Odyssey (2007) Available at http://reuse.cos.ufrj.br/odyssey, 7/03/2007.

[11] Sametinger, J. (1997) Software Engineering with Reusable Components. Springer, 1997.

[12]UMT (2004) UMT-QVT UML Model Transformation Tool, Available at http://umtqvt.sourceforge.net, 23/09/2005.

[13] XMI (2002) XML Metadata Interchange (XMI) specification, v1.2, OMG.

[14] COMPUWARE, 2006, "OptimalJ - Model-driven Java development tool". Available at: http://www.compuware.com/products/optimalj/, 20/12/2006.

[15] Jouault, F., Kurtev, I., (2005), "Transforming Models with ATL". In: Proceedings of the Model Transformations in Practice Workshop at MoDELS 2005, Montego Bay, Jamaica, October, 2005.

[16] Eclipse (2006). "UMLX: A graphical transformation language for MDA". Available at: http://dev.eclipse.org/viewcvs/indextech.cgi/~checkout~/gmt home/subprojects/UMLX/index.html, 0/02/2006.

[17] Kozikowski, J. (2005), "A Bird's Eye view of AndroMDA". In: http://www.andromda.org/contrib/birds-eye-view.html, 20/01/2006.

Acknowledgements:

The authors would like to thank CAPES and the Research Group in CDPe/PQTI of the TA002/2007, financed by Dell Computers of Brazil Ltd. with resources of Law 8.248/91, for partially financing this work.

An Empirical Exploratory Study on Inferring Developers' Activities from Low-Level Data

Irina Coman, Alberto Sillitti
Free University of Bolzano/Bozen, Italy
{icoman,asillitti}@unibz.it

Abstract

Automated data collection techniques offer information on low-level activities while approaches such as PSP require information on higher-level activities. The mapping between low-level activities and higher-level ones is still needed. The purpose of this research is mainly exploratory as a first step towards the development of a technique of inferring (semi)automatically the higher-level activities from low-level data, independently of the particular tools used by the developers. We present evidence that such an approach is realistic: there are two types of application usage that relate to two higher-level activities of developers. Based on our findings, we shape a hypothesis for automatically detecting these types of applications usage.

1. Introduction

Software companies are interested in improving the software development process, as proved by their interest in process modeling toolkits, [7]. The Personal Software Process, (PSP), [4], has been introduced in 1995 as an approach for empirically guided software process at the level of individual developers. In order to provide the needed complete and reliable data, automated and non-invasive techniques of data collection have been proposed, [8], [10]. However, automatic tools monitor low-level activities (such as access of a file) while processes such as the PSP require the monitoring of higher-level activities (such as design or coding). Therefore, a mapping between low-level activities and higher-level ones is essential in order to make real use of automatically collected data. We try to address this problem by investigating the possibilities for automatic inference of high-level activities from low-level data.

Section 2 of this paper presents a description of related works. Section 3 and subsections present our approach in terms of novelty, goals, data, analysis and results. Finally, section 4 mentions the limitations of our study and presents the conclusions and the directions that we are considering for future work.

2. Related work

Researches that relate to ours are mainly in the field of automated process discovery, modeling and monitoring, [2], [3], [5], [15]. There are approaches to map low-level, manually predefined events, into transitions in the process model enactment, [2], or to infer a formal model of the behavior of the process from low-level event streams, assuming a finite state machine model, [3], [5]. More focused approaches aim at automatically assessing whether developers are indeed following a TDD (Test Driven Development) process, [15], analyzing data collected automatically from specific development tools, [8].

Other researches relating to ours are investigating the general time structure of the activities of developers, [1], [11], the developers' working habits in terms of preferred ways of action, favorite tools and practices, [6], [16], software development tools usage and cognitive aspects, [9], and the strategies of solving particular development tasks, [12], [13], [14]. The data in these cases is not collected automatically.

3. Our approach

The main purpose of this study is to explore whether information regarding the tasks performed can be inferred from low-level, fully automatically collected data, and to shape hypothesis regarding ways of automated inference of such information.

By contrast to other approaches, we focus on the software process of developers rather than on the software process of a company, [2], [3], [5]. We explore possibilities as they emerge from the data, rather than assuming a specific model of the behavioral patterns of the process (such as the finite state machine in [3]). We only make use of automatically collected data, without relying on predefined patterns of events, [2]. We take into account all of the active time spent by the developer at her computer, rather than just the commit information, [5], or just the interaction with a specific development environment (such as Eclipse in [15]).

Higher-level studies regarding developers' behavior, such as [14], have offered a description of maintenance tasks as an interleaving of several activities. Our main hypothesis is that these activities have different time structures that are reflected at the low-level of automatically collected data. The time structure could be used in order to automatically infer these activities as they take place. Therefore we investigate the time structure of the usage of applications that correspond to these activities and we shape a hypothesis for automatic detection of the activities.

Our long-term goal is to automatically infer the chain of activities and to infer the task themselves using their structure with respect to activities. This would be useful for instance in order to ascertain the real time spent on each task and to identify complex tasks that might call for special attention. Moreover, by looking at the files involved during the time spent on the task, it would also be possible to have an always up-to-date map of the project, which relates the tasks to the physical artifacts, making thus valuable knowledge easily accessible to new developers.

3.1. Data

Given that we want to explore the possibilities of "what is" rather than to test the effect of something, we consider that real-world data is a must for our purpose. Although real-world data is harder to get and companies are less available than students, we prefer a smaller sample of real-world data rather than a bigger sample of laboratory-student data. As other studies have remarked, "data on real users, even if the sample is small, is revealing", [1].

We collect data by means of the PROM tool, [10], in a fully automated, non-invasive way. We considered only data that has been collected in a general manner, regardless of the specific software application that the developer was using.

We collected data over a 52 working days period, corresponding to almost 3 months, from 3 developers working in a software company. Out of the 52 working days, the three developers worked for 52, 51 and, respectively 46 days. The developers worked only on one web-application project, developed in Java. They worked only on bug-fixing and new feature tasks. They worked in one-week iterations. The software tools mainly used were Eclipse, Internet Explorer, Mozilla Firefox, Explorer, Notepad, HomeSite5, TopStyle Lite 2.1, IBExpert, SQL Server Enterprise Manager, Putty, WinSCP, AcrobatReader, and Microsoft Office. The project evolved, from 29316 LOC at the beginning of our study, to 57174 LOC at the end of our study.

Table 1. The format of the data: an event's structure.

Event	
User	2
Date	09.09.06
Start time	10:45:20
End time	10:46:32
File	OnClick.java
Package	Pack1.subpack1
Path	C:\project1\
Application	Eclipse
Duration (s)	72

The data is in the form of events of uninterrupted working time (staying on focus of the corresponding window) with a file (fine grained level) or with a software application (coarse grained level). We use the generic term of *file* to denote also a location in the case of applications that work with locations rather than files (e.g. Explorer). The time granularity of our data is of 1 second. In what follows, we call the uninterrupted working time with a file a *file event* and the uninterrupted working time with a software application an *application event*. An example of an event is given in Table 1.

3.2. Analysis

The main hypothesis, on which the assumption that higher-level information can be automatically inferred from low-level data is based, is that the time structure of the various higher-level activities involved in solving maintenance tasks is different, and that this time structure is reflected in the low-level data. Therefore, we look for differences in the time structure of various higher-level activities that are occurring during maintenance tasks, as reflected in the low-level data.

There are three activities occurring during maintenance tasks: "searching for task-relevant information, understanding the relations between information and editing, duplicating or otherwise referencing the necessary code" ([14]). Because understanding is more a cognitive activity than a concrete one, we consider it together with the searching in an activity that we call exploration. We consider the broader activity of implementation as consisting of "editing, duplicating or otherwise referencing the necessary code" or any other text needed. Therefore, we focus on two activities: exploration and implementation.

By their nature, there are applications that correspond mainly to exploration (like Internet browsers), mainly to implementation (like document editors or viewers) and those that can be used in both ways (like integrated development environments). Therefore, we investigate the time structure of these applications' usage in an attempt to identify possible differences among exploration and implementation. For this, we take into consideration, from our data, three categories of the most used applications: Internet related (as exploratory), document related (as implementation oriented) and development environments (as both exploratory and implementation oriented). The Internet related applications are IE and Firefox. The document-oriented applications are Acrobat Reader (depicted as acrord32), Notepad, Excel and Word (depicted as winword). The development environments are Eclipse and Homesite5.

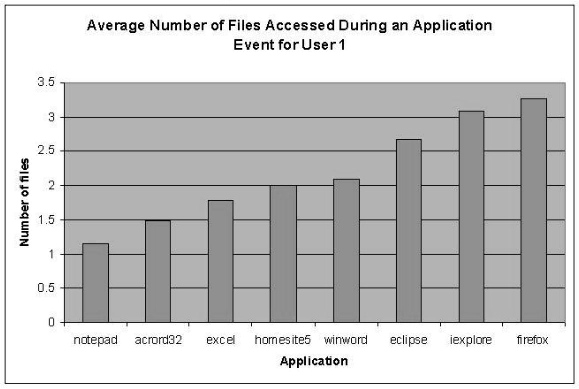


Figure 1. Average number of files during an application event for user 1.

As shown in Figure 1 and Figure 2, the Internet related applications exhibit frequent switches among several files during an application event (IE and Firefox), while the document oriented applications exhibit application events with only one file that is accessed for a longer period of time (Acrobat Reader, Excel). This can be explained by the fact that during exploration, the developers are searching for information and therefore spend relatively little time on a file, while browsing through several files. The document editors on the other hand are usually used for viewing or editing specific files rather than exploring through different files. In this case, the developers know already, possibly from previous explorations, what and where they want to view or change and they simply access the file or files needed.

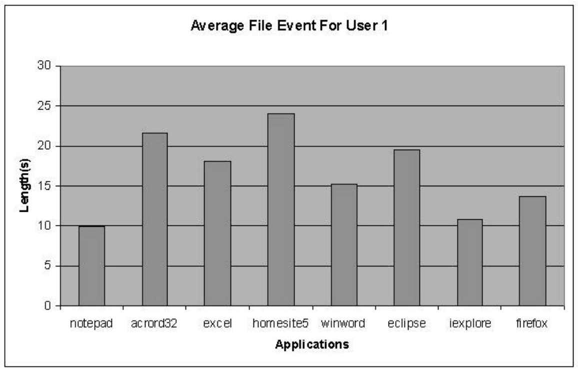


Figure 2. Average file event for user 1 and various applications.

There are also exceptions among the document related applications. Notepad exhibits the other document related applications' behavior regarding the number of files during an application event, but the time spent on a file is much smaller. This can be attributed to the fact that Notepad is a very simple text editor and therefore probably used only for simple viewing or editing tasks that don't take much time. Word, although a document application, is situated between the two classes of applications, rather than clearly belonging to one of them. This might be due to the fact that the developers were both consulting and editing requirements documents in Word.

The development environments are situated, both regarding the number of files and the average file event, between the Internet related and document oriented applications. This was expected given that they are intended both for exploration and editing. This represents further evidence that the average file event and the number of files are potentially good measures for identifying exploration and implementation.

The two main characteristics of the exploration are the usage of many distinct files, and the short time spent on each file. On the other hand, the main characteristics of the implementation are the usage of a small number of files and a relatively longer time spent on each file. Considering the fact that the implementation occurs naturally after the exploration and using the information gained through exploration, it seems as a reasonable supposition to consider that most of the files accessed during implementation, have been visited at least once during exploration.

Based on the previous observations, we propose, as hypothesis to be tested, that the two activities could be inferred from the evolution of the cumulative number of distinct files that are accessed over time by a developer, combined with the average time spent on one file. If our hypotheses are correct, the cumulative number of total distinct files should increase fast during exploration and remain constant or increase much slower during implementation. On the other hand, the average time spent per file should be low during exploration periods and should be bigger during implementations.

4. Conclusions and future work

The purpose of this study has been to investigate the possibility of automatic inference of high-level information from low-level, automatically collected data. The main hypothesis has been that the time structure of the various high-level activities involved in solving maintenance tasks is different, and that this time structure is reflected in the low-level data. We have found differences between the time structures of the usage of exploration and implementation related applications. Based on our results, we have shaped a hypothesis for automatic inference of exploration and implementation and we consider its proper testing as our next step.

A mapping is possible between the previously identified types of usage and the developers' activities during maintenance tasks that have been identified in [14]. Therefore, the automated inference of these types of usage could lead to the automated inference of higher-level tasks. This would be beneficial in order to properly map the low-level data to higher-level activities, which would, in turn, lead to benefits such as account of time spent per task, files that are concerned by various tasks and the possibility of real-time reconstruction of the software process as it actually happens.

Our results have to be considered keeping in mind the limitations of our study: the small size of our sample, the specificity of the project and development methodology investigated, the specificity of the tasks and the limitations of the tool which collected data only from the developers' computers, without being able to discern for short interruptions from the working environment.

5. References

[1] D. E. Perry, N. A. Staudenmayer, L. G. Votta, "People, Organizations and Process Improvement", *IEEE Software*, Vol. 11, No. 4, pp. 36 – 45, 1994

[2] N. S. Barghouti, B. Krishnamurthy, "Using Event Context and Matching Constraints to Monitor Software Processes", *Proc. 17th Intl. Conference on Software Engineering*, 1995

[3] J.E. Cook, A. L. Wolf, "Automatic Process Discovery through Event-Data Analysis", *Proc. 17th Intl. Conference on Software Engineering*, 1995

[4] W. S. Humphrey, "A Discipline for Software Engineering", *Addison-Wesley*, New York, 1995

[5] J.E. Cook, "Process Discovery and Validation through Event-Data Analysis", *Ph.D. Thesis*, Technical Report CU-CS-817-96, University of Colorado, September 1996

[6] J. Singer, T. Lethbridge, N. Vinson, N. Anquetil, "An Examination of Software Engineering Work Practices", *Proc. CASCON '97*, 209 – 223, 1997

[7] L. Benedicenti, G. Succi, T. Vernazza, "Improving Engineering", *ACM Applied Computing Review*, ACM Press

[8] P. M. Johnson et all, "Beyond the Personal Software Process : Metrics Collection and Analysis for the Differently Disciplined", http://csdl.ics.hawaii.edu/techreports/02-07/02-07.pdf, July 2002

[9] R. Naghshin, A. Seffah, R, Kleine, "Cognitive Walkthrough + Personae = An Empirical Infrastructure for Modeling Software Developers", *Proc. IEEE Symposium on Human Centric Computing Languages and Environments*, 2003.

[10] A. Sillitti, A. Janes, G. Succi, T. Vernazza, "Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data", *EUROMICRO 2003*, Turkey 2003

[11] V. M. Gonzalez, G. Mark, "Constant, Constant, Multi-tasking Craziness": Managing Multiple Working Spheres, *Proc. of the CHI*, 2004

[12] M. P. Robillard, W. Coelho, G. C. Murphy, "How Effective Developers Investigate Source Code: An Exploratory Study", *IEEE Transactions on Software Engineering*, Vol. 30, No. 12, December 2004

[13] C. Murphy, M. Kersten, M. P. Robillard, D. Cubranic, "The Emergent Structure of Development Tasks", *Proc. of ECOOP,* 2005

[14] A. J. Ko, B. A. Myers, M. J. Coblenz, H. H. Aung, "An Exploratory Study of How Developers Seek, Relate and Collect Relevant Information during Software Maintenance Tasks", *IEEE Transactions on Software Engineering*, December 2006

[15] H. Kou, P. Johnson, "Automated Recognition of Low-Level Process: A Pilot Validation Study of Zorro for Test-Driven Development", *Proc. of the Intl. Workshop on Software Process*, May 2006

[16] T. D. LaToza, G. Venolia, R. DeLine, "Maintaining Mental Models: A Study of Developer Work Habits", *ICSE 2006*, May 2006

TRAP.NET: A Realization of Transparent Shaping in .NET

S. Masoud Sadjadi and Fernando Trigoso
School of Computing and Information Sciences
Florida International University, Miami, FL, U.S.A
{sadjadi, ftrig001}@cs.fiu.edu

Abstract

We define adaptability as the capacity of software in adjusting its behavior in response to changing conditions. To list just a few examples, adaptability is important in pervasive computing, where software in mobile devices need to adapt to dynamic changes in wireless networks; autonomic computing, where software in critical systems are required to be self-manageable; and grid computing, where software for long running scientific applications need to be resilient to hardware crashes and network outages. In this paper, we provide a realization of the transparent shaping programming model, called TRAP.NET, which enables transparent adaptation in existing .NET applications as a response to the changes in the application requirements and/or to the changes in their execution environment. Using TRAP.NET, we can adapt an application dynamically, at run time, or statically, at load time, without the need to manually modify the application original functionality-hence transparent.

1. INTRODUCTION

The goal of our ongoing research is to improve software adaptability. Imagine a world where software systems do not have to stop every time there was a need for adapting the software to the new changes in its requirements or in its execution environment. For example, as a wireless user moves from one wireless cell to another, the applications available to the user must adapt to different environments and resources with minimal interruption in their service. Critical systems such as financial networks or power systems cannot afford to shut down due to the need to adapt to new conditions or security attacks. A hurricane prediction application running for hours cannot be restarted just because a few resources are out of service.

Adaptable software presents itself as a possible solution to the above problems. An application is said to be *adaptable* if it can change its behavior dynamically at runtime, which may be due to changes in its environment or due to new functional or non-functional requirements [1]. Unfortunately, developing adaptable software is nontrivial. An adaptable application involves both *functional code* and *adaptive code*. Functional code implements the business logic of the application while adaptive code implements the adaptation logic which enables the application to be adaptive. Usually, these two types of code are blended into one source code making its maintenance and adaptation difficult.

The *transparent shaping* programming model poses a solution to solve the difficulty of developing adaptable applications [1]. This model allows the design and development of adaptable applications without the need to modify their source code.

Transparent shaping produces adaptable programs in two steps. In the first step, usually executed at compile time, an existing program is transformed so its behavior can be adapted at runtime. And in the second step, executed at runtime, these transformations receive the new behavior so the program can be adapted.

The first step of transparent shaping generates an adapt-ready program. An *adapt-ready* program is a program whose behavior is initially equivalent to the original program except for the fact that it can be adapted at startup and/or run time.

Applying transparent shaping to object-oriented programs yields a new programming model called *Transparent Reflective Aspect Programming* (TRAP) [1]. In this paper, we provide a realization of transparent shaping following the TRAP model, called TRAP.NET, which is targeted for .NET applications.

TRAP.NET provides a language-independent mechanism for transparently producing adapt-ready programs from existing programs in .NET. TRAP.NET also provides a mechanism to adapt these adapt-ready programs. This adaptation can be static, at load time, or dynamic, at runtime.

Static adaptation is more restrictive than dynamic adaptation. Static adaptation can only occur once when the application is loading and it is useful for applications that will be deployed on different platforms. These applications only need to adapt to their corresponding platforms at startup time. Dynamic adaptation is useful for applications that need to adapt to changes in their environment or to new requirements without halting.

The major contributions of this paper are summarized as follows. First, we assess the expressiveness and effectiveness of .NET Attributes in providing a means to label what portions of an existing application should become adaptable. *Attributes* are pieces of metadata information that can be placed in the source code of .NET applications. TRAP.NET uses this metadata information to identify which pieces of functionality should be made adapt-ready. Second, we researched and developed a language-independent software tool that realizes the first step of transparent shaping appropriate for .NET applications. We call this software tool the *generator*. The generator automatically generates an adapt-ready application independent of the programming language used in the development of the original application. Finally, we researched and developed a language- and platform-independent software tool that realizes the second step of transparent shaping. We call this software tool the *composer*. The composer allows new adaptive behavior—to be added at startup or runtime—to replace the existing adapt-ready behavior.

The remainder of this paper is organized as follows. Section 2 provides background on the .NET technologies that TRAP.NET uses. Section 3 explains the design, implementation, and operation of TRAP.NET. Section 4 compares TRAP.NET to some related work. Finally, Section 5 offers some concluding remarks.

2. BACKGROUND

This section provides a brief overview of the .NET technologies that TRAP.NET uses to implement the transparent shaping model.

The Microsoft .NET Framework is a software component that provides vast pre-coded solutions to common program requirements in the form of class libraries. It also manages the execution of programs written specifically for this framework [6]. The .NET Framework provides a run-time environment called the co*mmon language runtime* (CLR, or just runtime) that runs .NET applications and provides services that make the development process easier. When compiling a .NET program, the compiler translates the source code into *common intermediate language* (CIL, or just IL), which is a CPU-independent set of instructions that can be efficiently converted to native code. CIL, formerly called *Microsoft intermediate language* or MSIL, resembles an object-oriented assembly language. TRAP.NET is language-independent as it provides adaptation at the CIL level.

When a .NET compiler produces code in CIL, it also produces the corresponding metadata. *Metadata* is "data about data;" in the programming context it is data about the program. Metadata describes the types in the code, including the definition of each type, the signatures of each type's members, the members that the code references, and other data that the runtime uses at execution time. The CIL and metadata are contained in a portable executable (PE) file referred to as *module*. The presence of metadata in the module along with CIL enables the code to describe itself. The runtime locates and extracts the metadata from the module as needed during execution.

.NET Reflection is the component that provides class libraries to access the metadata of .NET applications. Therefore it contains classes to describe every programming element such as assemblies, modules, namespaces, types, fields, methods, attributes, events, etc. Using reflection, TRAP.NET can access the metadata and the IL code of existing applications. TRAP.NET takes advantage of this feature and puts all the adaptive-related metadata into the same file and this way, there is no need to load a separate file with the metadata information about the adaptive behavior.

Reflection can also be used to dynamically create an instance of a type, bind the type to an existing object, or get the type from an existing object. Then it can invoke the type's methods or access its fields and properties. TRAP.NET can discover information about all the elements of an application. Also, it can modify an application's behavior mainly through the usage of dynamic methods. *Dynamic methods* are methods that can be generated and executed at runtime.

.NET Attributes are keyword-like descriptive declarations. They resemble programming languages reserved keywords such as *public* or *private*. These two keywords further define the behavior of class members by describing their accessibility to other classes. Because compilers are designed to recognize these predefined keywords, a developer does not traditionally have the opportunity to create their own. The CLR, however, allows the addition of attributes to annotate programming elements such as types, fields, methods and properties [7]. These attributes can be extracted using runtime reflection services. TRAP.NET uses attributes to label methods that should become adapt-ready and to gather metadata information about these methods.

3. TRAP.NET OVERVIEW

This section provides an overview of TRAP.NET from its usage perspective. It describes the steps required to achieve dynamic adaptation with TRAP.NET at development time, compile time and run time. Static adaptation is presented at the end of this section since it reuses the techniques used to achieve dynamic adaptation.

3.1. At Development Time

To tailor the TRAP approach to the .NET development practices we allow the user to manually place .NET Attributes to annotate which methods should be adapt-ready. In our particular case we implemented a custom attribute which we call the *AdaptReady* attribute. By placing this attribute, the user is staging the application so it is suitable for the generator at compilation time. The generator can then look for the methods with the *Adap-*

tReady attribute to automatically make them adapt-ready. The code snippet in Figure 1 shows this attribute with a method called *Some-Method*.

Another important step that occurs at development time is the fact that the user has to add a reference to the TRAP.NET class library. The *AdaptReady* attribute is defined in the TRAP.NET class library, thus to successfully compile the application with this attribute, the application has to reference this class library. This particular reference coalesces the original application and TRAP.NET into one application. This union makes the generation process much easier since most of the functionality to support adaptation can be placed in the TRAP.NET library. Therefore, most of the adaptive code that needs to be weaved can be simple calls to functions in the TRAP.NET library.

```
[AdaptReady(true)]
public void Some-Method()
{
    /* some implementation */
}
```

Figure 1. A method with the *AdaptReady* attribute.

3.2. At Compile Time

After the annotated application is compiled as it normally would, the user can send its application to the generator. The generator can be executed from a plug-in for Visual Studio or form the command-line.

The generator is in essence an aspect weaver which adds the adaptive aspect to the methods annotated with the *AdaptReady* attribute. The adaptive aspect in this case consists of hooks that will intercept and redirect the control flow as appropriate. As mentioned earlier, since all .NET assemblies are compiled into CIL, the generator can weave the adaptive aspect in CIL code. The addition of the adaptive aspect at this level makes the TRAP.NET generator language independent and transparent with respect to the original source code.

When the generator is executed, it receives as input the annotated assembly. This assembly is immediately loaded into an *Assembly* object provided by the .NET reflection facilities. This object allows the discovery of the types and methods of the staged assembly. The generator iterates through the list of types and methods and finds all the methods with the *AdaptReady* attribute.

At this point, as illustrated in Figure 2, first the annotated assembly is disassembled using ILDASM, which is a tool distributed with the .NET Framework, to create a text file with the intermediate language (IL) code of the assembly. Next, the source code of the method, in IL, is used as a base for the generation of the adapt-ready IL code. Finally, the adapt-ready IL code is reassembled using ILASM, which generates the adapt-ready assembly. The process described in this paragraph is called round-tripping, which involves three steps: disassembly, tinkering with the CIL source code, and reassembly [8].

During the tinkering phase, the generator only adds the hooks to the methods that have the *AdaptReady* attribute. The actual hooks consist of a simple *if-then-else* statement to intercept and redirect the control flow. The *if*-condition intercepts the control flow and checks whether adaptation is enabled for that particular method. If it is, the automatically generated code inside the *if*-condition redirects the control flow. It loads and invokes the new adaptive functionality for this method using a dynamic method. The *else*-condition wraps the original functionality of the method which is executed if adaptation is not enabled.

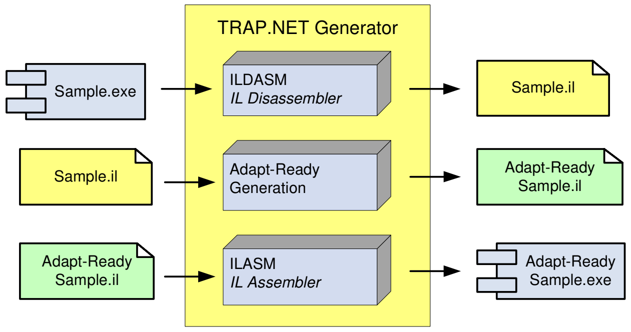


Figure 2. TRAP.NET Round-tripping.

Presenting the code of an adapt-ready method in CIL can be cumbersome due to the fact that intermediate language notation is very similar to assembly language which tends to be lengthy even for simplistic logic. Moreover, understanding CIL requires knowledge of CIL instructions and syntax. Therefore, for clarity, in Figure 3 we show what an adapt-ready method would look like in pseudocode before and after generation.

```
[AdaptReady(true)]
SOME-METHOD()
1   call original implementation
```

(a) *Some-Method* before generation.

```
[AdaptReady(true)]
SOME-METHOD()
1   if this method is adapted
2       then call INVOKE-DYNAMIC-METHOD()
3       else call original implementation
```

(b) *Some-Method* after generation.

Figure 3. Adapt-ready method in pseudo code before and after generation.

Inside the *if*-condition, the generator has to perform code generation to invoke the dynamic method. The dynamic method has to be invoked with all the parameters of the original method which may be one or more and may be of different types. Also, the return type of the invocation needs to be casted to the return type of the original method. Using reflection the generator can determine the number and the types of parameters as well as the return type of the method. Based on this information, the code to invoke the dynamic method is automatically generated.

Finally, besides adding the hooks, the generator also inserts a method call in the startup point of the application. When the adapt-ready application starts running, this method call is the first thing that gets executed. This method is part of the TRAP.NET class library and initializes all the components and data structures needed to support static and dynamic adaptation at runtime. As part of this initialization a communication channel is opened so the application can receive new functionality at runtime. This communication is implemented using *.NET Remoting*, which supports interprocess communication in distributed applications.

3.3. At Runtime

The last few steps to achieve dynamic adaptation with TRAP.NET occur at runtime when the adapt-ready application is executing. They are triggered when the user decides that the functionality of a particular adapt-ready method needs to adapt to some changing condition. Future work may involve automated decision making according to some policy. After this decision is made, the user develops the new functionality using the original source code. Then, the user can utilize the composer to upload the new adaptive functionality.

The composer is essentially a distributed application composed by two modules: the *client composer* and the *server composer*. The client composer is the user interface used to upload new functionality to the running application. The server composer—hosted by the TRAP.NET class library—is part of the running adapt-ready application and it serves requests from the client composer. The composer is the core of TRAP.NET because it achieves dynamic adaptation. Figure 5 shows the dependency of these components at runtime.

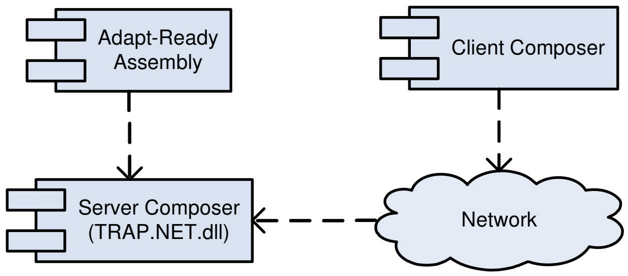


Figure 5. Dependency of components at runtime.

We developed two client composers, a Windows application composer and a Web application composer. Among all of their functionality, these composers perform three basic operations. The first one is to get the status of the adapt-ready application so the user can see which methods can be adapted. The second operation lets the user upload a new *delegate assembly* to the adapt-ready application. A delegate assembly is an assembly that contains *delegate methods*. Delegate methods contain the functionality that replaces the functionality of adapt-ready methods. Once the delegate assembly is loaded, the third operation lets the user adapt an adapt-ready method with a delegate method.

After the second operation, when user loads the delegate assembly, the server composer matches adapt-ready methods to their potential delegate methods candidates. Potential delegate methods are methods that have the same return type and parameter types as the adapt-ready method. Once this matching is complete, the results are returned in XML format to the client. The user interface then displays the adapt-ready methods with their potential delegate methods. At this time, the user may select one of the delegate methods for adaptation of its adapt-ready method. This adaptation request is sent to the server which prepares the contents of this delegate method so they are suitable for adaptation.

The preparation of the delegate method consists of the generation of a dynamic method with the contents and information of the delegate method. This generation is a complex process that relies heavily on reflection. The shell of the dynamic method has to have the same metadata as the delegate method. This metadata includes the parameter types, the declaring type and the local variables of the delegate method. After the shell of the dynamic method is completed, its contents are populated with the body of the delegate method. This body is the IL of the delegate method in byte code. Assuming the delegate method has no external references—that is, it only works with local variables and it does not call other methods—the newly created dynamic method is ready for usage. This dynamic method is stored in a data structure inside the server composer so it can be referenced when needed. The next time the adapt-ready method is called, it finds out that is has been adapted and its execution enters the *if*-condition which gets the dynamic method from the server composer. After the dynamic method is retrieved, the adapt-ready method invokes it with its current parameter values.

The process just described assumes that the delegate method had no external references to members outside the method itself. Non-trivial applications usually require access to external references. *Members* are any language element that can be referenced, i.e. methods, constructors, fields, properties and events. Each of these members may also have return types, parameters, access modifiers (such as *public* or *private*) and implementation details (such as *abstract* or *virtual*) among others. When a delegate method is developed and compiled it may reference members in the delegate assembly itself. After the delegate method is ported into a dynamic method in the running application, these references are still pointing to the delegate assembly. However, the delegate assembly is not being executed and it is out of the context of the running application thus, these references are really pointing to nothing. Therefore invocation of a dynamic method with references to the delegate assembly would fail. The server composer solves this issue by redirecting these references to the running application.

To redirect references into the running application, the composer takes the following steps. First, it *finds* the references in the body of the delegate methods. After they are found, it *resolves* them, that is, it gets the actual delegate member being referenced so it can discover its signature. Using the signature of the delegate member, it *locates* the member with the same signature in the running application. Then, it *replaces* the reference in the delegate body so it points to the member in the running application.

The functionality that finds references in the body of delegate methods had to be developed from scratch. Reflection only provides access to the metadata of methods, not their actual contents. As a consequence, we developed a component, called the *token parser*, which finds external references in methods. Every member has a unique metadata identifier. These metadata identifiers—also referred to as metadata tokens—are used to uniquely reference members. The token parser extracts these references by parsing the byte code of delegate methods.

Using reflection and the tokens discovered by the token parser, the composer can resolve the members being referenced. After these members are resolved, it locates them in the running application. Once the composer locates these members, it can obtain their metadata identifiers or tokens. At this point the composer has the tokens that represent references in both assemblies. The composer then proceeds to replace tokens in the delegate body.

After replacement is completed, the new body is assigned to the dynamic method. The references in the dynamic method body have now been redirected to members in the running application. Now, the dynamic method is ready for invocation.

So far we have presented how dynamic adaptation is achieved at runtime. TRAP.NET can be used in a mode that enables static adaptation. This type of adaptation occurs at load time by reusing the functionality that achieves dynamic adaptation. To achieve static adaptation the generator adds code to the startup of the application to pause it as soon as it starts executing. While the application is paused, the composer is the only available component which is waiting to receive delegate methods. The user then sends delegate methods and adapts the desired adapt-ready methods. The composer then flags these methods as adapted and will not allow new adaptive functionality during the execution of the program. When the user wishes, the application will be resumed.

4. RELATED WORK

Similar to TRAP.NET, other approaches to build adaptable programs involve *intercepting* calls to functional code, and *redirecting* them to adaptive code [1]. There are two main categories of related work. The first category involves approaches that *extend middleware* to support adaptive behavior (*e.g.,* Iguana/J [3] that extends JVM). Since the role of traditional middleware is to hide resource distribution and platform heterogeneity from the business logic of applications; it is a natural place to put adaptive behavior. However, these approaches generally become outdated after a newer version of the middleware (*e.g.,* JVM) is released.

The second category includes approaches to transparently *augment the application code* with facilities for interception and redirection. Examples related to our work include AspectJ [4], Aspect.NET [5] and TRAP/J [2]. AspectJ and Aspect.NET enable aspect weaving at compile time which is similar to the task the TRAP.NET generator performs. However, they do not provide a means for dynamically weaving new code into the application at runtime.

TRAP/J is an instance of TRAP in Java. To augment an existing Java program with the required hooks, TRAP/J uses compile-time aspect weaving provided by AspectJ. Following the TRAP approach, TRAP/J operates in two phases. In the first one, at compile time, TRAP/J converts an existing program into an adapt-ready program. This conversion is accomplished using an Aspect Generator and a Reflective Class Generator. These generators produce aspects and the reflective classes. Next, these two products, with the original source code, are passed to the AspectJ compiler which weaves the generated and the original source code together to generate and adapt-ready assembly. Note that the generation process in TRAP/J generates significantly more code than the generation process in TRAP.NET. The second phase occurs at runtime when using the reflective classes, new behavior can be introduced to the application.

A limitation of TRAP/J is the fact that it can only make classes adapt-ready. Even if the user only wishes to make one method in a class adapt-ready, TRAP/J will make the entire class adapt-ready. Performance and flexibility seem affected by this limitation. In contrast, TRAP.NET overcomes this limitation since it is able to make methods adapt-ready. Moreover, methods are the units of functionality and behavior in the object oriented paradigm. Since transparent shaping focuses on changing the business logic which is hosted by methods, TRAP.NET offers a more natural implementation. The state of the object is not changed by adaptation itself. Adapting a method only changes its functionality. The new adaptive functionality may change the state of the object as it was programmed by the user.

Other contributions of TRAP.NET include its language independence, tailoring of the transparent shaping model to .NET development practices and extensive member access. TRAP.NET can make any application adapt-ready, independently of the language that it was developed since it works at the intermediate language level. Also, by using attributes it customizes the transparent shaping model so it fits .NET development practices. Moreover, TRAP.NET is the implementation that offers the most types of member access to the adaptive functionality. The new adaptive functionality may need to access resources or members in the running application. TRAP.NET enables this accessibility.

5. CONCLUSIONS

The next major step for TRAP.NET should be the development of a case study with an application geared towards pervasive, autonomic or grid computing. This case study will produce important results to improve and evaluate TRAP.NET in many aspects including performance, reliability and usability.

Safe adaptation and security are two main concerns that remain pending in this research. Safe adaptation is the ability of a program to maintain its integrity during adaptation [9]. And, security deals with protecting an adaptable program from malicious entries [1]. These two issues are ongoing research areas for dynamic adaptation. Future work in TRAP.NET should support safe adaptation and security as these techniques become available.

The major achievement of this paper is the research on the design and development of TRAP.NET. This tool is a successful realization of transparent shaping using the TRAP model that provides dynamic adaptation of applications without the need to modify their original functionality. TRAP.NET provides the means to achieve separation of concerns when developing adaptive applications. It adheres to the aspect-oriented paradigm by adding adaptive functionality across the business logic of an application. It uses reflection to discover and modify the functionality of an application. Moreover, unlike similar tools, it intercepts only the methods selected by the user. The rest of the application remains intact maintaining its original performance. Adaptive code in TRAP.NET is achieved by developing delegates which can provide its own new functionality and reuse the already existing functionality of the adapt-ready application. This important achievement contributes to the developing area of software adaptation in order to support critical and long-time running applications such as the ones found in pervasive, autonomic and grid computing.

ACKNOWLEDGMENTS

This work was supported in part by IBM, the National Science Foundation (grants OCI-0636031, REU-0552555, and HRD-0317692). The authors are very thankful to the following students who played a significant role in the implementation of TRAP.NET: Allen Lee, Tuan Cameron, Ana Rodriguez, Juan H. Cifuentes, Javier Ocasio, Amit Patel, Mitul Patel, Enrique E. Villa, Frank Suero, Etnan Gonzalez, Edwin Garcia, Alain Rodriguez, and Lazaro Millo.

TRAP.NET is a follow-up work on the Transparent Shaping research originated in Michigan State University and the authors are thankful to Philip McKinley, Betty Chen, and Kurt Stirewalt who contributed to the original ideas in Transparent Shaping, its TRAP extension, and the realization of TRAP in Java, called TRAP/J. Our gratitude is also extended to our colleagues at Florida International University, Peter Clarke and Masoud Milani, who provided us with feedback on this work.

REFERENCES

[1] S. M. Sadjadi. Transparent Shaping of Existing Software to Support Pervasive and Autonomic Computing. *A Dissertation submitted to Michigan State University*, 2004.

[2] S. Masoud Sadjadi, Philip K. McKinley, Betty H.C. Cheng, and R.E. Kurt Stirewalt. TRAP/J: Transparent generation of adaptable Java programs. In *Proceedings of the International Symposium on Distributed Objects and Applications (DOA'04)*, Agia Napa, Cyprus, October 2004.

[3] Redmond, B., Cahill, V.: Supporting unanticipated dynamic adaptation of application behavior. *In Proceedings of ECOOP*, 2002.

[4] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Videira Lopes, J. M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP).* Springer-Verlag LNCS 1241, June 1997.

[5] Vladimir O. Safonov. Aspect.NET – an aspect-oriented programming tool for Microsoft.NET. In *Microsoft Research SSCLI RFP II Capstone Workshop 2005*, September 2005.

[6] .NET Framework. *Wikipedia*. 2 March 2007. Available at URL: http://en.wikipedia.org/wiki/ .NET_ framework.

[7] Attributes Overview. *MSDN Library for Visual Studio 2005.* 2005. Available at URL: http:// msdn.microsoft.com/library/en-us/cpguide/html/ cpconattributesovervi-ew.asp.

[8] Serge Lidin. Expert .NET 2.0 IL Assembler. Apress. 2006, pages 389-408.

[9] Ji Zhang, Zhenxiao Yang, Betty H.C. Cheng, and Philip K. McKinley. Adding safeness to dynamic adaptation techniques. In *Proceedings of the ICSE 2004 Workshop on Architecting Dependable Systems*, Edinburgh, Scotland, May 2004.

A Framework for Selecting Agile Practices and Defining Agile Software Processes

Patrícia Vilain
INE, UFSC, Brazil
vilain@inf.ufsc.br

Priscila Basto Fagundes
CTAI - SENAI/SC, Brazil
priscilab@ctai.senai.br

Thiago Leão Machado
INE, UFSC, Brazil
thiagolm@inf.ufsc.br

Abstract

The main goal behind agile methods is to reduce the time spent on software development. Each of those methods tries to promote rapid development with no loss of efficiency while keeping client and user satisfaction by applying its own set of agile principles and practices. Deciding which method to adopt for a given reality is a complicated matter, and sometimes it may be necessary to combine practices from different methods to better approach the problem. This paper presents a framework containing the agile practices adopted by several agile methods and interdependencies between such practices, as a basis for comparing and selecting which practices should be part of the software process. A tool for defining new software processes based on this framework is also presented. Besides helping managing the selected practices and respective interdependencies, this tool also allows generating and organizing the artifacts obtained from selected practices.

1. Introduction

Several methods proposing to accelerate the software development process have come up in the past years. These methods are referred to as agile methods and usually follow the same principles, such as: short iteration cycles, functionality driven planning, constant feedback, tolerance to changes, team proximity, client intimacy, and focus on the work environment [4]. A variety of different development practices is suggested by those methods in order to fulfill these agile principles. Therefore, deciding which agile method to adopt for a given reality may become complicated and time-consuming for organizations, and sometimes it may even be necessary to combine practices from different methods to better approach the problem.

This paper proposes a framework to support organizations in selecting or customizing their own agile software processes. The purpose of proposing a framework for agile methods is to provide a structure that contains, in an organized and consistent way, the agile practices and activities suggested by commonly used agile methods, thus allowing organizations to efficiently define an agile software process according to stakeholder needs.

The proposed framework includes agile practices adopted by several agile methods, namely: Extreme Programming (XP) [3], Scrum [5] [17], Feature Driven Development (FDD) [14] [6], Adaptive Software Development (ASD) [11], Dynamic System Development Method (DSDM) [9], Crystal Clear [8] [11], Lean Software Development (LSD) [15], and Agile Modeling (AM) [2]. It also identifies interdependencies between agile practices, thus assuring practices are applied in a consistent order within the development process being defined.

A tool for defining new software processes based on this framework is also presented. This tool not only helps managing the selected practices and respective interdependencies, but it also allows the generation and organization of the artifacts obtained from the selected practices.

The rest of this paper is organized as follows. Section 2 describes the proposed framework. Section 3 presents the tool we created to support the framework. Section 4 describes two case studies using the proposed framework. Finally, in section 5, we present our conclusions

2. The Proposed Framework

All the agile methods selected as the basis for the proposed framework present an incremental and iterative process and are indicated to small and medium sized teams that develop software with dynamic and constant changeable requirements. These methods were selected not only for being widely recognized as agile methods, but also because the available literature was considered sufficient to provide a good understanding of how such methods should be applied to software development. Although there are other methods like the Unified Process that can be applied in an agile way, those were not included here because, as pointed out by [11], the normal usage of such methods tends to be rigorous.

The proposed framework consists mainly of a set of agile practices that can be applied to software development. In order to specify this framework, we first had to determine which practices should be part of it by analyzing the agile methods selected above and comparing activities, practices and roles in each of those methods. We borrowed the list of activities from the incremental development process [16] as the basis for this comparative analysis of agile methods. Each of the practices included in the framework was then allocated to one of the activities of the incremental development process. Finally, the interdependencies among practices had to be identified to constrain their application within the development cycle, thus assuring practices are applied in a consistent order. The details of this process and the framework that resulted from it are described next.

2.1. Comparing Agile Methods

The set of activities proposed for the incremental development process [16] constitutes the basis for our comparative analysis of agile methods. According to [12], incremental development principles are being implicitly used since the 50's by different development approaches. These principles have gained more attention, especially since the 90's, when the agile approach emerged.

The strategy adopted for comparing agile methods was mainly to organize agile method practices and roles according to incremental process activities. For each activity in the incremental development process, we created a table that lists each agile method along with its corresponding activity and respective practices and roles, as in the table shown below for the activity Define Outline Requirements (Table 1). These tables help making the differences and similarities of agile methods more evident, and proved to be a very useful resource for the comparative analysis of the agile methods in question. The complete set of tables that resulted from this analysis can be found in [10].

Table 1. Activity define outline requirements.

Activity Define Outline Requirements			
Method	Specific. of the Activity	Practices	Roles
XP	Customers write user stories.	- Planning Game - Metaphor - On-site customer	- Customer - Manager (Big Boss) - Programmers
Scrum	Definition of the Product Backlog	- Product Backlog	- Scrum Master - Product Owner - Customer
FDD	Generation of engines for the documentation of requirements (use cases, user stories, UML class and sequence diagrams)	- Developing by Feature	- Manager Project - Chief Architect - Domain Expert - Domain Manager
ASD	Requirements defined during JAD sessions	-	- Executive Sponsor - Facilitator - Scribe
DSDM	System specifying including user classes and mandatory requirements	- Workshop - Prototyping	- Visionair - Techn. Coordin. - Executive Sponsor - Ambassador - Project Manager - Advisor User
Crystal Clear	Elaborate an actor-objective list	- Workshop	- Business Specialist - Ambassador User - Responsible Executive
LSD	Requirements gathering during the project	- Requirements - Disaggregating - Scope	
AM	Generation of a Document of Requirements; Executive General Vision; General Vision of the Project	-	-

2.2. Selecting Agile Practices

As a result of the comparative analysis described above, similar practices from different methods were identified and put together, thus helping eliminate redundant practices. Each activity of the framework along with the related practices is mentioned next. A complete description of these practices, the roles incorporated, and the set of methods that originally define each practice is found at [10] and [13].

The activity *Define Outline Requirements* includes the following practices: User Stories, List of Requirements, Detailed Documentation of Requirements, and Document Organization.

The activity *Assign Requirements to Increments* includes the following practices: Iteration Planning, Duration of Iterations, Distribution of Requirements to Responsible, and Documentation Update.

The activity *Design System Architecture* includes the following practices: Overall System Design, System Detailed Design, and Documentation of the Generated Design.

The activity *Develop System Increment* includes the following practices: Requirements Development During the Iterations, Writing of the Unit and Acceptance Tests, Collective Development of Code, Pair Programming, Refactoring, Daily Meetings, Simultaneous Development, Integration Parallel to Development, Version Control, Side-by-side Programming, and Documentation of Development.

The activity *Validate Increment* includes the following practices: Integration of the Increment before Validation, Execution of Unit and Acceptance Tests, and Code Inspection.

The activity *Integrate Increment* includes the practice Integration of the Resulting Increment.

The activity *Validate System* includes the practices Iteration Review Meeting and System Deployment into Production.

The activity *Final System* includes the following practices: Delivery of the System to Customer, Generation of a Brief Documentation, and Refinement of the Generated Documentation.

2.3. Interdependencies among Practices

The practices that are part of the framework can be chosen and executed according to development team needs, following the order suggested by the activities from incremental development. However, some practices are dependent on each other, that is, they can only be executed after or together with other practices, and some practices can only be executed when others are not. These interdependencies among practices are specified by an *activity matrix*. Due to space limitation, only a small part of the activity matrix is presented in table 2. The complete activity matrix containing all the activities and practices proposed in the framework is available at http://www.inf.ufsc.br/~vilain/agilemethods/ActivityMatrix.pdf.

Table 2. Partial activity matrix.

		Define outline requirements				Assign req. to increments				Design system architect.		
		User Stories	List of Req.	Detailed Doc. of Req.	Doc. Organiz.	Iteration Planning	Dur.of Iterations	Distrib. of Req.	Doc. Update	Overall System Des.	System Det.Design	Doc.of the Design
Des. system architect.	Overall System Des.	**D***	**D***	**D***		**D**						
	System Det. Design *	**D***	**D***	**D***		**D**				**D**		
	Doc.of the Design *									**D***	**D***	
Develop system increment	Developm.	**D***	**D***	**D***		**D**				**D**		
	Writ .Unit and Accep. Tests *	**D***	**D***	**D***						**D**		
	Collective Develop. *							**E**				
	Pair Progr *											
	Side-by-side Progr. *											
	Refactoring *											

Optional practices, that is, those that may not be executed, are marked with a * beside its name (horizontal entries). The dependence among practices is indicated by the letter "D" indicating the horizontal practice depends on the execution of the vertical practice. If a practice depends on the execution of at least one from a set of practices, we use the letter "D" followed by a *. The excluding practices are marked with letter "E", that is, the horizontal practice eliminates the possibility of executing the vertical practice. In this case, the development team should choose one between the two excluding practices. It is important to pay attention to the dependences between practices, because even though a practice is marked as optional, it may become mandatory if other practices that depend on it are executed.

3. Framework Supporting Tool

We created a tool that gives support to the definition of agile processes according to the proposed framework. This tool provides an easy-to-use graphical interface that allows developers to easily select which practices will be utilized in each of the activities of the development process, while automatically checking for the interdependencies among select practices as specified in the activity matrix described above. In other words, this tool can assist developers in defining new or customized agile software processes based on the activities and practices of the proposed framework. The tool is available at www.inf.ufsc.br/~vilain/agilemethods/fcama.zip and its detailed documentation can be found in [13].

The tool main screen is divided in three areas: process area, descriptive area, and work area. The process area presents the development process organized as a hierarchical tree, where practices are grouped in activities and activities are grouped in iterations. When a user selects an activity, iteration or practice, the editor screen for the selected artifact is shown in the work area, while the descriptive area presents a description of the practice currently active in the work area.

4. Case Studies

The proposed framework was utilized in two case studies. The first case study was on a research project developed by the development team of CIRAM/Epagri, the meteorological information center of the state of Santa Catarina, south of Brazil. The goal of such project was the development of a website to provide support to the exchange of information between companies participating in the project, as well as to provide information about the atmospheric discharges in order to monitor the intensity and location of lightnings. The technical team responsible for the development of this system had not used any specific development method or prescriptive process before. As team members were looking for an iterative and incremental method and that would generate minimal documentation, they agreed to utilize the framework proposed. After development was complete, all team members that participated in this first case study pointed out their appreciation to the framework, especially because it promoted efficiency while having no impact in the team development routine.

The second case study was on the development of the tool described in section 4. Since we were looking for an agile method, we decided to use the framework and verify its efficiency. However, the development process had a particular feature in this case: it was design and implemented by one single person. So, some of the practices proposed by the framework could not be observed. On the other hand, it was interesting to verify that the framework can also be applied to one-person-only development processes.

5. Conclusion

This paper presented a framework to support organizations in selecting or customizing agile software processes. The proposed framework is composed by practices from a variety of agile methods. It allows development teams to select the practices according to their needs, thus defining new software processes that more easily fulfill their expectations.

In the definition of a customized agile process, the selection of the practices available in the framework must be carefully made. As the framework comprises the AM practices, a process including several of these practices could produce a lot of documentation, and consequently one of the most important agile principles would be violated.

The framework proposed was successfully applied in two case studies. Team participants stated that the simplicity of the framework allowed its use without significant impacts in the development routine, while helping them to complete the development of the system within deadlines and budgets.

We also presented an automated tool to support the practices proposed by the framework in order to facilitate its adoption during a development process. Such tool supports the definition of a development process as well as the organization and generation of the necessary artifacts to the process defined. Unfortunately, we could not use the tool in our case studies because it was not available yet.

Although we did not find anything similar to the framework proposed here, some of the previously existing research work not only provided us with a theoretical background on agile methods but also helped us on selecting the guidelines used to compare the methods that compose the framework. [1] presents a comparison among the agile methods Crystal Family, DSDM, ISD and PP. [7] presents a comparison of the agile methods XP, Scrum, FDD, Crystal Family, and DSDM, in relation to agile principles.

As a continuation of work presented here, we intend to apply the framework in additional case studies using the tool we developed for this purpose. We also intend to analyze others agile methods in order to look for possibly useful agile practices that could be included into the proposed framework.

6. References

[1] P. Abrahamsson, J. Warsta, M. Siponen, J. Ronkainen, “New Directions on Agile Methods: A Comparative Analysis”. *IEEE Computer Science*, 2003.

[2] S. Ambler. *Agile Modeling: effective practices for Extreme Programming and the Unified Process*. Bookman, Porto Alegre, 2004. (in portuguese)

[3] K. Beck, *Extreme Programming Explained. Embrace change.* Addison-Wesley, Reading, Massachusetts, 2000.

[4] K. Beck, A. Cockburn, R. Jeffries, J. Highsmith, “Agile Manifesto”. http://www.agilemanifesto.org, 2001.

[5] M. Beedle, M. Devos, Y. Sharon, K. Schwaber, J. Sutherland, “SCRUM: An extension pattern language for hyperprodutive software development”. *Pattern Languages of Programs'98 Conference*, 1998.

[6] P. Coad,, *Java Modeling in Color with UML.* Prentice Hall, 1999.

[7] M. Cohn, “Selecting an Agile Process: Comparing the Leading Alternatives”. http://www.mountaingoatsoftware. com/pres/Selecting021015.pdf, 2002.

[8] A. Cockburn, *Crystal Clear: A Human-Powered Methodology For Small Teams.* Addison Wesley, 2004.

[9] DSDM Tour, http://www.dsdm.org/tour/default.asp.

[10] P.B. Fagundes, *Framework for Comparing and Analyzing Agile Methods.* Master Thesis, UFSC, 2005. (in portuguese)

[11] J. Highsmith, *Agile Software Development Ecosystems.* Addison Wesley, 2002.

[12] C. Larman, V.R. Basili, “Iterative and Incremental Development, A Brief History”. *IEEE Computer*, 36-6, 2003, 47-56.

[13] T. L. Machado, *A Tool to Suport the Framework for Comparing and Analyzing Agile Methods.* Undergraduate Thesis. UFSC, 2005 (in portuguese)

[14] S. Palmer, “Feature Driven Development - Integrating Best Practices”. http://www.step10.com/process/Integrating BestPractices.html, 2003.

[15] M. Poppendieck, T. Poppendieck, *Lean Software Development: An Agile Toolkit for Software Development Managers.* Addison-Wesley Professional, 2003.

[16] I. Sommerville, *Software Engineering.* Addison-Wesley, São Paulo, 2003.

[17] K. Schwaber, M. Beedle, *Agile Software Development with SCRUM.* Prentice Hall, 2002.

A Proposal to Delegate GUI Implementation using a Source Code based Model

Marco Monteiro
School of Technology and Management, Polytechnic Institute of Leiria
Campus 2, Morro do Lena - Alto do Vieiro, Apartado 4163, 2411-901 Leiria, Portugal
marco@estg.ipleiria.pt

Paula Oliveira
Engineering Department, University of Trás-os-Montes e Alto Douro
Quinta de Prados, Apartado 1013, 5001-801 Vila Real, Portugal
poliveir@utad.pt

Ramiro Gonçalves
Engineering Department, University of Trás-os-Montes e Alto Douro
Quinta de Prados, Apartado 1013, 5001-801 Vila Real, Portugal
ramiro@utad.pt

Abstract

In this paper we propose an architecture whose main goal is to improve productivity in user interface development for data-intensive applications. This objective is to be achieved by defining a high level model that describes the user interface structure. That model will be integrated in the source code through non-functional language extensions. Our final goal is allowing developers to define user interface model by adding language extensions to the source code and then acquiring an external software package to which they delegate the implementation of the concrete user interface.

1. Introduction

As the size and complexity of information systems increases, building maintaining and integrating its applications is getting harder. To keep up with that complexity, there's a constant need to improve productivity of the development process in the software industry.

Driven by that need, in this paper we propose an architecture whose main goal is to improve productivity in Graphical User Interface (GUI) development for data-intensive applications. Nowadays developers tend to create GUI by composition of various components. Our final goal is allowing developers to define GUI using language extensions and then acquiring an external software package (which we'll call smart template) to which they delegate the implementation of the concrete GUI.

This paper is organized as follows. Research problem and alternative solutions are discussed in section 2. Proposed solution is presented in section 3 and conclusions on section 4.

2. Overview

Currently, a large number of projects use Component Based Development (CBD), which allows applications development by assembling a set of pre-manufactured components. Each component is a black-box entity, which can be deployed independently and is able to deliver specific services [1].

GUIs are composed of various graphical elements, such as buttons or input fields. When developing GUIs, both the presentation and behavior aspects of those elements are to be considered. Presentation aspects concern the appearance and layout of GUI elements and behavior is related to the interaction between themselves or between them and the underlying code. Using CBD, each GUI element is mapped to a component and presentation or behavior aspects are defined by its properties, methods and events. Also, by using Rapid Application Development (RAD) tools, GUI layout design is made visually through composition of components. Compared to older processes and methodologies the advent of CBD and RAD tools has certainly increased the productivity of GUI development. However, CBD still hasn't redeemed its promises of reuse and flexibility [2] and there's still a lot of risks, challenges and unresolved issues in CBD [3].

The issue that drove us to study and propose a solution to improve productivity in this area, is related to the process of components composition and configuration. On large or very large applications, the same component can be reused several times on different contexts, which is the main factor for the productivity improvement accomplished by CBD. However, as the number of instances and complexity of components increases, developers time is increasingly spent on the tedious tasks of composing layouts, configuring components and maintaining consistency in presentation and behavior aspects of the GUI components through the entire application.

Next, we'll present some alternative solutions for the described problem, namely Cascade Style Sheets and Templates on section 2.1, Frameworks on section 2.2 and Automatic GUI generation on section 2.3.

2.1. Cascade Style Sheets and Templates

In 1994, Håkon Wium Lie [4] proposed the Cascading Style Sheets (CSS) language, to describe the presentation of a document written in a markup language, usually Hypertext Markup Language (HTML). It enables the separation of document presentation from document content and ensures visual consistency through central configuration. Another concept used in web applications development is the page template, that's a pre-developed page layout used to create new pages from the same design. It's a concept adapted by Microsoft ASP.Net 2 Master Pages. As CSS, page templates also allows developer to create consistent layout and presentation through entire sites or group of pages. Unlike CSS that acts on individual HTML elements, templates acts on entire pages. Both CSS and page templates are great to define presentation aspects, but very limited when defining behavior or interactions between graphical elements. Håkon Wium Lie himself stated that *"CSS was primarily designed to present documents, not user interfaces"* [5].

2.2. Frameworks

The word framework has a lot of meanings, depending of the context. Within Object-Oriented (OO) design perspective, a framework is a set of cooperating classes that makes up a reusable design for a specific kind of software. A framework provides architectural guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations. A developer customizes the framework to a particular application by subclassing and composing instance of framework classes [6][7].

Frameworks are used in several application domains and at different levels of abstractions. In GUI development context, frameworks can control components creation, deployment, layout and configuration, leaving developer free of those repetitive tasks. Contrary to CSS and page templates, frameworks aren't limited to presentation issues, as they can handle GUI behavior aspects. In the GUI domain, frameworks can be generic, like Apache Struts or MonoRail (Model-View-Controller based frameworks), they can be available from commercial supplier to complement and integrate with their own components catalog or they can be custom made for a specific application or set of applications. In any of those cases, frameworks are proprietary, each having its own architecture, interface, classes and idiosyncrasies, therefore, requiring long learning processes. Also, code produced is interconnected with frameworks hierarchy of classes, which is against separation of concerns and creates non portable code.

2.3. Automatic GUI generation

Automatic generation is potentially the most productive method to develop GUI, since by definition it allows developers to delegate GUI creation to an external application. Proposed solutions to generate GUI automatically are mainly model-based systems, that attempt to formally describe the tasks, data, and users that an application will have, and then use this formal models to guide the generation of the GUI. Some systems automatically design the GUI and others provide design assistance to developers[8]. These models are abstract models, meaning that they don't specify exactly how the GUI is going to look, but rather what elements are to be shown and how they should behave. Systems will then use that abstract description to generate concrete interfaces for various devices. Trying to fit the same application on different devices with different capabilities [9], is what makes automatic GUI generation so complex. Despite a lot of research, model-based automatic GUI generation still hasn't become common in GUI development, in part because building models is an abstract process and better results are often achievable by a human designer in less time [10]. Abstract models can be complex to build and maintain, thus keeping models and applications concrete GUI synchronized can be problematic.

There are also some commercial tools, like Oracle Designer, that generates not only the GUI but complete applications, embracing all development cycle, which is not appropriated to development methodologies like extreme programming [11], where constant changes and very fast prototyping are required. They're usually based on relational database (DB), and use its metadata to generate layouts for data representation. However, resulting applications have behavior limitations, because DB are typically restricted to create, retrieve, update and delete (CRUD) operations.

3. Proposed solution

According to our initial motivation, main goal is to improve productivity in GUI development for data-intensive applications. Based on that premise and on preliminary studies, it was defined a set of 5 guidelines for the solution. First, it should integrate seamlessly with GUI commercial components, avoiding the need to recreate code to generate GUI elements. Second, it should be as easy to use as possible, allowing developers to concentrate on producing business related code. Separation of concerns should be encouraged. Third, it should be flexible, allowing rapid changes at any phase of the product development. Fourth, it should produce very fast prototypes, preferably in a fully automated manner. Fifth, it can't be limited to work with relational DB or XML as data source. Data persistence shouldn't be of any concern to the solution, as stated by separation of concerns principle.

Next, we'll present a description of the proposed solution, namely the general architecture in section 3.1, the GUI Model in section 3.2, the Binding Framework in section 3.3 and Smart Templates in section 3.4.

3.1. Architecture

There are 2 main characteristics for the proposed system architecture that distinguish it from others. First, it'll use an abstract model to characterize GUI, but instead of relying on specialized GUI models, based on XML or any other format, it'll be integrated with the source code through language extensions, which we'll call Graphical User Interface Language eXtensions (GUILX). Second, instead of generating GUI automatically, system will delegate that responsibility to external software packages, which we'll call smart templates (check Figure 1). System will provide the necessary resources to bind the source code model with external smart templates, who will handle GUI implementation for the business code included in the model.

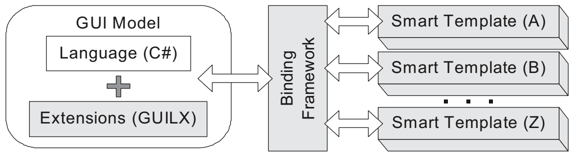


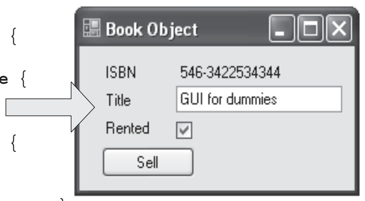
Figure 1. Proposed architecture.

3.2. GUI Model

Although it's not a common technique, using the source code as a model to generate GUI is not original. For instance, in 2004 Jelinek [9], used annotated source code to generate GUI. Despite some similar concepts, Jalinek model uses a tree-rewrite based language, as our model will use a mainstream OO language. That choice was made to facilitate integration with comercial GUI components. We'll use C#, but all concepts are also applicable to the Java language. Source code based models main advantage is the proximity between the model and the code we want to execute. This approach turns development process more flexible, as synchronization between business code and GUI model is no longer required. In our solution, the GUI model is composed of standard C# source code that defines business operations and GUILX language extensions to define the GUI related aspects. These extensions are declarative as they allow developers to define what GUI they want, instead of defining how to build it.

To better comprehend the model, lets analyze data-intensive applications. In those applications, GUI elements such as textboxs or grids, provide data for the user to read or write. Also, users can perform operations by activating events on GUI elements, like clicking on a button or a menu item. Comparing this reality with OO languages, such as C#, there are some similarities, as objects also have data, which can be encapsulated as properties and have associated operations called methods. Objects data and behavior can be mapped in GUI elements or set of elements. For example, considering a business class called "Book" with a read-only string property called "ISBN", a string property called "Title", a boolean property called "Rented" and a method called "Sell". By analyzing source code at run-time through applications metadata, we can generate the GUI elements and layout needed to represent instances of Book objects. GUI elements are chosen by the kind of language elements and accordantly to properties types and accessability (check Figure 2).

```
public class Book {
  public string ISBN {
     get { ... } }
  public string Title {
     get { ... }
     set { ... } }
  public bool Rented {
     get { ... }
     set { ... } }
  public void Sell { . . .}
}
```

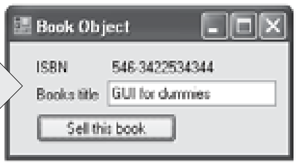


ISBN 546-3422534344
Title GUI for dummies
Rented ☑

Figure 2. Generation GUI from source code.

Although language metadata has already some useful information, it's not enough for defining a model to generate GUI. Filling that gap is GUILX language extensions responsibility, by enriching the metadata with structural information about GUI. This extensions are implemented by annotating the source code through .Net custom attributes, which provide a way for developers to extend native language by associating declarative information within the source code. "Show" is the first attribute defined on GUILX, indicating which language elements are meant to be available and with what description. If we analyze second example in Figure 3 and compare it with the first one, we can verify that the checkbox is not shown because ”Rented” property doesn’t have the ”Show” attribute. Also, ”Title” property and ”Sell” method have different descriptions.

```
public class Book {
   [Show]
   public string ISBN
      { get { ... } }
   [Show("Books title")]
   public string Title
      { get { ... }
        set { ... } }
   public bool Rented
      { get { ... }
        set { ... } }
   [Show("Sell this book")]
   public void Sell() { ... }
}
```



ISBN 546-3422534344
Books title GUI for dummies

Figure 3. Usage of GUILX ”Show” attribute.

Definition of GUILX is still in progress but is critical for the success of the solution. It must be rich enough to ensure that a complete prototype (even if a very basic one) can be generated from the GUI model but simple enough to avoid cluttering the source code with GUI related details.

3.3. Binding Framework

The ”Binding Framework” will be responsible for the connection of smart templates and the GUI model. It’ll allow the smart template to query the GUI model, to create object instances and to invoke methods of that objects. Also, it’ll serve as a controller, maintaining the execution context for the GUI elements, thus controlling navigation through entire application. It must always know what object instance is the user viewing and where to go or what to show next. Every time there’s a need to map an object instance to some GUI element, framework will notify the smart template to change interface accordantly.

3.4. Smart Templates

Proposed solution is designed to support various smart templates, one at a time. The idea is allowing developers to define a GUI model and then adquire a smart template to which they delegate all GUI implementation. Smart templates are specialized frameworks, developed by external entities that provide complete GUI services to the GUI model, by complying with the rules specified by the binding framework. There can be smart templates developed by different suppliers, for different devices and using completely different methods. One can generate GUI automatically, other can generate GUI partially and another can generate GUI from manual definitions.

4. Conclusion

Although the proposed solution is still in a embryonic state, we’re expecting productivity improvements by allowing developers to focus on business code development and reducing the repetitive tasks of composing layouts and configuring GUI components. Proposed solution is expected to be easier to use than custom frameworks, because learning a declarative language (GUILX) is easier than learning a complete class hierarchy. Also, in the proposed solution, business code doesn’t use any direct code related to GUI development, thus ensuring the separation of concerns principle. Compared to other methods of automatic GUI generation, we’re also expecting easier development, due to the fact that the model is integrated in the source code, therefore being easier to create and maintain that an abstract model. However, the success of the proposed solution depends on the definition of the GUILX language, which requires a correct balanced between simplicity and flexibility.

References

[1] C. Szyperski, *Component Oriented Programming*. Springer, 1998.

[2] H. de Bruin and H. van Vliet, “The future of component-based development is generation,” 2002.

[3] P. Vitharana, “Risks and challenges of component-based software development,” *Communications of the ACM*, vol. 46, no. 8, pp. 67–72, 2003.

[4] H. W. Lie, “Cascading html style sheets; proposal.” published 10 Oct 1994. Available from: http://www.w3.org/People/howcome/p/cascade.html; accessed on 28/01/2007.

[5] H. W. Lie, *Cascading Style Sheets*. PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, 2005.

[6] I. Jacobson, M. Griss, and P. Jonsson, *Software reuse: architecture, process and organization for business success*. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA, 1997.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1995.

[8] J. Nichols and A. Faulring, “Automatic interface generation and future user interface tools,” *ACM CHI 2005 Workshop on The Future of User Interface Design Tools*, 2005.

[9] J. Jelinek and P. Slavik, “Gui generation from annotated source code,” in *TAMODIA ’04: Proceedings of the 3rd annual conference on Task models and diagrams*, (New York, NY, USA), pp. 129–136, ACM Press, 2004.

[10] B. Myers, S. Hudson, and R. Pausch, “Past, present, and future of user interface software tools,” *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 7, no. 1, pp. 3–28, 2000.

[11] K. Beck, “Embracing change with extreme programming,” *Computer*, vol. 32, no. 10, pp. 70–77, 1999.

Cost-based Analysis of Multiple Counter-Examples

Flavian Vasile Samik Basu
Department of Computer Science, Iowa State University
Email: {flavian,sbasu}@cs.iastate.edu

Abstract

We study the problem of counterexample analysis, where a counterexample is presented as a sequence of states in the model that witnesses the violation of a desired specification. We claim that analyzing multiple counterexamples, instead of a single one, will provide valuable hints regarding the cause of the violation; specifically when the counterexamples are closely related. In this setting, we developed a cost-sensitive satisfiability method which takes as input the set of counterexamples represented as logical formulas in DNF and identifies the minimal *set of literals that must be altered in order to eliminate all counterexamples. We will refer to this minimal set of literals as the* causal set *of the counterexamples, as they can "cause" the removal of counterexamples. Minimality, in our context, is defined by the least possible update to the existing model that eliminates all counter-examples. It is computed using a cost model where any sequence that is not a counterexample is conservatively classified as a potential positive/correct sequence. We show that our notion of causality is equivalent to widely-used causality theories of Lewis and Pearl.*

1 Introduction

Counterexamples resulting from verification (e.g. model checking [10, 3]) of a system against a property, provide valuable insights for understanding the problems in modeling of the system or in the system itself. Since the size of the counterexample can be of similar size with the model being verified, it is important to develop error localization techniques which can further assist the user to zoom in on specific segments of the counterexample and take appropriate corrective measure.

Related Work. A number of techniques [1, 7, 6, 11] focus on obtaining the divergence between a counterexample and the closest positive example and classify the diverging point in the model as the cause of the corresponding counterexample.

Perhaps the closest to our approach is the work by Groce [5, 6] where the author shows that the solution follows the Lewis theory of causation. Furthermore, the author also relies on cost-sensitive satisfiability where the cost function depends on the concrete location of the literals and transition actions.

In [11] the authors primarily focus on addressing the problem of selecting the best positive trace in the case of multiple equally close positive examples. [1] also uses closest positive traces to find error locations. Once an error location is identified, that location is "marked" so that the subsequent verification process does not search the model along that location; the goal is to iteratively identify all error locations.

In [7] multiple counterexamples and closest positive traces are also used in order to report general characteristics in terms of their transitions. They identify error-causes on the basis of the set of transitions that are present in at least one counterexample, the set of transitions that are present in all of the counterexamples, the set of transitions that are present only in the counterexamples, and the set of transitions that are present in all of the counterexamples and in no positive trace.

Another promising approach for error-cause localization relies on identifying the error-causing inputs [12, 13]. The technique, referred to as *delta debugging*, is subsequently extended to consider internal variables of the models.

In contrast to the above techniques, which typically analyze one counterexample and depend on the presence of a positive example, we propose to analyze multiple counterexamples and identify the common aspects of all counterexamples.

Our Solution. The main motivation of our approach is to identify the *common/minimal* conditions under which all counterexamples can be eliminated. The intuition behind our approach is that there is information that can be exploited if all counterexamples are analyzed, specifically when these counterexamples are generated by the same inherent error in the system. A counterexample in our context is defined by a conjunction of constraints which must be satisfied for its feasibility. The set of counterexamples, denoted by `CE`, is therefore defined by a DNF formula where each disjunct corresponds to a single counterexample. The set of conditions under which all the counterexamples are eliminated, can be obtained by finding the solution of satisfiability of $\neg$CE. The satisfiable conditions are referred to as causal set for $\neg$CE.

Note that there can be multiple causal sets for $\neg$CE. As such, it is necessary to identify the best possible solution that can provide users the most *succinct* and *precise* information regarding the likely cause of the error. This is achieved by associating a cost with each condition and identifying a set of conditions with minimum cost. The cost function for a condition, in essence, measures the amount by which the correct behavior of existing model will be altered if the condition is used to eliminate counterexamples. In other words, *if we interpret the minimum cost solution as a set of possible changes to the system-model then the solution alters the behavior of the initial model as least as possible in order to remove all counterexamples*.

Another distinguishing feature of our technique is that in contrast of existing techniques which aim to align a counterexample to its closest positive example or a correct trace, our technique can select a solution from the space of new possible behaviors of the program if this proves to be more economical from the point of view of the cost analysis.

At a high level, the difference in techniques can be explained as follows. Given a system-model M and the error condition E, the space of all possible traces is divided in four areas: correct traces covered by the region ($M \wedge \neg E$), the set of all counterexamples ($M \wedge E$) and two other areas corresponding to the behavior not covered by the current model M: $\neg M \wedge E$ and $\neg M \wedge \neg E$. If an incorrect trace (from $M \wedge E$) is aligned to the closest correct trace (from $M \wedge \neg E$) then the difference is reported as a Lewis cause for the error. However the causality interpretation only holds because the solution space is restricted to traces in $M \wedge \neg E$, i.e., it is assumed that the only way to remove the error is to alter the counterexample to fit in $M \wedge \neg E$. However, the counterexample may be due to faulty modeling resulting in an incomplete model. In other words, there may be correct/desired behavior in the region $\neg M$ that is not modeled. As the primary purpose of error localization is to provide appropriate user-guidance for counterexample removal (and not automatic correction of models), it is important to also consider the $\neg M$ in addition to existing correct traces when analyzing counterexamples. The closest corrective measure to the counterexample may therefore lie outside the current model M.

Contribution.

1. We present an alternative way for error cause localization by analysis of multiple counterexamples. The technique can potentially identify causes that are beyond the existing state-space of the model being verified.
2. Our technique is based on applying cost-sensitive `MINSAT` method and identifies the minimal set of conditions under which all counterexamples can be eliminated. The technique is sound and complete.
3. We show that our definition of minimal causality of counterexamples is equivalent to Pearls theory of causation and implies a solution according to the widely used Lewis theory of causation. We provide valuable insights to the problem of causality identification of counterexamples.

Organization. The rest of the paper is organized as follows. Section 2 describes our technique for counterexample analysis (Subsection 2.1) and illustrates our technique using an example (Subsection 2.2). In Section 3, we present the relationship of our results with Pearl and Lewis theory of causality. Section 4 includes the future avenues of research.

2 Analysis of Counterexamples

We are interested in finding the minimum number of changes required to remove all counterexamples. In the following we show that the problem can be reduced to minimum-cost satisfiability problem and then offer an interpretation of its solutions as causes according to the well-established causal theories due to Pearl [9] and Lewis [8].

2.1 Causal set

2.1.1 Preliminaries

Given a model M and an error condition E, the counterexamples, if any, can be generated using the following formula:

$$\mathtt{CE} = \left[I(V_1) \bigwedge_{i=1}^{k-1} T(V_i, V_{i+1}) \right] \wedge E$$

In the above, $I(V_1)$ describes the valuations of variables (V_1) in the start states of M and $T(V_i, V_{i+1})$ describes the updates in the variable valuations at the i-th transition in the reachability graph from $I(V_1)$. The formula can be written as a DNF formula $\mathtt{CE} = Tr_1 \vee Tr_2 \vee \ldots \vee Tr_m$ where m is the number of counterexamples and Tr_i is a term described by conjunction of literals representing constraints over variables in M. Satisfiability of the above formula ensures the presence of at least one counterexample in the model M. Conversely, the satisfiability of its negation eliminates all counterexamples

$$\neg\mathtt{CE} = Cl_1(l_1, l_2, \ldots, l_n) \wedge Cl_2(l_1, l_2, \ldots, l_n) \wedge \ldots \wedge Cl_m(l_1, l_2, \ldots, l_n) \quad (1)$$

where $Cl_i(l_1, l_2, \ldots, l_n)$ is a clause defined as $\neg Tr_i$ and l_i is a literal corresponding to specific constraint over variables in M. We will also use Cl_i to denote $Cl_i(l_1, l_2, \ldots, l_n)$. The literals in $\neg$CE are negation of constraints required for the counterexamples. Satisfiability of a

```
 1: int main () {
 2: int i1, i2, i3; //input values
 3: int least = i1;
 4: int most = i1;
 5: if (most < i2)
 6:   most = i2;
 7: if (most < i3)
 8:   most = i3;   //(ERROR: least = i3)
 9: if (least > i2)
10:   least = i2;  //(ERROR: most = i2)
11: if (least > i3)
12:   least = i3;
13: assert (least <= most); //specification
14: }
```

Figure 1. Minmax.c

For: $\texttt{i2} < \texttt{i3} < \texttt{i1}$	For: $\texttt{i2} < \texttt{i1} < \texttt{i3}$
3: $\texttt{least}^3_{=1}$	3: $\texttt{least}^3_{=1}$
4: $\texttt{most}^4_{=1}$	4: $\texttt{most}^4_{=1}$
5: $\texttt{most}^4_{\geq 2}$	5: $\texttt{most}^4_{\geq 2}$
7: $\texttt{most}^4_{\geq 3}$	7: $\texttt{most}^4_{< 3}$
9: $\texttt{least}^3_{> 2}$	8: $\texttt{most}^8_{= 3}$
10: $\texttt{most}^{10}_{= 2}$	9: $\texttt{least}^3_{> 2}$
11: $\texttt{least}^3_{> 3}$	10: $\texttt{most}^{10}_{= 2}$
12: $\texttt{least}^{12}_{= 3}$	11: $\texttt{least}^3_{\leq 3}$

Figure 2. Counterexamples: Error at Line 10.

set literals in $\neg\texttt{CE}$, containing at least one literal from each conjunct (clause), will result in the satisfiability of $\neg\texttt{CE}$ and elimination of all the counterexamples. We will refer to this set as *causal set*.

Example. Consider the Example in Figure 1 which takes as input three variables `i1`, `i2` and `i3`, and assigns the largest and smallest of these variables to `most` and `least` respectively. Assume that, there exists an error in the program where the statement at Line 10 is written as `most=i2` instead of the correct version `least=i2`.

There will be two counterexamples that witness the violation of the assertion at Line 13: Figure 2. The counterexamples are shown along with the conditions on inputs that are required for their feasibility. The constraints at each line number of the counterexample are represented as $\texttt{var}^L_{\texttt{op}\ n}$ denoting the constraint $\texttt{op}$ ($\in \{<, >, =, \leq, \geq, \neq\}$) between an input variable in and variable $\texttt{var}$ ($\in \{\texttt{least}, \texttt{most}\}$) which was assigned to at Line L. For example, the constraint at Line 3 for both counterexamples states that valuation of `least` at Line 3 is equal to the valuation of `i1`.

Each counterexamples is, therefore, represented as conjunction of constraints. Proceeding further, $\neg\texttt{CE}$ is:

$$\begin{bmatrix} \texttt{least}^3_{\neq 1} \vee \texttt{most}^4_{\neq 1} \vee \texttt{most}^4_{<2} \vee \texttt{most}^4_{<3} \vee \texttt{least}^3_{\leq 2} \vee \\ \texttt{most}^{10}_{\neq 2} \vee \texttt{least}^3_{\leq 3} \vee \texttt{least}^{12}_{\neq 3} \end{bmatrix}$$
$$\bigwedge$$
$$\begin{bmatrix} \texttt{least}^3_{\neq 1} \vee \texttt{most}^4_{\neq 1} \vee \texttt{most}^4_{<2} \vee \texttt{most}^4_{\geq 3} \vee \texttt{most}^8_{\neq 3} \vee \\ \texttt{least}^3_{\leq 2} \vee \texttt{most}^{10}_{\neq 2} \vee \texttt{least}^3_{>3} \end{bmatrix} \tag{2}$$

2.1.2 `MinCostSat` Satisfiability

Once a logical formula that stands for the negation of all counterexamples ($\neg\texttt{CE}$) is obtained, identifying the causal set Δ of literals which can satisfy $\neg\texttt{CE}$ is straightforward: Δ=$\texttt{SAT}(\neg\texttt{CE})$. Note that there can be more than one causal set. Our goal is to obtain a compact and precise causal set. We formulate the problem as a minimization problem.

Definition 1 (`MinCostSat` Satisfiability) *Given a Boolean formula F in conjunctive normal form with m clauses and n literals ($l_1, l_2, \ldots, l_n$), and a cost function that assigns non-negative costs $\mathcal{C}(l_i)$ to l_i, the `MinCostSat` problem is the problem of finding a set $\Delta \subseteq \{l_1, l_2, \ldots, l_n\}$ which satisfies F and minimizes the objective function: $\sum_{j=1}^{|\Delta|} \mathcal{C}(l_j)$ where $|\Delta|$ is the size of Δ.* □

`MinCostSat` problem is NP-C as the `SAT` is a special case of `MinCostSat` where the cost of every literal is 0. Existing technique, e.g. [4], can be effectively applied to solve the problem.

A Simple Cost model. In order to associate a non-negative cost $\mathcal{C}(l_i)$ to the literal l_i, we will consider the minimum number of literals required to *cover* all clauses in $\neg\texttt{CE}$ This leads to a simple cost function:

$$\mathcal{C}(l_i) = \frac{1}{|\texttt{cover}^-(l_i)| -_a |\texttt{cover}^-(\neg l_i)|} \tag{3}$$

where $-_a$ denotes absolute difference and $\texttt{cover}^-(l_i)$ is defined as follows.

Definition 2 ($\texttt{cover}^-$) *Given a set of counterexamples denoted by the formula $\texttt{CE}$, $\texttt{cover}^-(l_i)$ returns the set of clauses in which l_i appears in $\neg\texttt{CE}$. $|\texttt{cover}^-(l_i)|$ denotes the size of $\texttt{cover}^-(l_i)$.* □

The Equation 3 assigns a cost value to a literal l_i which is inversely proportional to the number of clauses in $\neg\texttt{CE}$ that can be potentially made satisfiable (number of counterexamples that can be removed) by satisfying l_i. Observe that, the $|\texttt{cover}^-(\neg l_i)|$ is subtracted as presence of $\neg l_i$ in $\neg\texttt{CE}$ reduces the effect of l_i in eliminating counterexamples. For the counterexamples in Figure 2, $|\texttt{cover}^-(\texttt{least}^3_{\neq 1})|$ is 2 and $|\texttt{cover}^-(\texttt{least}^3_{=1})|$ is 0 (see Equation 2).

The above cost model ensures the *succinctness* of the solution where succinctness is defined by the number of literals needed for satisfiability of $\neg\texttt{CE}$. Such a simple cost model is not precise as it does not take into consideration the location of the counterexample at which the literal is appearing. Typically, the initial part of the counterexamples result from the initialization and all counterexamples will include this initialization part. As such following the simple cost model, the minimum cost satisfiability will always result in solutions that include the literals from the initialization segment (which may not be the best possible solution for counterexample removal). To counter this situation, we also take into consideration the location at which the literals appear when computing their cost.

A cost model for precision. Recall that, a literal l_i is in our solution implies that we are using l_i to remove all counterexamples which contains $\neg l_i$; essentially we are saying that a possible way to remove counterexamples is to alter $\neg l_i$ to l_i. We claim that a l_i is more precise that l_j if and only if $\neg l_i$ covers less number of model-traces compared to that covered by $\neg l_j$. The intuition is to identify a solution which effects the existing model minimally.

To add precision to the cost function, we identify the *potential* number of traces, a literal can cover. I.e., the cost function of Equation 3 is refined as

$$\mathcal{C}(l_i) = \frac{\text{Potential program coverage of } l_i}{|\texttt{cover}^-(l_i)| -_a |\texttt{cover}^-(\neg l_i)|}$$

The numerator is computed by considering the maximum length of the counterexamples in the model and the height at which the literal (more specifically its negation[1]) appears. For example, in case of programs, the height of a literal is difference between the size of the program and the line number as which the literal appears. Assuming a branching factor of 2 per-state in any trace of the model, a literal at height h can cover 2^h traces. A weight w can be associated with the literal which can be fine-tuned by the user to identify appropriate number of traces covered by the literal for branching factor not equal to 2.

For example, if the average branching factor of a program is k, then w can be computed as follows: $k^h = 2^{w \times h}$, i.e., $w = log\ k$. The w, in essence, establishes the belief of the user about the model-behavior/traces that the literal covers. Sub-unitary values of w represent a reduction of the number of possible traces covered by l_i and supra-unitary values an increase in that number. By varying the values of w, the user can keep the solution really close to the errors (act conservatively) or explore more of the options beyond the existing model-space. Therefore,

$$\mathcal{C}(l_i) = \frac{2^{w(l_i) \times h(l_i)}}{|\texttt{cover}^-(l_i)| -_a |\texttt{cover}^-(\neg l_i)|} \tag{4}$$

In the above, $w(l_i)$ is the tunable parameter and $h(l_i)$ is the height at which negation of l_i appears in the counterexamples. Intuitively, a literal l_i in $\neg\texttt{CE}$ is costly if:

1. its negation $\neg l_i$ appears higher up in the counterexample: as potentially more model behavior may get effected if $\neg l_i$ is altered to l_i to remove a counterexample;

2. the user provides a high valuation for w: again estimating higher number of model behavior depending on $\neg l_i$; or

3. the counterexample coverage of $\neg l_i$ is low: altering $\neg l_i$ to l_i will not remove many counterexamples.

The above cost model will maximize our chances of obtaining a solution for removal of all counterexamples and at the same time will allow the user to fine-tune the solution using the factor w.

Remark 1 *We are not considering any positive examples; as such any trace that is not a counterexample is deemed positive example. This assumption allows us to obtain solutions beyond existing model-space and possibly to attain precise result as per the user requirement.*

Specifically, there can be cases where aligning counterexample to a positive example may not realize the best possible solution due to incorrect modeling. So, one of the alternative will be to empower the user with the choice to use appropriate alignment. This is achieved by using w in the Equation 4.

Let $\Delta_s = \{l_1, l_2, \ldots, l_k\}$ be the causal set of literals required for the satisfiability of $\neg\texttt{CE}$. We will write $\Delta_s \models \neg\texttt{CE}$ to denote that under the literals in set Δ_s the formula $\neg\texttt{CE}$ is satisfiable. The overall cost of the solution is given by $\mathcal{C}(\Delta_s) = \sum_{i=1}^{|\Delta_s|} \mathcal{C}(l_i)$ where $l_i \in \Delta_s$.

2.2 Illustrative Example

Consider the example in Figures 1, 2 and Equation 2. Observe that there are 4 locations with branching factor 2 each and 9 locations with branching factor 1. Therefore, the average branching factor of the overall program is computed as follows:

$$k = \frac{\text{Total number of branching degree}}{\text{Total size of the program}} = \frac{4 \times 2 + 9 \times 1}{13} = 1.307$$

The cost of the literals in Equation 2 is computed using the Equation 4 where $w = logk$.

$$\mathcal{C}(\texttt{least}^3_{\neq 1}) = 7.3,\ \mathcal{C}(\texttt{most}^4_{\neq 1}) = 5.6,\ \mathcal{C}(\texttt{most}^4_{<2}) = 4.3,$$
$$\mathcal{C}(\texttt{most}^4_{<3}) = \infty,\ \mathcal{C}(\texttt{least}^3_{\leq 2}) = 1.5,\ \mathcal{C}(\texttt{most}^{10}_{\neq 2}) = 1.1,$$
$$\mathcal{C}(\texttt{least}^3_{\leq 3}) = \infty,\ \mathcal{C}(\texttt{least}^{12}_{\neq 3}) = 1.3,\ \mathcal{C}(\texttt{most}^4_{\geq 3}) = \infty,$$
$$\mathcal{C}(\texttt{least}^3_{>3}) = \infty,\ \mathcal{C}(\texttt{least}^8_{\neq 3}) = 3.8,$$

For example, $\mathcal{C}(\texttt{least}^3_{\neq 1}) = \frac{2^{log(1.307) \times (13-3)}}{2} = 7.3$

[1]Observe that l_i appears in $\neg\texttt{CE}$ because of the presence of $\neg l_i$ in the counterexample.

For: `i2` $<$ `i3` $<$ `i1`		For: `i2` $<$ `i1` $<$ `i3`		For: `i1` $<$ `i2` $<$ `i3`	
3:	$\texttt{least}^{3}_{=1}$	3:	$\texttt{least}^{3}_{=1}$	3:	$\texttt{least}^{3}_{=1}$
4:	$\texttt{most}^{4}_{=1}$	4:	$\texttt{most}^{4}_{=1}$	4:	$\texttt{most}^{4}_{=1}$
5:	$\texttt{most}^{4}_{\geq 2}$	5:	$\texttt{most}^{4}_{\geq 2}$	5:	$\texttt{most}^{4}_{<2}$
7:	$\texttt{most}^{4}_{\geq 3}$	7:	$\texttt{most}^{4}_{<3}$	6:	$\texttt{most}^{6}_{=2}$
9:	$\texttt{least}^{3}_{>2}$	8:	$\texttt{least}^{8}_{=3}$	7:	$\texttt{most}^{6}_{<3}$
10:	$\texttt{most}^{10}_{=2}$	9:	$\texttt{least}^{8}_{>2}$	8:	$\texttt{least}^{8}_{=3}$
11:	$\texttt{least}^{3}_{>3}$	10:	$\texttt{most}^{10}_{=2}$	9:	$\texttt{least}^{8}_{>2}$
12:	$\texttt{least}^{12}_{=3}$	11:	$\texttt{least}^{8}_{\leq 3}$	11:	$\texttt{least}^{8}_{\leq 3}$

Figure 3. Counterexamples: Error at Lines 8, 10.

Some of the costs are equal to infinity as the corresponding literal appears in positive and negative form equal number of times in the counterexamples. The minimum cost solution for $\neg$`CE` (Equation 2) using the associated cost is $\texttt{most}^{10}_{\neq 2}$ which correctly identifies the statement (Line 10 contains the erroneous assignment to `most`) where error was injected in the example (Figure 1).

Next consider the case, where two errors are inserted in the example Figure 1; in addition to the error at Line 10, the statement at Line 8 is also incorrectly written as `least=i3` (correct version is `most`=i3). The corresponding counterexamples are shown in Figure 3 and the $\neg$`CE` is

$$\begin{bmatrix} \texttt{least}^{3}_{\neq 1} \vee \texttt{most}^{4}_{\neq 1} \vee \texttt{most}^{4}_{<2} \vee \texttt{most}^{4}_{<3} \vee \texttt{least}^{3}_{\leq 2} \vee \\ \texttt{most}^{10}_{\neq 2} \vee \texttt{least}^{3}_{\leq 3} \vee \texttt{least}^{12}_{\neq 3} \end{bmatrix} \\ \bigwedge \\ \begin{bmatrix} \texttt{least}^{3}_{\neq 1} \vee \texttt{most}^{4}_{\neq 1} \vee \texttt{most}^{4}_{<2} \vee \texttt{most}^{4}_{\geq 3} \vee \texttt{least}^{8}_{\neq 3} \vee \\ \texttt{least}^{8}_{\leq 2} \vee \texttt{most}^{10}_{\neq 2} \vee \texttt{least}^{8}_{>3} \end{bmatrix} \\ \bigwedge \\ \begin{bmatrix} \texttt{least}^{3}_{\neq 1} \vee \texttt{most}^{4}_{\neq 1} \vee \texttt{most}^{4}_{\geq 2} \vee \texttt{most}^{6}_{\neq 2} \vee \texttt{most}^{6}_{\geq 3} \vee \\ \texttt{least}^{8}_{\neq 3} \vee \texttt{least}^{8}_{\leq 2} \vee \texttt{least}^{8}_{>3} \end{bmatrix} \quad (5)$$

Taking the same valuation of w and the cost equation from Equation 4, the corresponding costs are

$\mathcal{C}(\texttt{least}^{3}_{\neq 1}) = 4.9$, $\mathcal{C}(\texttt{most}^{4}_{\neq 1}) = 3.7$, $\mathcal{C}(\texttt{most}^{4}_{<2}) = 8.5$,
$\mathcal{C}(\texttt{most}^{4}_{<3}) = \infty$, $\mathcal{C}(\texttt{least}^{3}_{\leq 2}) = 2.9$, $\mathcal{C}(\texttt{most}^{10}_{\neq 2}) = 1.1$,
$\mathcal{C}(\texttt{least}^{3}_{\leq 3}) = 1.7$, $\mathcal{C}(\texttt{least}^{12}_{\neq 3}) = 1.3$, $\mathcal{C}(\texttt{most}^{4}_{\geq 3}) = \infty$,
$\mathcal{C}(\texttt{least}^{8}_{\neq 3}) = 1.9$, $\mathcal{C}(\texttt{least}^{8}_{\leq 2}) = 1.5$, $\mathcal{C}(\texttt{least}^{8}_{>3}) = 0.8$,
$\mathcal{C}(\texttt{most}^{4}_{\geq 2}) = 8.5$, $\mathcal{C}(\texttt{most}^{6}_{\neq 2}) = 6.5$, $\mathcal{C}(\texttt{most}^{6}_{\geq 3}) = 5.0$

The minimum cost solution for satisfiability of $\neg$`CE` in Equation 5 is the set containing $\texttt{least}^{8}_{>3}$ and $\texttt{most}^{10}_{\neq 2}$. The set correctly identifies the erroneous assignments at Lines 8 and 10.

3 Causal Interpretation of MinCost Solution

Causality plays a major role in error explanation. We investigate and analyze our causal properties using Lewis [8] and Pearl [9] theory of causality and aim to provide a better understanding of the causal analysis of counterexamples.

3.1 Pearl Theory of Causality

The cornerstone of Pearl's theory of causality is the axiom named the *Causal Markov Condition* which states that a variable X is independent of every other variable (except X's effects) conditional on all of its direct causes (its Markov Blanket).

Markov Blanket. Suppose V is a set of random variables (corresponding to events), and $V_A \subseteq V$ and $V_C \subseteq V$. We say that V_C is the set of direct causes of V_A relative to V if:

1. The variables in V_C are causes of V_A. In the probabilistic sense that means that the probability of V_A occurring given that V_C occurred is bigger than in the default case: $P(V_A|V_C) > P(V_A)$[2].

2. For any subset $V_i \subseteq V \backslash (V_A \cup V_C)$
$$P(V_A|V_C) = P(V_A|V_C \cup V_i)$$

3. No proper subset of V_C satisfies (1) and (2)

Markov Blanket for removing all counterexamples. Let `CE` be the counterexamples in the model. Following the Markov blanket conditions (see above), the causal set Δ_s is said to be the Markov blanket of all counterexamples if

1. $\Delta_s \models \neg$`CE`.

2. Literals in Δ_s leads to satisfiability of $\neg$`CE` irrespective of the other literals. The statement is trivially true as conditional probability of satisfying $\neg$`CE` under Δ_s is 1 ($P(\neg\texttt{CE}|\Delta_s) = 1$).

3. No proper subset of Δ_s satisfies (1) and (2)

From the above discussion, it follows that the `MinCostSat` solution Δ_s of $\neg$`CE` is the Markov Blanket of $\neg$`CE`. As Δ_s is obtained via satisfiability, it must satisfy the above conditions (1) and (2). Condition (3) is ensured by minimum cost satisfiability condition.

3.2 Lewis Theory of Causation

We showed that the solution of `MinCostSat` can be interpreted as a cause according to Pearl's definition. We now move to another widely known definition of causality in the model-checking community which is due to Lewis.

In [6], the author effectively explains the Lewis theory of causality as: "For Lewis, an effect E is dependent on a cause C at a world W iff at the world(s) most similar to W in which $\neg C$, it is also the case that $\neg E$. Causality does not

[2]$P(A|B)$ is conditional probability of A under B

depend on the impossibility of $\neg C$ and E being simultaneously true in any possible world, but on what happens when we alter W as little as possible, other than to remove the possible cause C."

MinCostSat follows Lewis causality theory. Let $\Delta_s = \{l_1, l_2, \ldots, l_k\}$ be the causal set of $\neg\texttt{CE}$ where (following Equation 1)

$$\neg\texttt{CE} = Cl_1(l_1, l_2, \ldots, l_n) \wedge \ldots \wedge Cl_m(l_1, l_2, \ldots, l_n)$$

We will consider a "world" being described by the valuations of the literals. As such, the world in which $\neg\texttt{CE}$ is satisfiable requires the satisfiability of the literals in Δ_s; $\Delta_s \models \neg\texttt{CE}$. Consider any closest world obtained from Δ' via altering the valuation of one of the literal in Δ_s, i.e., $l_i \in \Delta_s$ and $\neg l_i \in \Delta'$. Δ_s and Δ' agree on all valuations of literals except l_i. We need to prove that in any such closest world Δ', the $\neg\texttt{CE}$ is not satisfied. The proof follows from the discussion below.

Observe that,

$$\texttt{cover}^-(l_i) \cap \texttt{cover}^-(\neg l_i) = \emptyset \qquad (6)$$

as both l_i and $\neg l_i$ cannot appear in the same Cl_j in $\neg\texttt{CE}$. Furthermore, the number of Cl_js covered by the literals in $\Delta - \{l_i\}$ is less than the number of clauses in $\neg\texttt{CE}$. This is because Δ_s is obtained by minimum satisfiability of $\neg\texttt{CE}$ and every literal is responsible for satisfiability of at least one clause. Therefore,

$$\sum_{j \neq i, j=1}^{|\Delta_s|} |\texttt{cover}^-(l_j)| < m \text{ where } l_j \in \Delta_s - \{l_i\} \qquad (7)$$

It follows from Equations 6, 7 and the fact that the Δ_s covered all clauses in $\neg\texttt{CE}$

$$\texttt{cover}^-(\neg l_i) \subseteq \texttt{cover}^-(\Delta_s - \{l_i\})$$

The clause that is covered by l_i is not satisfiable by replacing l_i by $\neg l_i$. Therefore, $\Delta' \not\models \neg\texttt{CE}$; the closest new world defined by the literals in Δ' does not satisfy the clauses in $\neg\texttt{CE}$.

In conclusion, we proved that MinCostSat produces a set of causes that are valid under both Pearls and Lewis definitions of causality. A subtle point needs to be reinforced: these are causes for the event $\neg\texttt{CE}$ and not for the event CE. We do not return the causes (ways) for producing the error but the causes of removal of errors.

4 Discussion

In this paper, we explored minimum cost satisfiability to identify ways to remove counterexamples with minimal changes to the existing model. As future work, we will consider also the positive examples in the model and investigate the usage of our technique as a complimentary approach to the existing techniques. Another avenue for development is an iterative application of the current method. If the corrective measure suggested by our solution is automatically applied to the existing model M, a new model M' will be generated. The new model can be model checked and, in case of errors, can be fixed iteratively. The iterative process may or may not converge. Furthermore, applying automatic corrective measures requires that logic of the program is preserved at each iteration. As such, it is necessary to add this program logic information as requirements; however coming up with such requirements may be non-trivial. At present we are implementing our technique using CBMC model checker [2] and we will evaluate applicability of our technique in such iterative process.

Acknowledgment. This research has been supported in part by the NSF grant 0509340.

References

[1] T. Ball, M. Naik, and S. Rajamani. From symptom to cause: Localizing error in counterexample traces. In *Proceedings of POPL*, 2003.

[2] CBMC. Bounded model checking for ansi-c. Available at http://www.cs.cmu.edu/~modelcheck/cbmc/.

[3] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verifi cation of fi nite-state concurrent systems using temporal logic specifi cations. *ACM TOPLAS*, 8(2), 1986.

[4] Z. Fu and S. Malik. Solving the minimumcost satisability problem using sat based branch and bound search. In *Proceedings of CAD*, 2006.

[5] A. Groce. Error explanation with distance metrics. In *Proceedings of TACAS*, 2004.

[6] A. Groce. *Error Explanation and Fault Localization with Distance Metrics*. PhD thesis, Carnegie Mellon University, 2005.

[7] A. Groce and W. Visser. What went wrong: Explaining counterexamples. In *Proceedings of SPIN Workshop on Model Checking of Software*, 2003.

[8] D. K. Lewis. *Counterfactuals*. Harvard University Press, 1973/revised 1986.

[9] J. Pearl. *Causality: Model Reasoning and Inference*. Cambridge University Press, 2000.

[10] J. Queille and J. Sifakis. Specifi cation and verifi cation of concurrent systems in Cesar. In *Proceedings of the ISP*, 1982.

[11] S. Y. Shen, Y. Qin, and S. Li. Localizing errors in counterexample with iteratively witness searching. In *Proceedings of ATVA*, 2004.

[12] A. Zeller. Yesterday, my program worked. today, it does not. why? In *ACM SIGSOFT Software Engineering Notes, v.24 n.6*, 1999.

[13] A. Zeller and R. Hildebrandt. Simplifying and isolating failure-inducing input. In *IEEE Transactions on Software Engineering*, 2002.

Common Coupling as a Measure of Reuse Effort in Kernel-Based Software

Liguo Yu
CS and Informatics
Indiana University South Bend
South Bend, IN, USA
ligyu@iusb.edu

Stephen R. Schach
EECS Department
Vanderbilt University
Nashville, TN, USA
srs@vuse.vanderbilt.edu

Kai Chen
SSE Research
Motorola Labs
Schaumburg, IL, USA
kai.chen@motorola.com

Abstract

An obstacle to software reuse is the large number of major modifications that frequently have to be made as a consequence of dependencies within the reused software components. In this paper, common coupling is categorized and used as a measure of the dependencies between software components. Five open-source operating systems were analyzed and compared with respect to common coupling and reuse effort. We conclude that the way that common coupling is implemented in the Mach and BSD kernels induces few dependencies between software components, which required less effort in order to be reused to produce Darwin.

1. Introduction

Software reuse has become a topic of interest within the software community because of its potential benefits. These include increased productivity and quality, and decreased cost and time-to-market. Obviously the biggest savings are to be found in large-scale reuse, that is, the reuse of a large portion of an existing software product. Unfortunately, reuse generally does not scale up for large-scale software components [1] [2] [3] [4] [5].

One problem is that large software components may be dependent on other components. On the one hand, suppose that the components of a software product are classes that communicate exclusively by message passing. The dependency between the components is low, and it should be possible to reuse one component in a new software product with little difficulty. But if a software product consists of components, all of which reference a large number of global variables, it may be impossible to reuse any one component in a new product without first totally redesigning that component, thereby all but defeating the purpose of reuse. The problem becomes even more complex if we wish to build a new software product by reusing two disparate components produced by two different organizations, particularly if the two reused components have to interact with one another as well as with the rest of the new product.

Many software products, including operating systems and database management systems, are kernel-based. That is, each implementation consists of required kernel components, together with specific optional architecture-specific or hardware-specific nonkernel components.

Coupling is a measure of the degree of interaction between two software components. It reflects the modifiability and the maintainability of a software product [6]. There are many different categorizations of coupling, all of which include common (global) coupling (two software components are common coupled if they reference the same global variable). Certain types of coupling, especially common coupling, are considered to present risks for software development [7]. For example, to reuse a kernel component in another software product, it is important that the kernel component should have minimal dependency on other components. Accordingly, it is important that the kernel components have as little common coupling as possible.

In a previous study [8], we defined a new categorization of common coupling within kernel-based software, and used it to measure the maintenance effort. In this paper, we extend our categorization and use it to evaluate reuse effort in kernel-based software.

2. Common Coupling and Reuse Effort

Common coupling induces dependencies between software components. As described in [8], these dependencies are induced by the definition-use mechanism; we say component `C1` is dependent on component C2 via global variable gv if C1 uses gv and C2 defines gv (that is, if C2 changes the value of gv and C1 utilizes that value).

In our previous study, global variables were categorized in terms of five categories, as summarized in Table 1 [8]. We introduced this categorization of common coupling within the context of kernel maintenance and discussed the impact of the existence of global variables in each category on kernel maintenance. In this paper, our categorization of common coupling is applied to software reuse effort.

Table 1. Categorization of global variables [8]

Category	Description
1	A global variable defined in one or more kernel modules but not used in any kernel modules.
2	A global variable defined in one kernel module and used in one or more kernel modules.
3	A global variable defined in more than one kernel module, and used in one or more kernel modules
4	A global variable defined in one or more nonkernel modules and used in one or more kernel modules.
5	A global variable defined in one or more nonkernel modules and defined and used in one or more kernel modules.

We consider two types of reuse. We refer to the reuse of one or more independent kernel components as *kernel-component reuse* and the reuse of the entire kernel as *entire-kernel reuse*. When we wish to refer to either kernel-component reuse or entire-kernel reuse, we use the umbrella term *kernel reuse*.

Dependencies induced by common coupling affect the reuse effort; in general, more effort is needed to reuse a component with a large number of global variables. However, reuse effort is also affected by the category into which each global variable falls.

A category-1 global variable is not used in a kernel component, so definitions of the global variable in other components (kernel or nonkernel) cannot affect kernel components. All kernel components are independent with respect to this global variable. Accordingly, the presence of a category-1 global variable will not cause difficulties for kernel reuse. Therefore, no kernel reuse effort is associated with a category-1 global variable.

A category-2 or category-3 global variable is defined in a kernel component but not in any nonkernel component. It is used in kernel components. A category-2 or category-3 global variable therefore induces dependencies between kernel components. A kernel component that defines a category-2 or category-3 global variable can affect the reuse effort of any kernel component that uses that global variable. Turning to the reuse effort of the entire kernel, this is not affected by the presence of a category-2 or category-3 global variable because there is no definition outside the kernel.

A kernel component that uses a category-4 or category-5 global variable is dependent upon nonkernel components that define that global variable. Thus, the presence of a category-4 or category-5 global variable in a kernel component negatively impacts both kernel-component reuse as well as entire-kernel reuse. Hence, more effort for kernel reuse is associated with category-4 and category-5 global variables than for categories 2 and 3.

Table 2 summarizes the impact of global variables in different categories on kernel reuse effort.

Table 2. The impact of global variables on reuse

Category number	Kernel-component reuse effort	Entire-kernel reuse effort
1	No impact	No impact
2	Negative impact	No impact
3	Negative impact	No impact
4	Negative impact	Negative impact
5	Negative impact	Negative impact

3. New Terminology

As indicated in the previous section, reuse is hampered by definitions in nonkernel components that affect uses in kernel modules. In order to be able to quantify this phenomenon, we introduce additional terminology which is utilized in Section 5.

- Terminology 1: A definition of a global variable that induces a dependency of a kernel component on another component is called a component-dependency-inducing definition.
- Terminology 2: A global variable is kernel-on-nonkernel-dependency-inducing if it induces a dependency of a kernel component on a nonkernel component.
- Terminology 3: A kernel component is use-dependency-induced if it contains a use of a kernel-on-nonkernel-dependency-inducing variable.
- Terminology 4: A nonkernel component is definition-dependency-inducing if it contains a definition of a kernel-on-nonkernel-dependency-inducing variable.

4. Mach, FreeBSD, and Other Open-Source Operating Systems

In 2001, Apple released OS X, an operating system for Macintosh computers [9]. OS X is structured around Darwin, an open-source core. Darwin was produced through the integration of two existing open-source operating systems, Mach [10] and FreeBSD [11]. However, neither Mach nor FreeBSD is a ready-to-use building blocks. Modifications had to be made and effort had to be spent on each of those components in order to incorporate them into the new product [12] [13]. Besides Mach and FreeBSD, Darwin also incorporates some components from OpenBSD and NetBSD.

The reuse of Mach and FreeBSD was essentially entire-kernel reuse; the reuse of OpenBSD and NetBSD was kernel-component reuse. We wished to understand the effort involved in modifying the different pieces that were reused to produce Darwin. Accordingly, we studied the two major pieces from which Darwin 7.x was built [14]: version 3.0 of Mach and version 5.1 of FreeBSD. Besides FreeBSD and Mach, we also studied the latest versions of OpenBSD and NetBSD.

Linux is undoubtedly the most widely used open-source operating system. In order to gain additional insights into differences or similarities between Mach, FreeBSD, OpenBSD, and NetBSD from the viewpoint of reuse, we decided to study Linux as well.

More precisely, we studied the following open-source operating systems: Mach 3.0, FreeBSD 5.1, OpenBSD 3.3, NetBSD 1.6, and Linux 2.4.20. All these operating system are kernel-based and written in C. In this paper, a component is defined to be a source code file (".c" file or ".h" file). The size of the product is measured in thousands of lines of code (KLOC). Data regarding the structure of these systems are provided in Table 3, in which the BSD and Linux data are taken from [15].

5. Case Studies

5.1. Common Coupling in General

We analyzed common coupling in the Mach, FreeBSD, OpenBSD, NetBSD, and Linux operating systems. Global variables appearing in kernel components were identified by the Linux cross-referencing tool, lxr. Every instance of a global variable was determined to be either a definition or a use of that variable. An overview of our results is summarized in Table 4. As shown in the Table, there are 77 distinct global variables in the Mach kernel. Altogether, there are 332 instances of global variables in Mach kernel components. (For the sake of comparison, we observe that Linux has many more instances of global variables in kernel and nonkernel components than Mach or the three BSDs.)

Table 3. The structure of five operating systems

Operating system	Kernel components	Nonkernel components	Kernel KLOC	Total KLOC
Mach	71	792	30	345
FreeBSD	131	3353	108	1793
OpenBSD	81	4569	55	1825
NetBSD	85	11527	64	3329
Linux	26	9407	14	4260

Table 4. Global variables in five operating systems

Operating system	Number of global variables	Number of instances of global variables in kernel components	Number of instances of global variables in nonkernel components
Mach	77	332	228
FreeBSD	75	483	770
OpenBSD	75	343	521
NetBSD	66	378	1222
Linux	99	1022	14088

In general, global variables induce dependencies between software components and make the components difficult to reuse. However, as outlined in Section 2, the different categories of global variables have different effects on the reuse effort. To understand how global variables in the open-source operating systems affect the kernel reuse effort, each global variable was assigned to one of the five categories.

5.2 Dependency of a Kernel Component

In order to analyze dependencies within the kernel and between kernel components and nonkernel components, we need to examine definitions and uses of global variables in more detail.

As stated in Section 3, a definition of a global variable that induces a dependency of a kernel component on another component is called a *component-dependency-inducing definition* (CDID). A reusable component should be dependent on as few other components as possible. A global variable in category 2, 3, 4, or 5 is used in a kernel component and defined in another component. Therefore, a definition of a category-2, -3, -4, or -5 global variable induces the dependency of a kernel component on another component, either in the kernel or the nonkernel. More specifically, a definition of a category-2, -3, or -5 global variable in a kernel component or a category-4 or -5 global variable in a nonkernel component is a component-dependency-inducing definition. Table 5 lists the number of component-dependency-inducing definitions in the operating systems we consider here.

Table 5. Dependencies of a kernel component

Operating system	Total number of CDID	Number of CDID per kernel component	Number of CDID per kernel KLOC
Mach	140	1.97	4.58
FreeBSD	149	1.14	1.37
OpenBSD	165	2.04	2.95
NetBSD	168	1.98	2.60
Linux	1908	73.38	134.08

The entries in Table 5 may be interpreted as follows: Suppose we wish to reuse a kernel component K of Mach. On average, we will then have to modify 1.97 definitions of global variables in other components that induce dependencies in K and thereby affect its reuse. Similarly, if we wish to reuse 1,000 lines of Mach kernel code, on average we will need to need to modify 4.58 definitions of global variables in other components that induce dependencies and thereby affect the reuse of this code. (For the sake of comparison, we remark that if we wish to reuse a Linux kernel component, on average we will need to modify 73.38 definitions of global variable in other components; to reuse 1,000 lines of Linux kernel code, on average we will need to modify 134.08 definitions of global variables in other components.)

From Table 5, we can see that Mach, FreeBSD, OpenBSD, and NetBSD have a relatively small number of component-dependency-inducing definitions per kernel component and per kernel KLOC. This shows that, on average, a kernel component K in Mach, FreeBSD, OpenBSD, or NetBSD has few dependencies on other components, which makes K comparatively easy to reuse. The relatively independent property of a kernel component in Mach, FreeBSD, OpenBSD, or NetBSD means less effort is needed to reuse kernel components.

5.3 Dependencies of the Kernel as a Whole

In this paper, we are more concerned with entire-kernel reuse than kernel-component reuse. As we mentioned before, in most cases, successful reuse of kernel-based software depends on the reuse effort of the entire kernel. In the following, we discuss the reuse effort for the Mach, FreeBSD, OpenBSD, and NetBSD kernels. As stated in Section 3, a global variable is kernel-on-nonkernel-dependency-inducing if it induces a dependency of a kernel component on a nonkernel component.

As mentioned before, a category-4 or -5 global variable is the most undesirable. According to terminology 2, such a global variable is kernel-on-nonkernel-dependency-inducing, because it has a definition in a nonkernel component and a use in a kernel component. That is, the definition of a category-4 or -5 global variable induces a dependency of a kernel component on a nonkernel component; these dependencies adversely affect the entire-kernel reuse effort. Table 6 enumerates the kernel-on-nonkernel-dependency-inducing global variables in the open-source operating systems we consider here.

Table 6. Kernel-on-nonkernel-dependency-inducing global variables

Operating system	Number of global variables	Number of instances of uses in kernel	Number of instances of definitions in nonkernel
Mach	13	38	40
FreeBSD	10	48	46
OpenBSD	20	63	84
NetBSD	19	95	81
Linux	44	523	1667

Considering entire-kernel reuse, there are 13 global variables that make Mach kernel components dependent on nonkernel components. These 13 global variables are used 38 times in kernel components. Also, these 13 global variables are defined 40 times in nonkernel components. An implication of Table 6 is that, if we wish to reuse the entire Mach kernel, we either need to modify the 38 uses of kernel-on-nonkernel-dependency-inducing variables in kernel components to remove the dependencies, or we also need to incorporate 40 definitions in nonkernel components (or some combination of the two alternatives). Combining Table 6 with Table 4, we see that, although there are 332 instances of global variables in the Mach kernel, only 38 of them induce dependencies of a kernel component on a nonkernel component. Furthermore, of the 228 instances of global variables in nonkernel components, only 40 of them make it difficult to reuse the entire kernel. A similar result is found for FreeBSD, OpenBSD, and NetBSD.

Now we determine how many kernel components have to be changed, or how many nonkernel components have to be reused if we want to reuse the entire kernel.

Dependencies between components caused by global variables are induced by the definition–use relationship. As stated in Section 4, a kernel component is use-dependency-induced if it contains a use of a kernel-on-nonkernel-dependency-inducing variable, and a nonkernel component is definition-dependency-inducing if it contains a definition of a kernel-on-nonkernel-dependency-inducing variable. Use-dependency-induced kernel components use the value of a kernel-on-nonkernel-dependency-inducing variable; definition-dependency-inducing nonkernel components define the value of a kernel-on-nonkernel-dependency-inducing variable, which means that a use-dependency-induced kernel component is dependent on at least one definition-dependency-inducing nonkernel component. A kernel is difficult to reuse if it has too many use-dependency-induced kernel components and if there are too many definition-dependency-inducing nonkernel components.

Table 7 shows the number of *use-dependency-induced kernel components* (UDIKC) and *definition-dependency-inducing nonkernel components* (DDINC) in the operating systems we consider here. Multiple occurrences of the same component are ignored. For example, if kernel component K contains multiple uses of kernel-on-nonkernel-dependency-inducing global variables gv1 and gv2, it is nevertheless counted as only one use-dependency-induced kernel component, because modifications will have to be made to kernel component K irrespective of the number of uses of kernel-on-nonkernel-dependency-inducing global variables. Similarly, if nonkernel component NK contains multiple definitions of kernel-on-nonkernel-dependency-inducing global variables gv3 and gv4, it is likewise counted as only one definition-dependency-inducing nonkernel component.

Using Mach as an example to explain the entries of Table 7, 12 kernel components in Mach are use-dependency-induced kernel components. There are 19 definition-dependency-inducing nonkernel components. This means that 12 kernel components have dependencies on 19 nonkernel components via common coupling. More precisely, 12 kernel components use at least one kernel-on-nonkernel-dependency-inducing variable in a total of 38 instances, which depend on 19 nonkernel components that define a kernel-on-nonkernel-dependency-inducing variable in a total of 40 instances.

Now, suppose that all 77 Mach kernel components are to be reused. Two extreme approaches could be taken. First, we could modify the 12 use-dependency-induced kernel components in 38 places to remove the dependencies of kernel components on nonkernel components. Second, we could reuse the 19 definition-dependency-inducing nonkernel components together with the kernel. Clearly, any combination of these two extreme approaches could also be adopted. Turning now to reusing the FreeBSD kernel, we could similarly modify the 14 use-dependency-induced kernel components in 48 places, reuse the 25 definition-dependency-inducing nonkernel components together with the kernel, or adopt some combination of the two extreme approaches.

Table 7. Dependencies of kernel on nonkernel

Operating system	Kernel		Nonkernel	
	Number of UDIKC	Number of instances of uses	Number of DDINC	Number of instances of definitions
Mach	12	38	19	40
FreeBSD	14	48	25	46
OpenBSD	13	63	40	84
NetBSD	16	95	41	81
Linux	21	523	534	1667

Recapitulating, suppose we wish to reuse the entire Mach kernel, from Table 4, it appears that we would have to modify 332 instances of global variables in kernel modules. By considering only those instances that induce dependencies of a kernel component on a nonkernel component, we see from Table 6 that only 38 of the 332 instances would have to be changed. Finally, by considering use-dependency-induced kernel components, we see from Table 7 that the number of kernel components that would have to be changed is 12. Alternatively, 19 definition-dependency-inducing nonkernel components would have to be reused together with the entire kernel. This shows that the Mach kernel and the FreeBSD kernel are relatively independent as a whole, which means less effort is needed to perform entire-kernel reuse.

(In passing, we remark that it is hard to find a good strategy for reusing the 26 Linux kernel components. On one hand, if we modify the 21 use-dependency-induced kernel components in 523 places, we may completely change the functionality of the kernel. On the other hand, reusing the 534 definition-dependency-inducing nonkernel components together with the kernel would result in widespread unnecessary and redundant reuse. Furthermore, a kernel is generally difficult to reuse if it references a kernel-on-nonkernel-dependency-inducing variable gv and there are many definitions of gv in nonkernel components and many uses in kernel components. Linux has more instances of uses of kernel-on-nonkernel-dependency-inducing variables in kernel components and instances of definitions in nonkernel components than Mach or the three BSDs, which means that the Linux kernel is strongly dependent on nonkernel components.)

Software reuse depends on a large number of disparate factors [16]. One factor is the effort spent on customizing and reusing these components. The reuse effort of a kernel-based software product depends on the reuse effort of its kernel and this, in turn, depends on the definitions and uses of global variables within the kernel and nonkernel components. From the viewpoint of dependencies, reusing both the Mach and FreeBSD kernels is relatively effortless, irrespective of the precise reuse mechanism followed.

7. Conclusions and Threats to Validity

In this paper, we have utilized our categorization of common coupling based on definitions and uses of global variables to analyze the reuse effort for a software component. Common coupling in different categories has different effects on the reuse effort in kernel-based software. Our results show that common coupling within the Mach kernel and the BSD kernel is well designed, inducing only a few dependencies of kernel components on nonkernel components. As a result, relatively less effort is required for entire-kernel reuse of these two operating systems.

One threat to the validity of our study is using common coupling as a measure of reuse effort. Our result could certainly be strengthened if a validation test could be performed in which common coupling is used as the independent variable and reuse effort as the dependent variable. However, the goal of our study (as stated toward the end of Section 1) was to understand the relation between common coupling and software reuse effort by analyzing the structure of software components. This analysis was based on a well discussed and studied approach. Therefore, we believe our analysis is valid.

Our research was performed on five open-source software products. The approach can be used on other kernel-based software products to identify components that are easy to reuse from the viewpoint of dependencies. In addition, it can be used to suggest how future software should be written to reduce dependencies and thereby promote reuse.

8. References

[1] C.W. Krueger, "Software reuse," ACM Computing Surveys, vol. 24, no. 2, 1992, pp. 131-183.

[2] D.E. Perry, H.P. Siy, and L.G. Votta, "Parallel changes in large-scale software development: an observational case study," *ACM Transactions on Software Engineering and Methodology*, vol. 10, no. 3, 2001, pp. 308–337.

[3] F.P. Brooks, "No silver bullet: essence and accidents of software engineering," *IEEE Computer*, vol. 20, no. 4, 1987, pp. 10–19.

[4] K.J. Sullivan and J.C. Knight, "Experience assessing an architectural approach to large-scale systematic reuse," *Proceedings of the 18th International Conference on Software Engineering*, Berlin, March 1996, pp. 220–229.

[5] W.B. Frakes and S. Isoda, "Success factors of systematic reuse," *IEEE Software*, vol. 11, no. 5, 1994, pp. 14–19.

[6] W. P. Stevens, G. J. Myers, and L. L. Constantine, "Structured design," *IBM Systems J.,* vol. 13, no. 2, 1974, pp. 115–139.

[7] S. R. Schach, B. Jin, D. R. Wright, G. Z. Heller, and J. Offutt, "Quality impacts of clandestine common Coupling," *Software Quality Journal*, vol. 11, 2003, pp. 211–218.

[8] L. Yu, S.R. Schach, K. Chen, and J. Offutt, "Categorization of common coupling and its application to the maintainability of the Linux kernel," *IEEE Transactions on Software Engineering*, vol. 30, no. 10, 2004, pp. 694–706.

[9] Apple Computer, "Mac OS X hits stores this weekend," March 2001, www.apple.com/pr/library/2001/mar/-21osxstore.html.

[10] Mach Project, "Mach 3.0 sources," undated, www-2.cs.cmu.edu/afs/cs/project/mach/public/-www/sources/sources_top.html.

[11] FreeBSD Project, http://www.freebsd.org/

[12] Apple Computer, "Inside Mac OS X," May 2000, www.apple.com.pl/infotech/varia/macosx/InsideMacOS X-SystemOverview.pdf.

[13] J. West, "How open is open enough? modeling proprietary and open source platform strategies," *Research Policy*, vol. 32, no. 7, 2003, pp. 1259–1285.

[14] Kernelthread, "What is Mac OS X?" 2005, www.kernelthread.com/mac/osx/arch_xnu.html.

[15] L. Yu, S.R. Schach, K. Chen, G.Z. Heller, and J. Offutt, "Maintainability of the kernels of open-source operating systems: a comparison of Linux to FreeBSD, NetBSD, and OpenBSD," *Journal of System and Software,* vol. 79, no. 6, 2006, pp. 807-815.

[16] G. Caldiera and V.R. Basili, "Identifying and qualifying reusable software components," *IEEE Computer*, vol. 24, no. 2, 1991, pp. 61–70.

An Approach to Validating Translation Correctness From SAM to Java

Yujian Fu
School of Computing & Information Science
Florida International University
yfu002@cs.fiu.edu

Zhijiang Dong
Department of Computer Science
Middle Tennessee State University
zdong@mtsu.edu

Gonzalo Argote-Garcia
School of Computing & Information Science
Florida International University
gargo001@cis.fiu.edu

Leyuan Shi
Department of Industrial and Systems Engineering
University of Wisconsin at Madison
leyuan@engr.wisc.edu

Xudong He
School of Computing & Information Science
Florida International University
hex@cs.fiu.edu

Abstract

SAM is a formal software architecture description model based on Petri nets and temporal logic. SAM Parser is a tool to automatically translate a SAM architecture design into a program in ArchJava/Java/AspectJ with run-time verification capability. In this paper, we present an approach to show the correctness of the translation algorithm implemented in SAM Parser. Our approach requires a restricted Petri net model with an interleaving semantics, defines the operational semantics of a Java program using communication traces, and shows the consistency between an execution sequence of a Petri net and a communication trace of the corresponding Java program.

1 Introduction

There is a growing awareness, both in industry and academia, of the crucial role of formally proving the correctness of safety-critical portions of systems. Most verification methods focus on the verification of a design against requirements or an implementation with respect to a given design specification. However, the above verification task can be eliminated if we have a correct translation from a specification to an implementation. As addressed by Tony Hoare in the “Grand Challenge” [7] in 2003, verifying the correctness of translation is challenging because of the complexity and syntactic and semantic diversity of the target architectures, as well as the sophisticated analysis and optimization algorithms used in the process of translation.

Formally verifying a full-fledged general translation from a specification to an implementation is not possible, due to its size, variety of source and target languages, and, possibly, proprietary considerations. However it is feasible to correctly translate restricted formal specifications into programs in some suitable high-level programming language. In the SAM Parser [3], we implement an algorithm to automatically translate a given SAM architecture design description into a program in ArchJava/Java/AspectJ with runtime monitoring code. In this paper we provide an approach to prove the correctness of the translation.

A translation mapping X is correct means that it preserves the meaning of a specification (source) in the implementation (target) [9], which includes both *consistency* and *completeness*. Translation *consistency* refers to the dynamic behavior in the source is preserved in the target. There are no internal contradictions, whereas translation *completeness* requires that each entity in the source is correctly reflected and represented in the target.

Definition 1 (Translation Correctness) *Let X be a translation that maps a specification $sp \in \mathcal{S}$ (the set of specifications) to a program construct $prog \in \mathcal{J}$ (the set of programs), $X : \mathcal{S} \rightarrow \mathcal{J}$. The specification sp is a 3-tuple $< A, B, C >$, where A denotes the architecture structure, B denotes the behavior of the specification, and C denotes the constraints of a system. The correctness of translation X is defined by the following two conditions:*

1. *Consistency: $B \approx X(B)$ and $C \approx X(C)$ where $\approx$ denotes the semantic consistency relation to be defined later.*
2. *Completeness: $\forall a \in A . \exists a' \in \mathcal{J}.$ such that $X(a) = a'$, and*

$\forall b \in B.\exists b' \in \mathfrak{J}.$ *such that* $X(b) = b'$, *and* $\forall c \in A.\exists c' \in \mathfrak{J}.$ *such that* $X(c) = c'$.

Several rules for architecture structure translation for SAM model were discussed in the work [3], which validates the *completeness* as part of the translation correctness. This paper focuses on the *consistency* proof of the translation correctness for the architecture model (SAM).

The rest of this paper is organized as follows. Section 2 gives a brief introduction of SAM. Then the method for the translation validation is proposed in Section 3. After that, we discuss the restricted PrT net with its semantics in section 4 and Java program and its communication trace semantics in section 5. The sequential consistency between PrT net and it Java program is proved in section 6. Discussion is given in section 7. Finally, conclusion is in the section 8.

2 Preliminaries

SAM [15] is a general formal framework for specifying and analyzing software architecture with its dual formalism – Petri Nets and temporal logic.

Predicate Transition Nets Predicate Transition (PrT) nets [4] are a class of high level Petri nets. A PrT has a net structure: (P,T,F), where P is a set of places represented by circles, T is a set of transitions represented by rectangles and T is disjoint from P, and F is a relation between P and T represented by arcs. Each place is assigned a sort indicating what kind of tokens it can contain. The tokens in a place can be viewed as a multi-set over the sort. A marking of a PrT net is a function that assigns tokens to each place. A label is assigned to each arc to describe types and numbers of tokens that flow along this arc. Each transition has a boolean expression called guard, which specifies the relationship among arcs related with the transition. A transition is enabled if there is an assignment to all variables occurred in arcs related with the transition such that each incoming place contains the set of tokens specified by the label of the arc, and the guard of the transition is satisfied. An enabled transition is fired under an assignment by removing tokens from incoming places and adding tokens to outgoing places.

Linear Temporal Logic Temporal formulas are built from elementary formulas using logical connectives $\neg$ and $\wedge$ (and derived logical connective $\vee$, $\Rightarrow$, and $\Leftrightarrow$), universal quantifier $\forall$ (and derived existential quantifier $\exists$), and temporal operators always $\Box$, future $\Diamond$, until $\mathcal{U}$, and next $\bigcirc$ (next). A LTL formula, $\Box(p \rightarrow \Diamond q)$ means that the situation that p is true implies eventually q is true always exists.

SAM Formally, a SAM model consists of a set of compositions $C = \{C_1, C_2, ..., C_k\}$ and a hierarchical mapping function h relating compositions. Each composition C_i, representing an architecture of a sub-system at an abstract level, is a three-set tuple $< C_{m_i}, C_{n_i}, C_{s_i} >$ where C_{m_i} is a set of components, C_{n_i} a set of connectors and C_{s_i} a set of constraints. Components/connectors consist of two parts: a set of property specifications and a behavior model, denoted by S_{ij} and B_{ij} respectively for a composition C_i and a component/connector C_{m_ij}/C_{n_ij}. Property specifications S_{ij} and constraints of a composition are expressed in temporal logic, while behavior model B_{ij} is defined in Petri net model. The interface of a behavior model B_{ij} consists of a set of places, named ports, each of which has either no outgoing arcs or no incoming arcs. For details in the analysis and modeling of SAM please refer to [6].

3 The Validation Approach

The translation from SAM to java construct includes three parts – architecture structure translation, behavior model translation, and property translation. The target programming languages for the three parts are ArchJava (for SAM structure), Java (for SAM behavior), and AspectJ (for properties). The structure of the translation is shown in the Figure 1.

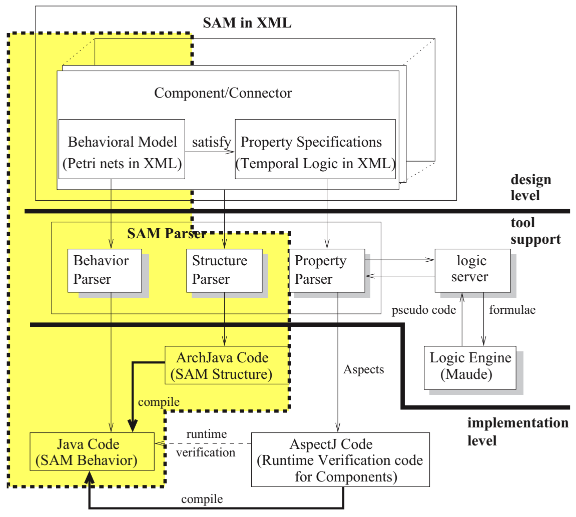


Figure 1. An Architecture of Translation

Both ArchJava and AspectJ code are compiled to java code, they are synthesized with the behavior Java code during execution. The translated properties are used to generate the runtime monitor to validate the system model. In this paper, we only consider how to validate the architecture structure and behavior model translation, which are highlighted in the Figure 1. Since ArchJava is compiled into Java code, assuming compiler is correct, we only consider the translation correctness of behavior Java program in SAM. Figure 2 shows the methodology used for translation validation.

The approach can be captured in the following steps (Figure 2). First, we impose some restrictions on the sorts and variables for the PrT nets. These restrictions facilitate the

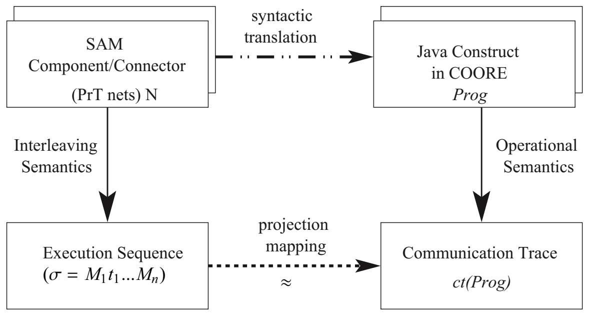

SAM Component/Connector (PrT nets) N
syntactic translation
Java Construct in COORE Prog
Interleaving Semantics
Operational Semantics
Execution Sequence ($\sigma = M_1 t_1 ... M_n$)
projection mapping
≈
Communication Trace ct(Prog)

Figure 2. The Methodology of Translation Validation

automatic Java code generation. Secondly, we restrict our java program for each component/connector to a sequential program and represent it with a sequential object oriented assertion language COORE [14]. Thirdly, we construct a mapping relation from the behavior model of SAM component/connector (a PrT net) to the syntax of COORE. Moreover, the communication trace of a Java program is defined on the given operational semantics of COORE. Finally, we show the sequential consistency between the state sequence of PrT net and the communication trace by projection calculation.

4 Restricted PrT Nets

In this section, we first give some restrictions for the PrT nets. One of the reasons is we can automatically generate the behavior code for the SAM model. After that, we explain the semantics used to express their behaviors.

The following restrictions are imposed on the PrT nets:

1. The sorts of PrT nets are either Java primitive types (denoted by D_{java}) such as *int*, *boolean* etc., or defined as a Java class (denoted by $D_{java-def}$). A product type D_{prod} can be defined by either primitive type or user defined type or mixed.

   $D ::= D_{java}|D_{java-def}|D_{prod}$ where $D_{prod} = D(\times D)^*$

2. Only the label of an incoming arc can define new variables.

3. The trap places only appear in the places that have *Set* sort.

4. Each Set sort is assumed to have finite number of elements. Thus the substitutions for each variable in the guard are finite since we model a closed system.

We use PrT nets to refer to the PrT nets with the above restrictions throughout this paper.

The dynamic semantics of PrT nets is defined by the transition firings. Although there are several well-known semantic models of Petri nets such as interleaving, interleaving set, causal in the literature, interleaving semantics, where the behavior is defined as a set of execution sequences and every execution step involves only one transition firing, is adequate to study safety and liveness properties of a system. We can define the interleaving semantics using the sequence of markings with the occurrence of corresponding transitions for each set of substitutions.

Definition 2 (Interleaving Semantics of PrT nets) *A sequence $\sigma = M_0[t_0/\alpha_0 > M_1[t_1/\alpha_1 > ...[t_{n-1}/\alpha_{n-1} > M_n$ with $n \geq 0$ is called a finite interleaving execution starting with M_0 iff $\forall i \in Nat$ and $0 \leq i \leq n$ and $M_{i-1} \rightarrow_{t_{i-1}/\alpha_{i-1}} M_i$, where $M_i : P \rightarrow \mathcal{P}(St)$, α_i denotes a substitution for the variables in a guard condition of a transition t_i, St denotes set of sorts.*

In the interleaving semantics of PrT nets, we examine each individual transition instead of set of transitions during the execution. Thus the preset and the postset of a transition provide a local view for the system execution. Let $\{p_{r1}, ..., p_{rm}\}$ be the preset of a transition t, $\{p_{o1}, ..., p_{on}\}$ be the postset of a transition t. The firing of transition t with binding α is represented as follows:

$F|_{t/\alpha} = \{p_{r1}(\alpha), ..., p_{rm}(\alpha)\}t\{p_{o1}(\alpha), ..., p_{on}(\alpha)\}$

This definition provides a possible local reasoning for the system in the interleaving semantics.

5 Semantics of Restricted Java Program

In this section, we first discuss how to restrict translated Java program from SAM component/connector to a sequential object oriented (OO) program, which means we can represent it using an assertion language COORE [14] – a simple sequential OO language. Then we define communication trace for translated Java program based on the operational semantics of COORE.

5.1 Restricting Translated Java Program

As shown in Figure 1, the translated code has two parts – architecture structure and behavioral model. Architecture structure includes components and connectors in SAM model, which are mapped into the component class in ArchJava code. Each component class implements a thread. The behavior model for each component/connector in SAM is a PrT net, which is translated to Java code. In this paper, we only consider the Java code translated from the behavior model of SAM – a PrT net.

The execution of the generated code of PrT net is a randomized algorithm, i.e., to run the translated PrT net we randomly choose an enabled transition and fire it. The execution stops when there is not enabled transition in this PrT net. Since in the runtime checking, we only examine one execution path, based on interleaving semantics, we can view this translation code for each component/connector of SAM model as a sequential OO program. Hence, we can use COORE [14]to describe this sequential OO program, we can use COORE [14] to define it. Due to the space limitation, readers please refer to the work [13] for details of COORE.

5.2 Communication Trace

An object's observable communication trace (proposed in the CSP [10] model to describe an execution path) can be defined by the history of all communication events between an object and its environment, which represents an abstract view of its state, readily available for reasoning about past and present behavior. We define a communication event for an OO program as follows.

Let *Prog* be an OO program, *Obj* be the set of object id (*Oid*), *Class* is the set of classes defined in the *Prog*, *Field*, *Meth* be the set of fields and methods respectively, S be the set of statements. We define a communication event for an OO program as follows.

Definition 3 (Communication Event) *A communication event of an object* $o \in Obj$ *is defined by a tuple* $< o_1, o_2, m >$ *such that* $o_1, o_2 \in Obj$ *and* $o_1 \neq o_2$*,* $m \in Meth$*.*

Either o_1 or o_2 can be null, in which case we abbreviate a communication event as $< o_i, m >$, where $i = [1, 2]$. Communication events are the ingredients of the communication trace, which is defined as follows.

Definition 4 (Communication Trace) *A communication trace of an OO program Prog, denoted by* $ct(Prog)$*, is composed of a sequence of communication events.*

We use lower case to represent elements in a set, e.g., a method $meth \in Meth$. We define the communication trace of translated Java program constructively in Table 1 based on the operational semantics of COORE [13].

In the Table 1, $\mathcal{E}[\![e]\!]$ be the evaluation on an expression e. Explanations of evaluation rules can be found in [13].

6 Establishing Behavioral Consistency

In this section, a mapping relation is established between elements in a PrT net and communication events of a translated Java program. We prove the semantic consistency between a PrT net execution sequence and the communication trace of its translated Java program.

Table 1. Communication Trace

$ct(Prog) = ct(class*)$

$ct(class) = ct(construct)(ct(field*);;ct(meth*))$,
where ;; denotes possible interleaving order among $meth_i$ and $field_i$

$ct(construct) = ct(o = (C, id))ct(e_i)ct(p_1, ..., p_n \mapsto o, v_1, ..., v_n)$

$ct(field*) = ct(field_1)...ct(field_n)$, where $n \in Nat$

$ct(meth*) = ct(meth_1)...ct(meth_n)$, where $n \in Nat$

$ct(meth(C, m)) = ct(o = (C, id))ct(e_i)ct(S)ct(e)$,
where $meth(C, m) \equiv t\ m(e_1, ..., e_n)\{S\ \ return\ e\}$

$ct(meth(C, m)) = ct(o = (C, id))ct(e_i)ct(S)$,
where $meth(C, m) \equiv void\ m(e_1, ..., e_n)\{S\}$

$ct(if(e)\ S_1 else\ S_2) = ct(e)(ct(S_1) \cup ct(S_2))$

$ct(while(e)\ S) = (ct(e)ct(S))*$

$ct(S_1; S_2) = ct(S_1)ct(S_2)$

$ct(u := e.m(\overrightarrow{e})) = \mathcal{E}[\![u := e.m(\overrightarrow{e})]\!]ct(e.m(\overrightarrow{e}))$

$ct(u := newC(\overrightarrow{e})) = \mathcal{E}[\![u := newC(\overrightarrow{e})]\!]$

$ct(u := e) = \mathcal{E}[\![u := e]\!]$

$ct(e) = \mathcal{E}[\![e]\!]$

6.1 A Mapping Relation

In reality, in the translated Java program, there are many internal methods, fields, assignments, expressions in the output communication trace. The communication trace for even one fired transition can contain a lot of information. Here, we only consider those methods, fields and necessary expressions that are directly related to a transition firing ($F|_{t/\alpha}$). The elements of a communication trace that are related to a PrT net in SAM are shown in the Table 2.

Table 2. Communication Events for PrT Elements in SAM Component/Connector

SAM	Java	Communication Event
substitution α_t	variable assignment	set of tokens tk
place $p_i(\alpha_t)$	Oid p_i p_i.contains(tk)	$< p_i, contains(\text{tk}) >$
transition t_i	Oid t_i t_i.fired() Oid g g.guardEvaluate(Vector pv)	$< t_i, fired() >$ $< g, guardEvaluate(Vector pv) >$

As discussed before, a firing rule for a transition includes the preset and the postset of the transition. If we fix an order on the places then after each firing of a transition, we can get the same sequence for the same transition under different markings. Since each input (output) place object is evaluated based on the index of the input (output) place vector [1], we can use the index of vector as the order for places.

From Table 2, we can define a mapping relation f that maps each element in the set of places P, set of transitions T, and tokens $CONs$ in a PrT net N to communication events in the communication trace. Let e be an element defined in the set $E = P \cup T \cup CONs$, we have $\forall e \in E, f(e) \in ct(Prog_N)$. For instance, for a component $User$ and a port $uInfoReq\_15$ specified in SAM model, we have

$f(\text{“}uInfoReq\_15\text{”}) = < uInfoReq\_15,$
$uInfoReq\_15.contains(token) > \in ct(Prog_{User})$.

6.2 Semantic Consistency

To reflect the firing for each transition in a PrT net, we need to extract the necessary information that is relevant to the firing of each transition, i.e., the preset and the postset of a transition as well as the transition itself. In the following, we first define the projection on a communication trace generated from a PrT net.

Definition 5 (Projection on a Communication Trace) *Given a PrT net $N = (Net, Spec, Insc)$ with net structure defined as $Net = (P, T, F)$, its communication trace is $ct(Prog_N)$. Let S be the set of elements defined the Java construct, The projection $Proj$ of communication trace $ct(Prog_N)$ on the set S, $Proj(ct(Prog_N), S)$, is the sequence arising from $ct(Prog_N)$ when all literals not contained in S are deleted.*

For the SAM example with a component $User$ and port id “$uInfor\_15$”, $Proj(ct(Prog_{User}), \{\text{“}uInfor\_15\text{”}\}) = ...\text{“}uInfor\_15\text{”}...\text{“}uInfor\_15\text{”}...)$.

Corollary 1 *Given a PrT net $N = (Net, Spec, Insc)$ with net structure defined as $Net = (P, T, F)$, its communication trace is $ct(Prog_N)$. Let $Prog_N|_{t/\alpha}$ be the set of preset and postset of transition t w.r.t. the Java construct, i.e.,*

$$\begin{aligned} Prog_N|_{t/\alpha} = \{ &< p_{r1}, p_{r1}.contains(tk_{r1}) >, ..., \\ &< p_{rm}, p_{rm}.contains(tk_{rm} >), \\ &< g, g.guardEvaluate(Vectorpv) > \\ &< t, t.fire() >, < p_{o1}, p_{o1}.contains(tk_{o1}) >, ..., \\ &< p_{on}, p_{on}.contains(tk_{on}) > \} \end{aligned}$$

The set of tokens tk_{ri} (where $1 \le i \le m$) in the Prog are defined by a substitution α (Table 2). $F|_{t/\alpha}$ be the firing of a transition t with α, f be the mapping function defined in the section 6.1. We have $Proj(ct(Prog_N), Prog_N|_{t/\alpha}) = f(F|_{t/\alpha})$.

Finally, we can lift each transition firing in the Java construct to the execution sequence for a PrT net in the SAM model.

Proposition 1 (Semantic Consistency $\approx$) *Given a PrT net $N = (Net, Spec, Insc)$ with net structure defined as $Net = (P, T, F)$, its communication trace is $ct(Prog_N)$.*

For each transition $t \in T$ of a restricted PrT net N, a firing for the transition t with substitution α is denoted as $F|_{t/\alpha}$, let $N|_t$ be the set of preset and postset of transition t as well as transition t, i.e., $N|_t = \{p_{r1}, ..., p_{rm}, p_{o1}, ..., p_{on}, t\}$, $\sigma = M_0[t_0/\alpha_0 > M_1[t_1/\alpha_1 > ...[t_{n-1}/\alpha_{n-1} > M_n$ be an execution sequence defined in Definition 2, then we have

$\forall 0 \le i \le n.\ Proj(\sigma, N|_{t_i/\alpha_i}) = F|_{t_i/\alpha_i}$.

We say a Java program is semantically consistent with its high level PrT net model if

$$\forall 0 \le i \le n.\ Proj(ct(Prog_N), Prog_N|_{t_i/\alpha_i}) \approx f(Proj(\sigma, N|_{t_i/\alpha_i})).$$

From the firing of a transition t and the mapping relation Table 2, we can get the communication sequence for $f(Proj(\sigma, N|_{t_i/\alpha_i}))$ that is composed of communication events in the last column of Table 2. Based on the Corollary, we can deduce the proposition.

7 Discussion and Conclusion

In our communication trace, we did not consider the object initiation and object completion, which were discussed in [8]. In the work [8], they aim at proving was to prove the correctness of an OO programming language COORE, while our work assume these OO features are true implicitly. Moreover, the verification method used in the work [8] is the asynchronous method call instead of synchronous ones, while in our translated Java program synchronous methods are used for each place that has token updated. In the work [5], the synchronization and concurrent mechanism of C/C++ were discussed and used for the same purpose, the translation rules from hierarchical Predicate Transition Nets to C/C++ were established. Many works have been done on the translation from high level Petri nets to different programming languages [11, 12], however, there is less work on the translation validation.

When we use interleaving semantics [2] to describe a concurrent system, we need to consider if any given concurrent behavior can be faithfully reproduced through an appropriate choice of a sequential interleaving. The answer is Yes, if we can simulate a concurrent execution by sequential nondeterministic interleavings at a sufficiently high level of granularity of the basic computational operations.

This paper presented an approach to validating the translation correctness of individual components/connectors modeled in PrT nets in a SAM architecture design to Java programs. The translation correctness is based on a restricted version of PrT nets and is established through definitions of the communication traces for Java program and a mapping relation between the execution sequences of a given PrT net and the communication traces of its target Java Program. Our future work includes the proof of translation correctness of the compositions of multiple components and connectors involving synchronizations.

Acknowledgement: This research was partially supported by NSF grants HRD-0317692 and NSF grant IIP-0534428 and Florida International University Graduated School Dissertation Year Fellowship.

References

[1] Java tutor. Available from http://java.sun.com.

[2] E. Best. *Semantics of sequential and parallel programs*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[3] Y. Fu, Z. Dong, and X. He. A Methodology of Automated Realization of a Software Architecture Design. In *Proceedings of The Seventeenth International Conference on Software Engineering and Knowledge Engineering (SEKE2005)*, 2005.

[4] H. J. Genrich. Predicate/Transition Nets. *Lecture Notes in Computer Science*, 254, 1987.

[5] X. He. Translating hierarchical predicate transition nets into cc++ programs. *Information and Software Technology*, 42(7):475–488, 2000.

[6] X. He and Y. Deng. A Framework for Specifying and Verifying Software Architecture Specifications in SAM. volume 45 of *The Computer Journal*, pages 111–128, 2002.

[7] T. Hoare. The verifying compiler: A grand challenge for computing research. *Journal of the ACM*, 50(1):63–69, 2003.

[8] E. B. Johnsen and O. Owe. An asynchronous communication model for distributed concurrent objects. In *Proceedings of the Software Engineering and Formal Methods, Second International Conference on (SEFM'04)*, pages 188–197, Washington, DC, USA, 2004. IEEE Computer Society.

[9] C. M. Lott. Correctness is congruent with quality. *Software Engineering Notes*, 15(5):19–20, October 1990.

[10] R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[11] K. H. Mortensen. Automatic code generation method based on coloured petri net models applied on an access control system. In Nielsen, M. and Simpson, D., editors, *Lecture Notes in Computer Science: 21st International Conference on Application and Theory of Petri Nets (ICATPN 2000), Aarhus, Denmark, June 2000*, volume 1825, pages 367–386. Springer-Verlag, 2000.

[12] P. A. Palanque, R. Bastide, and V. Sengés. Automatic Code Generation From a High-Level Petri Net Based Specification of Dialogue. In *EWHCI'94 (East-West Conference on Human Computer Interaction)*, St Petersburg (Russia), August 1994.

[13] C. Pierik. *Validation Techniques for Object-Oriented Proof Outlines*. PhD thesis, Computer Science Department, Utrecht University, May 2006.

[14] C. Pierik and F. S. de Boer. A proof outline logic for object-oriented programming. *Theoretical Computer Science*, 343(3):413–442, 2005.

[15] J. Wang, X. He, and Y. Deng. Introducing Software Architecture Specification and Analysis in SAM through an Example. *Information and Software Technology*, 41(7):451–467, 1999.

QSEE Project: An Experience in Outsourcing Software Development for Space Applications

Valdivino Santiago[1], Fátima Mattiello-Francisco[1], Ricardo Costa[2], Wendell Pereira da Silva[1] and Ana Maria Ambrósio[1]

[1]National Institute for Space Research, São José dos Campos, SP, Brazil
[2]DBA Engenharia de Sistemas, Rio de Janeiro, RJ, Brazil
valdivino@das.inpe.br, fatima@iss.inpe.br, rcosta@dba.com.br, wendell@das.inpe.br, ana@dss.inpe.br

Abstract

Nowadays, IT outsourcing is an increasing market in the world. This tendency can also be observed in space sector. Particular attention to software acquisition is given by the European Cooperation for Space Standardization (ECSS) providing volumes in software quality assurance and software engineering for space missions in order to make it easier the customer and supplier relationship. Facing the challenges of involving the Brazilian industry in the satellite mission's development as supplier, the National Institute for Space Research (INPE) has put effort in order to use the ECSS standards. This paper reports an INPE´s experience in outsourcing software development of a payload instrument on-board of a Brazilian scientific satellite mission using ECSS tailored form. The supplier is a Brazilian software company, CMMI-3 formally evaluated, which main business is development of non-critical software.

1. Introduction

The National Institute for Space Research (INPE) in Brazil has been involved in satellite development since 1979 with the creation of the *Missão Espacial Completa Brasileira* (MECB – Complete Brazilian Space Mission). According to MECB, INPE was responsible for building data collecting and remote sensing satellites besides the ground segment infrastructure for operating the satellites in orbit. Since the beginning of MECB two data collecting and two remote sensing satellites were developed and launched, the latter in cooperation with China Academy of Space Technology (CAST). Futhermore, INPE has a program to build small satellites which aims to provide the scientific community information related to phenomena of Earth equatorial atmosphere, high energy astrophysics and weather climate. These satellites are being conceived in the Departament of Atmospheric and Space Sciences (CEA) at INPE.

In Brazil, there is an increasing number of software companies for the IT market which are looking for capacitation and improvements on the quality of their products following software process improvement models, like CMMI and MPS.BR [1][2]. In 2005, according to SOFTEX, a non-governamental organization whose main goal is to increase Brazilian software companies competitivity and their participation into national and internartional market, 1,385 professionals attended courses related to the MPS.BR model, 50 companies implemented and 5 were evaluated according to such a model [3]. On the other hand, there is no evidence that any of these companies have expertise to develop software for satellite on-board computers. For the case of Brazilian satellites developed until now, the overall on-board computer software was mainly developed at INPE or jointly with CAST/China.

The research project *Qualidade do Software Embarcado em Aplicações Espaciais* (QSEE – Quality of Space Application Embedded Software) promoted INPE´s experience in outsourcing the development of satellite payload embedded software. The supplier is a Brazilian software company CMMI-3 formally evaluated, named *DBA Engenharia de Sistemas*. The QSEE project uses the European Cooperation for Space Standardization (ECSS) standards in order to guide the relationship between customer and supplier. This project is a first real attempt in order to enable a Brazilian software company to develop software for satellite payload computers and because that it plays a major role in industry software development for space domain in Brazil.

This paper is organized as follows. Section 2 details QSEE project's scope. Section 3 discusses how the ECSS standards were used as a basis for defining the interaction between INPE (customer) and DBA (supplier). Section 4 addresses the Software Independent Verification and Validation activities applied to the software product. Section 5 addresses the lessons learned while section 6 concludes the paper.

2. QSEE Project's Scope

The main goals of QSEE project are: i) to transfer to Brazilian software industry INPE´s knowledge in software development for space applications, particularly Verification and Validation tools, methods and techniques used for payload embedded software on-board of scientific satellites and balloon applications; ii) to update the software development methodology for scientific satellites and balloon payloads under development at CEA/INPE; iii) to create a methodology so that INPE can accept software developed by private software companies. So, QSEE was conceived to deal only with the development of satellite payload embedded software and it is not interested in hardware design and construction.

In order to reach these goals a satellite payload embedded software, named SWPDC, was specified by INPE. The *Monitor e Imageador de Raios X* (MIRAX – X-ray Monitor and Imager) satellite was the case study. MIRAX is a small X-ray astronomy satellite mission designed to monitor a large region around the central Galactic plane for transient phenomena and, due to its requirements, it will provide an unprecedented discovery-space spectral coverage to study X-ray variability in detail [4]. For QSEE, a simplified version of the computing subsystem architecture for MIRAX was chosen as shown in Figure 1.

In Figure 1, the On-Board Data Handling (OBDH) is the satellite platform computer in charge of processing platform and payload information and making the communication with the Ground Station. Furthemore, the payload is composed of two scientific instruments: Central Electronics Unit (CEU) and IONEX. IONEX indeed is a scientific experiment of another small Brazilian scientific satellite under development and it does not belong to MIRAX. Its purpose is only to address the multiplicity of instruments usually found in such systems. The CEU is the main instrument and it is composed of three computers: Payload Data Handling Computer (PDC) and two Event Pre-Processors (EPPs). The purpose of the EPPs is to accomplish a fast data processing of the X-ray cameras detectors signals and they are usually known as front-end processors.

The SWPDC is embedded into the PDC and its main responsibilities are to collect and format data from the EPPs, to receive and execute commands from the OBDH, to transmit telemetry data to the OBDH, to be able to generate housekeeping information of the CEU computing subsystem, to perform data memory management, to implement fault tolerance mechanisms and to support loading of new programs on the fly. The SWPDC has been developed in C/Assembly programming languages.

As hardware development is out of this project's scope, all computers in Figure 1, except PDC, are ordinary Intel-based PCs. The PDC is a development kit based on an improved version of the 8051/8032 microcontroller. Kit´s microcontroller runs in 40 MHz, has 288 kB internal flash memory, two UARTs, one 10-bit A/D converter among other features [5].

Next section describes the approach adopted in order to outsource the SWPDC development.

3. ECSS Tailoring

The absence of no uniform system of space standards and requirements in Europe resulted in higher costs, lower effectiveness and in a less competitive industry. So, the European space community realized that a solution had to be found to overcome these problems, and expressed their will to develop a new coherent system of European space standards which resulted in the ECSS standards [6].

The ECSS standards system has three branches, designated as Management, Product Assurance and Engineering. The Management standards define the process requirements to be applied to the overall project activities during the life cycle, the Product Assurance standards define the requirements for the management and performance of product assurance activities and the Engineering standards are devoted to the products themselves [7]. As there is a large amount of standards applicable to all products and projects, a selection and tailoring of ECSS standards are needed, at customer level, in order to meet the expectations of customers. For that, adaptation of ECSS standards shall be based on identified specific project objectives and constraints.

The ECSS standards selected and tailored to QSEE project were the Management ECSS-M-30A [8], ECSS-M-30-01A [9], the Product Assurance ECSS-Q-80B [10], and the Engineering ECSS-E-10-02A [11], ECSS-E-40 Part 1B [12] ones. The ECSS-M-30A defines the principles and requirements to be observed during the management of the project phasing and planning. It was mainly used to identify the relantionship between space project phases (0, A, B, C, D, E and F [13]) and the project reviews. Reviews held among customer and supplier representatives are generally conducted at the end of a phase aiming to verify if the objectives of the phase are met and the outputs of a phase have been produced in conformance to the specifications established in a previous phase. Identification and structure of all activities and information related to the project reviews, including the reviews bodies, are provided by the ECSS-M-30-01A. In QSEE it was mainly used to define a minimun set of representatives to attend the defined reviews.

The ECSS-E-10-02A establishes the requirements for the verification of a space system product. Based on this standard, for example, INPE defined to accomplish verification and inspection by two methods: test and inspection.

The ECSS-Q-80B defines a set of software product assurance requirements to be used for the development and maintenance of software for space systems.

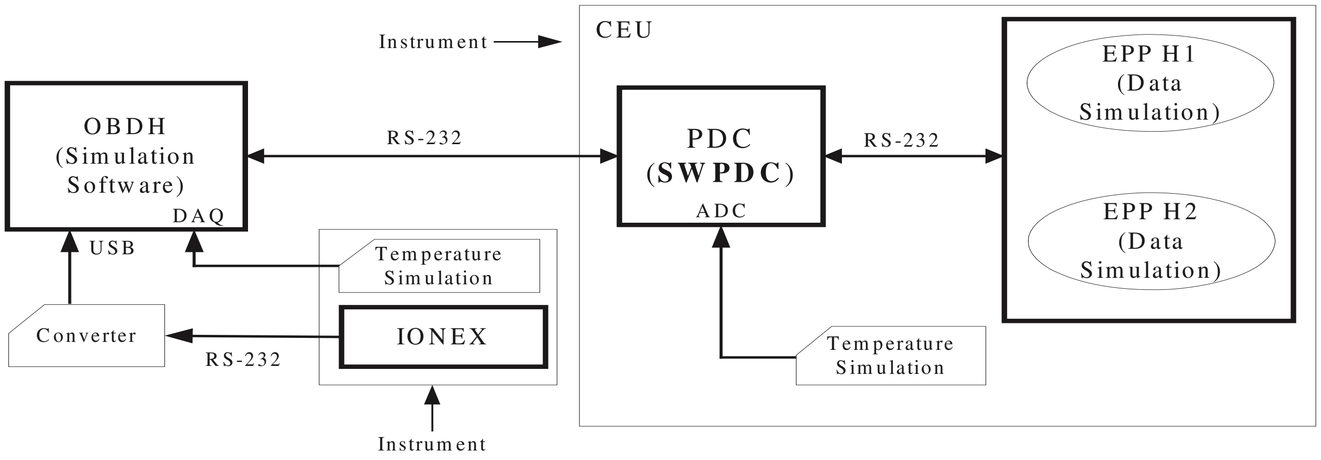


Figure 1 – Computing Subsytem Architecture for QSEE Project (Downsizing of MIRAX satellite). Legend: DAQ = Data Acquisition Board; USB = Universal Serial Bus; ADC = Analog-to-Digital Converter.

In QSEE, it was used to provide INPE proper confidence according to requirements related to the software product assurance programme implementation (e.g. Software Product Assurance Planning and Control, Software Product Assurance Reporting, ...) to the software process assurance (e.g. life cycle definition and reviews, milestones, ...) and to the software product quality assurance (e.g. Technical Specification, ...). By its turn, the ECSS-E-40 Part 1B covers all aspects of space software engineering including requirements definition, design, production, verification and validation, transfer, operations and maintenance. Due to its broadest software engineering aspect, the ECSS-E-40 Part 1B was the main basis of the tailoring in order to define the software life cycle processes, the deliverables which are input and output at each phase and so on. It is important to stress that these two standards are strictly devoted to the software component of the space application.

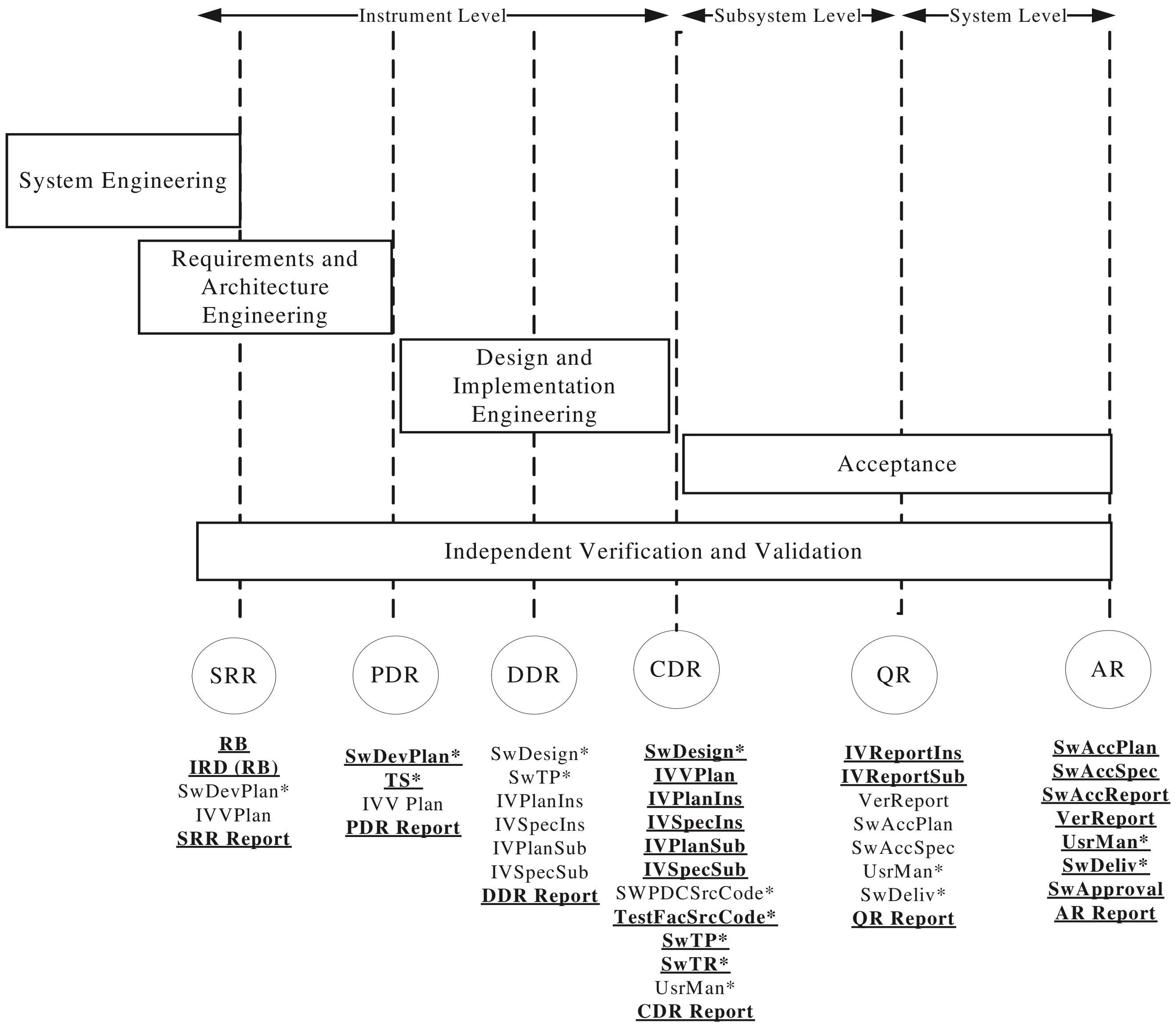
3.1 Sofware Life Cycle, Reviews and Deliverables

Reviews are the main interaction points between customer and supplier [12]. According to ECSS-E-40 Part 1B, the reviews relevant to the software engineering process are System Requirements Review (SRR), Preliminary Design Review (PDR), Detailed Design Review (DDR), Critical Design Review (CDR), Qualification Review (QR) and Acceptance Review (AR). Following this standard, these six were the ones established to exist between INPE and DBA. Figure 2 shows the software life cycle processes and their relationship with the software reviews.

DDR is a review specially recommended for flight software, at the end of the detailed design. It aims to review the detailed design, to review the software technical budget status (e.g. CPU and memory) and to review the completeness and stability of the TS (see legend in Figure 2) requirements, just in case of evolution of TS requirements after the PDR [12]. At first glance, QSEE project would not have such a review. However, during the SWPDC development, it became clear that it was necessary to exist a review where the detailed software design could be discussed and, because this is the first DBA project in the space domain, DDR was indispensable to the software life cycle. Although not shown in Figure 2, the TS was input to the DDR because there were evolution of requirements and it helped to better understand the SwDesign deliverable contents.

In Figure 2, beneath each review there is a set of deliverables evaluated during the review. For example, in the SRR, the RB, the IRD(RB), the SwDevPlan, the IVVPlan and the SRR Report are the deliverables which are input and/or output of such a review. Also, the deliverables in boldface and underlined should be in its final issue as the output of the review. For instance, the RB, which expresses the customer´s requirements, is input of the SRR and it must be modified according to the Review Item Discrepancies (RIDs) generated by SRR´s participants and its final issue should be within the SRR output. On the other hand, the IVVPlan is also input of the SRR but its final issue should be within the output of the CDR only. The deliverables marked with an asterisk (*) means they are supplier's responsibility; the others with no mark are customer's responsibility.

The ECSS-E-40 Part 1B recommends an input deliverable to be discussed in the PDR which addresses the architectural aspects of the software. In a rough analogy this deliverable, known as Design Definition File (DDF), is the equivalent to the Preliminary Design Document in the PDR and it should be developed apart from the TS. Also, the DDF is the primary input to the CDR review process. However, DBA TS deliverable has already addressed SWPDC architectural design aspects by means of UML Component diagrams and also with an equivalent of a static software architecture.



Legend:

RB = Requirements Baseline
IRD (RB) = Interface Requirements Document
SwDevPlan = Software Development Plan
IVVPlan = Independente Verification and Validation Plan
TS = Software Technical Specification
SwDesign = Software Design Document
IVPlanIns = Independent Validation Plan – Instrument Level
IVSpecIns = Independent Validation Test Specification–Instrument Level
IVPlanSub = Independent Validation Plan – Subsystem Level
IVSpecSub = Independent Validation Test Specification – Subsystem Level
SwTP = Software Test Plan
SwTR = Software Test Report

SWPDCSrcCode = SWPDC Source Code
TestFacSrcCode = Test Facilities Source Code
UsrMan = SWPDC User Manual
IVReportIns = Independent Validation Report – Instrument Level
IVReportIns = Independent Validation Report – Subsystem Level
VerReport = Verification Report
SwAccPlan = Software Acceptance Test Plan
SwAccSpec = Software Acceptance Test Specification
SwAccReport = Software Acceptance Test Report
SwDeliv = Software Delivery Document
SwApproval = Software Approval Document

Figure 2 – Software Life Cycle Processes, Reviews and Deliverables (* = Supplier's Responsibility Deliverable).

With such an approach, it became useless to produce a deliverable with these characteristics for the PDR. INPE accepted the DBA approach and the DDF equivalent was not evaluated in the PDR.

4. Software Independent Verification and Validation

One of the processes related to the software life cycle presented in Figure 2 was the Independent Verification and Validation (IVV). This process is mainly concerned to the delegation of the software acceptance activities to a third part team. The IVV processes are very suitable for mission critical software, like the ones found in space applications.

In QSEE project, the IVV activities have been developed by an independent team at INPE with cooperation of the University of Campinas (UNICAMP). The IVV process started with the construction of a verification matrix where are set up the verification methods (test and inspection) demanded by the customer in the different verification levels, i.e. the stages of software development and integration [13]. INPE has defined three verification levels: instrument, subsystem and system [11]. These levels can be seen in top of Figure 2.

In the instrument level, the embedded software is verified and validated according to the supplier´s Quality Assurance process. As shown in Figure 2, INPE has demanded two deliverables specific related to this process: Software Test Plan (SwTP) and Software Test Reports (SwTR). These deliverables should be in its final issue in the CDR, where the supplier, among other artifacts, delivers the software source code to the customer.

In the subsystem level, the customer is ready to apply all tests planned and specified until the CDR. As in a scientific satellite usually there are more than one instrument, the IVV team should elaborate not only an IVV Plan, where the activities to verify and validate the payload embedded software for all instruments are described, but also Independent Validation Plans (IVPlanIns) and Specifications (IVSpecIns) for each instrument presented in the satellite. These deliverables should take into account the validation of the instruments separately. For instance, in QSEE the IVV should build an IVPlanIns and an IVSpecIns for CEU and also for IONEX. After validating the instruments separately, the instuments should be integrated and another set of plans and specifications are needed: the IVPlanSub and IVSpecSub. In these two deliverables, the IVV team explains how the instruments integrated should be tested, which sequence of commands will be executed and so on. Notice that there also reports related to the different levels of testing: IVReportIns and IVReportSub.

Firstly, in the subsystem level, integration tests of all instruments embedded software are carried out in a test environment where the OBDH is simulated. After this stage, the payload (all instruments) should be integrated with the real platform computer (OBDH) and new tests should be applied. Performing integration tests of all instruments before the integration with the real OBDH and other satellite platform subsystems provides a great confidence that the payload softwares are according to their specifications and possible problems are more related to the integration aspects (e.g. cabling problems) of the instruments with the satellite platform than to the instuments themselves.

In the system level, tests should be carried out with the entire satellite and the ground segment. The IVV team should concern about the way tests will be executed, because test facilities and environment are different from previous levels. In order to do that, the IVV team should develop a Software Acceptance Test Plan (SwAccPlan), a Software Acceptance Test Specification (SwAccSpec) and Software Acceptance Test Reports (SwAccReport). It is important to stress that acceptance processes apply exactly in the subsystem and system levels.

The IVV team developed an acceptance strategy in order to generate test cases. This methodology, named CoFI [14], supports a tester to create a behavioural modelling of the system aiming at automatic test cases generation. The software behaviour under both, the normal conditions and under hardware faults, are modelled in Finite State Machines (FSMs) and test cases are generated by an automatic test case generation tool named Condado [15].

For test execution, a software tool has been developed to automate this activity of test process. The QSEE-TAS tool [16] main functionalities are test configuration and planning, test cases preparation and automatic execution, test project management and automatic test report generation in XML/XSL format. The QSEE-TAS enables a tester to create test cases manually and execute them automatically. As the reports are also automatically generated, the time spent to execute the test cases was significantly reduced

Analysing test reports is another activity of the test process. A process to cover this activity was created in QSEE as follows: (i) firstly, the test executor observes the test result in the QSEE-TAS; (ii) According to this result and relying on the RB, IRD (protocol specifications among OBDH, PDC and EPPs), TS, SwDesign, UsrMan and SWPDCSrcCode deliverables, the tester assigns a preliminary verdict. The QSEE-TAS enables five verdict values: Pass, Fail, Incloclusive, Error and Pass with Restriction. The first four values are according to the literature and Pass with Restriction was specifically created to situations where a software behaviour was desired but it is not implemented. However, the absence of this behaviour is not enough to establish a Fail verdict; (iii) Every week the IVV team and client representatives at INPE meet each other in order to evaluate the test reports. All test reports with a verdict different to Pass are reviewed; (iv) During this meeting a decision is made about the final verdict values of the test cases execution. A test case with a Fail verdict results in a Non-Conformance Record (NCR).

5. Lessons Learned

The first lesson learned is that the CMMI-3 maturity level of the DBA allowed its Software Quality Assurance team to identify very easily the needs required by INPE for the development of the instrument embedded software. Although DBA were not familiar with the software characteristics and development environment imposed by the customer, the CMMI-3 level showed confidence to the customer in terms of its solid software development structure based on well-established processes.

One interesting INPE´s point of view is closely related to the generation of NCRs and the IVV process. In an IVV approach, it is fundamental to understand that what is being verified and validated is not only the software source code but the entire software product. The reviews between customer and supplier are important in order to identify possible discrepancies in the deliverables elaborated and these are described in RIDs. Even though the customer can create many RIDs on supplier´s deliverables during the SRR, PDR, DDR and CDR, it is possible that some deliverables still have some undected problems. For example, during the execution of the tests specified by the IVV team, the software can exhibit a behaviour that is correct according to the IVV team analysis but one deliverable (e.g. TS) can be in disagreement with this behaviour. So, a NCR related to the TS deliverable can be generated. So, NCRs are not created only due to software source code non-conformances.

DBA company realized that it is vital to the supplier to understand, beforehand in the analysis phase, test steps and environment that will be applied to the software product during the acceptance processes accomplished by the customer. This understanding allows the resources and activities planning to be consistent with the project execution. Also, projects of satellite payload embedded software usually require the development of specific simulators and test environments in order to validate the software in the different physical models (e.g. Engineering Model, Flight Model, ...) [11][13].

The main advantage of outsourcing the development of satellite embedded software is to allow Brazilian software companies to become expert in a strategical area like space applications. Furthemore, such an approach may allow INPE to build more satellites in less time because it can delegate the development of satellite computer embedded software to other institutions. The main disadvantage of this process is the initial effort INPE should apply in order to allow these companies to develop software for this domain. In QSEE project, DBA software supplier attended training courses at INPE for 6 months in order to understand the characteristics, tools, processes related to the SWPDC development.

6. Conclusions

This paper presented an experience in outsourcing the development of satellite payload embedded software to a Brazilian software supplier. The QSEE project relied on the ECSS standards in order to guide the interaction between customer (INPE) and supplier (DBA). Because it is the first real experience in order to enable a Brazilian software company to develop software for satellite on-board computers, QSEE is an important project not only to transfer the concepts related to the development of space applications software to a supplier but also to enable INPE to define a methodology to accept software from companies in the future. Actually, the QSEE project is on the acceptance process where the tests planned and specified by the IVV team are being executed on the SWPDC delivered by the DBA in the CDR. The QR and AR reviews will happen in the last stage of the project.

7. References

[1] SOFTEX. MPS.BR – Melhoria de Processo do Software Brasileiro: Guia Geral. SOFTEX, 2006, p. 56, Versão 1.1 (In Portuguese).

[2] Chrissis, M.B.; Konrad, M.; Shrum, S. CMMI: Guidelines for Process Integration and Product Improvement. Boston: Addison Wesley Professional, The SEI Series in Software Engineering, 2007, p. 704, 2nd Edition.

[3] SOFTEX. Relatório Anual 2005. SOFTEX, 2006, p. 36 (In Portuguese).

[4] Braga, J. MIRAX Mision Overview and Status. In: The Transient Milky Way: A Perspective for MIRAX Conference, 2005, São José dos Campos-SP. AIP Conference Proceedings 840. Melville-NY: American Institute of Physics, 2006, p. 3-7.

[5] STMicroelectronics. uPSD33xx Turbo Series Data Sheet. STMicroelectronics, 2005.

[6] European Cooperation for Space Standardization (ECSS). Available at: <http://www.ecss.nl/>. Acessed on: 03 March 2007.

[7] European Cooperation for Space Standardization. ECSS-M-00A - Space Project Management: Policy and Principles. Noordwijk, The Netherlands: ESA/ECSS, 1996, p. 37.

[8] European Cooperation for Space Standardization. ECSS-M-30A - Space Project Management: Project Phasing and Planning. Noordwijk, The Netherlands: ESA/ECSS, 1996, p. 40.

[9] European Cooperation for Space Standardization. ECSS-M-30-01A - Space Project Management: Organization and Conduct of Reviews. Noordwijk, The Netherlands: ESA/ECSS, 1999, p. 38.

[10] European Cooperation for Space Standardization. ECSS-Q-80B - Space Product Assurance: Software Product Assurance. Noordwijk, The Netherlands: ESA/ECSS, 2003, p. 70.

[11] European Cooperation for Space Standardization. ECSS-E-10-02A - Space Engineering: Verification. Noordwijk, The Netherlands: ESA/ECSS, 1998, p. 144.

[12] European Cooperation for Space Standardization. ECSS-E-40 Part 1B - Space Engineering: Software – Part 1: Principles and Requirements. Noordwijk, The Netherlands: ESA/ECSS, 2003, p. 112.

[13] Mattiello-Francisco, M.F.; Santiago, V.; Costa, R.; Jogaib, L. Verificação e Validação na Terceirização de Software Embarcado em Aplicações Espaciais. In: V Simpósio Brasileiro de Qualidade de Software, 2006, Vila Velha-ES (In Portuguese).

[14] Ambrosio, A.M.; Martins E.; Mattiello-Franscisco, M.F. Vijaykumar N. L.; Santiago, V.; Carvalho, S.V. A Methodology for Designing Fault Injection Experiments as an Addition to Communication Systems Conformance Testing. In: Workshop on Dependable Software - Tools and Methods (DSN), 2005, Yokohama, Japan.

[15] Martins, E.; Sabião, S.B.; Ambrosio, A. M. ConData: a tool for automating specification-based test case generation for communication systems. Software Quality Journal, 1999, v.8, n. 4, p. 303-319.

[16] Silva, W.P.; Santiago, V.; Mattiello-Francisco, M.F.; Passos, D. QSEE-TAS: Uma Ferramenta para Execução e Relato Automatizados de Testes de Software para Aplicações Espaciais. In: XX Simpósio Brasileiro de Engenharia de Software - Sessão de Ferramentas, 2006, Florianópolis-SC. Porto Alegre-RS: SBC, 2006, p. 43-48 (In Portuguese).

Broadening the Use of Process Patterns for Modeling Processes

Hanh Nhi Tran, Bernard Coulette
GRIMM-IRIT, University of Toulouse 2,
5 allées A. Machado, 31058 Toulouse, France
{tran,coulette}@univ-tlse2.fr

Bich Thuy Dong
University of Natural Sciences,
227 NguyenVanCu, Q5, HoChiMinh, Vietnam
thuy@hcmuns.edu.vn

Abstract

Generally, process patterns are considered as patterns capturing reusable development activities, and serve as building blocks for constructing new processes. However, such a definition is not adequate to represent the original idea of process patterns that aims to capture and reuse diverse process knowledge. In this work, we broaden the definition and application of process patterns to take more advantage of them for different process modeling needs. First, we formalized the process pattern concept so that it can capture various kinds of process knowledge. Then, we proposed different ways for reusing process patterns to generate or improve process models. These propositions were rigorously defined in a UML-based meta-model to permit describing processes based on process patterns with standard notations, and to facilitate the development of supporting tools.

1. Introduction

Software Process Modeling deals with producing explicit, formal representations of software processes to help understand, analyze, enact and improve software development. Because software processes are intrinsically complex, process modeling is one of the most fundamental challenges of Software Process Technology [2]. Reusing valuable process knowledge gained through the modeling of software processes is thus important to reduce process modeling effort and propagate process best-practices.

Inspired by the success of software product patterns, the process community proposed the concept of process pattern to capture and reuse explicitly process knowledge. Unfortunately, this attractive concept has still been poorly exploited due to the limited definition, the inadequate formalization and the lack of supporting methodology.

Generally, process patterns are defined as patterns describing proven processes for realizing development tasks (e.g., the *Technical-Review* pattern in [1] describes how to organize and conduct the review of one or more deliverables). Defined in this way, process patterns are only used as building blocks for constructing new processes. However, process knowledge of interest includes not only such software development tactics, but also other heuristics of process modeling, organizing and execution (e.g. the patterns proposed in [10] define pragmatic and abstract building blocks for modeling recurrent situations in collaborative works). This kind of process knowledge can be useful for both process construction and process improvement, but it is currently ignored in process pattern formalization and application.

We argue that the existing definitions of process patterns are not enough general to cover the diversity of process knowledge. Based on these inadequate definitions, the ways that process patterns are formalized and used are limited too. We think that broadening the view on process patterns can help us taking more advantage of process patterns for different process modeling needs.

Thus, we present in this paper a meta-model that enables the representation of diverse process patterns and their applications for elaborating process models based on process patterns.

The second section gives a brief survey of some significant works on process patterns and contrasts them with our approach. The third section presents how our process pattern meta-model supports capturing and representing various process knowledge. The fourth section describes how process patterns can be used to generate or improve process models. The conclusion sums up our contributions and future works.

2. State-of-art

In this section we want to summarize what kinds of process knowledge are generally captured in published process patterns and how those process patterns are formalized and reused by the existing works for facilitate process modeling.

We classified process knowledge into three principal types according to the level of abstraction. (a) generic process structures that are applicable for any process (e.g., the templates for modeling different process workflows[20]); (b) general development methods or techniques that can be applied for different application domains (e.g., the inspection technique for verify software artifacts[3]); (c) concrete hands-on experiences that are relevant to a particular process for producing a specific kind of product (such as a process for designing information systems adopted by a specific team).

Most of process-related patterns in literature capture the process knowledge type (b) [1,3,4,6,9] or (a) [10,15,16,20]. Normally, the knowledge type (c) is captured in internal process patterns of an organization.

There are few works on process patterns formalizing, we describe here some of those that define a meta-model for process pattern.

In the Living Process Framework [7], Gnatz et al. introduced the process pattern notion as a modular way to document development knowledge. However, they did not define explicitly the structure of a process pattern and interrelationships of patterns. In contrast, to facilitate patterns organization, Hagen et al. developed PROPEL [8], a UML-based language that describes explicitly the internal process pattern structure as well as relationships between patterns. The language PROMENADE [5] proposed by Ribo et al. defines process pattern as a process template and provides a set of operators for supporting process reuse and harvesting mechanisms. The draft submission of SPEM2.0 [12] describe process patterns as reusable clusters of activities in common process areas. Pattern applications are supported through the Activity Use mechanism, which defines relationship kinds to reflect automatically the changes of a pattern in all processes that applied that pattern. SPEM also ignores the internal pattern structure relationships between patterns. All the above works adopted a rather limited definition for process pattern concept, i.e. they focused mainly on capturing process knowledge type (b) and (c). They proposed a unique way to reuse process patterns as building blocks for generating new development processes.

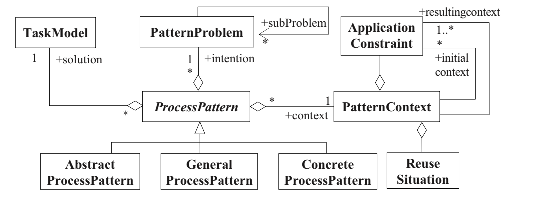
In contrast to these works, we define process patterns as patterns for modeling processes [19]. Thus, the process patterns in our definition can capture diverse process knowledge in all (a), (b) and (c) (c.f. [18] for a process patterns classification). We want to provide different mechanisms to apply process patterns not only for elaborating new process models, but also for redesigning or improving existing ones.

3. Process Patterns Formalization

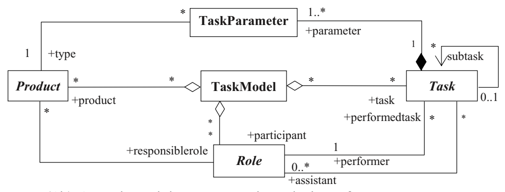
To enable a practical use of process pattern in process modeling, we formalize this concept by introducing it into a process meta-model. In order to meet the standardization and facilitate the development of supporting tools, our meta-model is defined as a MOF-conformed [14] meta model inspired by SPEM1.1[11] and reusing the UML 2 Infrastructure library [13].

In our meta-model, a process pattern (*ProcessPattern*) captures a model (*TaskModel*) representing a solution that can be applied in a given context (*PatternContext*) to resolve a recurrent process modeling problem (*PatternProblem*). (Figure 1a).

The *TaskModel* concept is used to describe a (fragment of) process. It is an aggregation of several process elements, i.e. *Task*, *Product* and *Role* (Figure 1b). A *Task* is a unit of controlled work (i.e. scheduled) realized by a *Role* to create or modify *Products*.



(1a) Meta-model for process patterns



(1b) A TaskModel represents the solution of a process pattern

Figure 1 Meta-model of process pattern concept

Process modeling problems can vary from describing specific development tasks to suggesting efficient generic process structures. Consequently, solutions provided by process patterns could describe diverse process knowledge at different levels of abstraction.

We distinguish three kinds of process patterns: *abstract*, *general* and *concrete* to capture respectively three process knowledge type (a)(b)(c) mentioned in section 1.

An *AbstractProcessPattern* captures a generic recurrent structure for modeling or organizing processes. In such a pattern, the precise semantic of elements is not important but the relations between them express the solution. For instance, in [10] Lonchamp described the pattern *"Division of labour"* as a solution for modeling collaborative tasks performed by several actors on subparts of a document to build the whole document in a parallel way (Figure 3a). This common structure can be applied to model any collaborative work without considering the semantics of its tasks.

A *GeneralProcessPattern* provides a general development method that is partially specified and must be refined further to be applicable for a specific purpose. For example, Figure 3b shows the *Fagan inspection process* [3] for detecting defects of software artifacts. In this process, the precise actions of the tasks *Preparation* (tester inspects individually the artifact) and *Rework* (author modifies artifact) must be specialized when applying for a specific kind of products (e.g. requirements, design, code, etc.); therefore we use a general pattern to describe it.

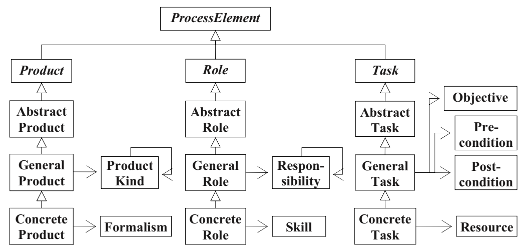
Finally, a *ConcreteProcessPattern* captures a completely specified solution of a process in a particular context. For example, Figure 3c describes a very specific process adopted at the *Vietnamese University of Natural Sciences* (NSU) for designing information systems. This process is used for development teams having one developer and one tester. The tasks of this process are described with concrete steps and manipulated products defined by the development approach and technique used at NSU (e.g. RUP-based process). Thus, we use a concrete pattern to capture this particular process.

To support representing process patterns at the above levels of abstraction, we also describe process elements at three abstraction levels: *abstract, general* and *concrete*. An *abstract process element* is used to refer to an unspecified element without defining its precise semantic (e.g. a task *T*, a product *P*). A *general process element* is partly defined but could be specified further (e.g. a *code product* can be object-oriented or functional with different characteristics; a *review task* can work on code or design document with different details, techniques). A *concrete process element* is completely specified for a specific purpose (e.g. a *UMLDesignReview* task).

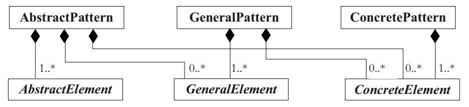
This hierarchy is applied for all types of process elements (Figure 2a), i.e. an *AbstractElement* can be *AbstractTask*, *AbstractProduct* or *AbstractRole*, etc. An abstract process pattern must contain at least an abstract element. A general process pattern has no abstract element but has at least a general element. A concrete process pattern contains only concrete elements.

The possible compositions of process patterns on different levels of abstraction are shown in Figure 2b.

Figure 3 shows an example of different process patterns in our categorization.

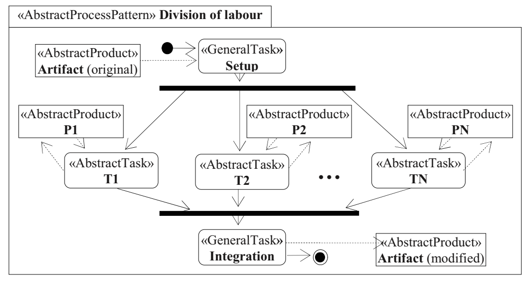


(2a) Definition of process elements on different levels of abstraction



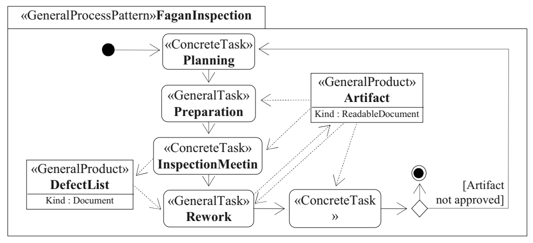
(2b) Compositions of process patterns on different abstraction

Figure 2 Abstraction levels of Process Elements



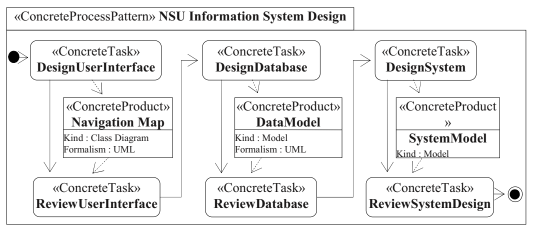
«AbstractProcessPattern» Division of labour
«AbstractProduct» Artifact (original)
«GeneralTask» Setup
«AbstractProduct» P1
«AbstractProduct» P2
«AbstractProduct» PN
«AbstractTask» T1
«AbstractTask» T2
...
«AbstractTask» TN
«GeneralTask» Integration
«AbstractProduct» Artifact (modified)

(3a) A structure enabling parallel collaborative works (adapted from [10])



«GeneralProcessPattern»FaganInspection
«ConcreteTask» Planning
«GeneralTask» Preparation
«GeneralProduct» Artifact
Kind : ReadableDocument
«ConcreteTask» InspectionMeetin
«GeneralProduct» DefectList
Kind : Document
«GeneralTask» Rework
«ConcreteTask »
[Artifact not approved]

(3b) Fagan method for inspect software artifacts (adapted from [3])



«ConcreteProcessPattern» NSU Information System Design
«ConcreteTask» DesignUserInterface
«ConcreteTask» DesignDatabase
«ConcreteTask» DesignSystem
«ConcreteProduct» Navigation Map
Kind : Class Diagram
Formalism : UML
«ConcreteProduct» DataModel
Kind : Model
Formalism : UML
«ConcreteProduct » SystemModel
Kind : Model
«ConcreteTask» ReviewUserInterface
«ConcreteTask» ReviewDatabase
«ConcreteTask» ReviewSystemDesign

(3c) The Design Process of NSU team having 1 designer and 1 tester

Figure 3 Example of three kinds of process patterns

3. Applications of Process Patterns

After examining many issues in process modeling, we distinguish two types of problems: *defining a process element* and *organizing process elements*.

The first problem concerns the question of *how to define the details of a process element* (including development tasks, work products or participant roles). A pattern addressing such a question captures a model describing the content of the required process element. For example, the process pattern in Figure 3b can be used to define a *technical review task* of a development process. This type of problems is ordinary but frequent.

The question for the second problem is *how to organize a group of process elements* to satisfy a special situation or certain constraints. Such problems focus on how to achieve high quality process models and effective executions of those models. A pattern dealing with this type of problems provides a model for structuring a (fragment of) process, i.e. for establishing certain relationships between process elements. The pattern shown in Figure 2a, for example, provides a process flow that takes into account late information by allowing cloning process steps. Although this type of problems is currently overlooked in process pattern formalization, it do exist and was discussed in many works [15, 16, 20].

During process modeling, there are several scenarios where process designers may want to reuse process patterns. Based on the two above types of process modeling problems, we identified two main uses of process patterns as follows.

3.1 Using Process Patterns to generate the content of a process element

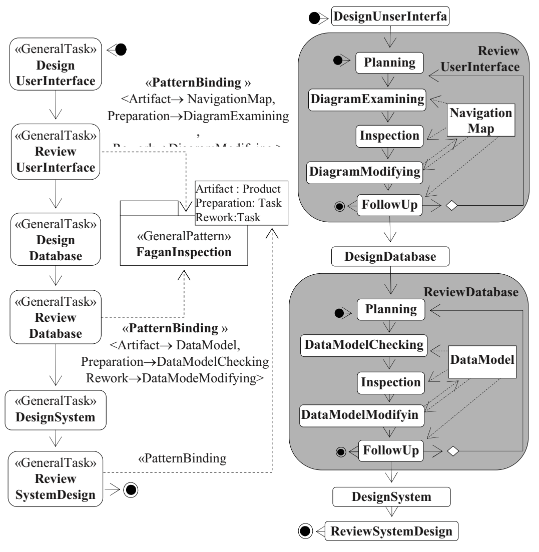
Often, in the process definition phase, process designers need to elaborate a detail description for a process element. In such cases, process patterns can be used as building blocks to construct new processes quickly. To our knowledge, this is the most popular and unique use of process patterns proposed by the process community for process modeling.

In our meta-model, this kind of process pattern reuse is represented by the relation *PatternBinding* between a process element (source) and a process pattern (target). The presence of a *PatternBinding* relation implies that the contents of the model captured in the target process pattern will be copied into the bound process element.

We realize process patterns reuse by the abstraction mechanism. More precisely, we define process patterns as parameterized templates [17] and allow explicit parameter substitutions in the relation *PatternBinding* to refine process models captured in process patterns on instantiating them[1].

We show in Figure 4 an example illustrating the use of process patterns to generate process elements' contents through the relation *PatternBinding*. For the sake of simplicity, we will not represent the products in this example.

Figure 4a shows the baseline process model of NSU containing unspecified process elements. When refining this baseline process, the process designer decided to apply the well-known Fagan process to review software artifacts. Therefore, he reused the process pattern *FaganInspectionPattern* (Figure 3b) to generate the content of the tasks *ReviewUserInterface*, *ReviewDatabase* and *ReviewSystemDesign*. *FaganInspection* is defined as a general pattern with three parameters corresponding to the general elements to be specialized: the product *Artifact* to be inspected, the tasks *Preparation* and *Rework*[2] for detecting defects and modifying artifact. When binding the pattern to generate the content of a specific process element, these parameters were substituted by appropriate actual parameters. For example, the parameter *Preparation* was substituted by *DiagramExamining* for the bound task *ReviewUserInterface*; and by *DataModelChecking* in the bound task *ReviewDatabase*. The unfolded models of the tasks *ReviewUserInterface* and *ReviewDatabase* are shown in Figure 4b.



(4a) NSU Baseline Process Model with unspecified process elements

(4b) Resulting model of the binding relation from *ReviewUserInterface and ReviewDatabase* to *FaganInspection* pattern

Figure 4 Example of process pattern application for generating a process element's content

[1] Due to the space limit, we cannot present here the detailed definition of this relation.

[2] In this pattern, there are the concrete tasks (*Planning*, *InspectionMeeting* and *FollowUp*) that remain the same for any review task. Thus, they are not exposed as parameters. *DefectList* is ignored to simplify the example.

3.2 Using Process Patterns to (re)organize process elements

For process elements organizing problems, another way to apply process patterns is required. The general situation in such cases is that process designers want to apply new relations on a group of existing process elements, or to add new information to that group. Process patterns for such problems can be used as process structure templates to (re)organize processes. However, this kind of process pattern reuse has not been exploited yet in practical process modeling. We define therefore the relation *PatternApplying* to enable reusing process structure templates.

A *PatternApplying* relation represents the application of a source pattern to a specific situation involving specific target process elements playing the roles of the elements defined in the pattern. As for the relation *PatternBinding*, we allow parameter substitutions in this relation to promote the use of process patterns on different levels of abstraction[3].

The presence of a *PatternApplying* relation implies that the structure of the process model captured in a source pattern will be applied to the target model. The specific way this application is realized depends on the *applyingMode* specified in the relation. We define three *applyingModes* for the relation *PatternApplying*: *change*, *replace* and *extend*.

If the mode c*hange* is chosen, all relations (if existed) between target model's elements will be removed, only new relations and dependencies (together with associated elements) defined among the elements of the source pattern will be applied to the corresponding elements in the target model.

If the mode *replace* is chosen, the content of the source pattern will override the existing one of the target model. More precisely, in the case the target model contains elements and relations other than the source pattern, these existing elements will be conserved in the resulting model only if they are not contrary to the ones defined in the source pattern, otherwise they will be replaced by the pattern's elements.

The mode *extend* indicates that the content of the source pattern will be merged with the existing one of the target model. However, the existing elements are intact and the new ones will be ignored if there are conflicts [4] between them.

[3] The extraction of our meta-model that defines these relations will be presented in another work.

[4] Conflicts may come from contrary dependencies between tasks' execution orders, products' impacts; or from the incoherent relations between tasks and manipulated products, participants roles, etc. The detailed analysis for such conflicts will be present in another work.

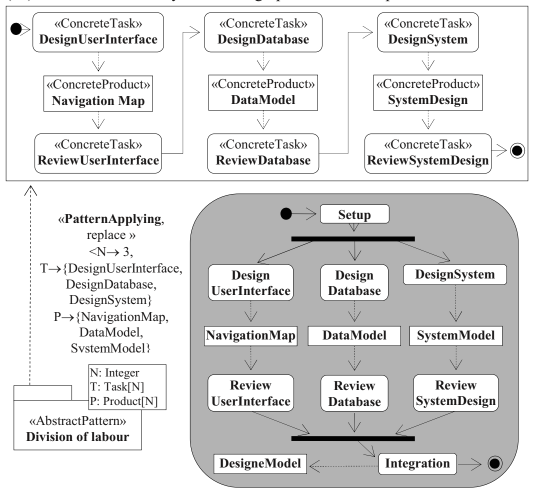
To illustrate the use of process patterns to reorganize the structure of process models, we take a scenario at a NSU design team.

To design an information system, normally this team uses the concrete process described in Figure 3c. This process was constructed for NSU small teams composed of one designer and one tester. Thus, the three components of design model, i.e. interface, database and system designs are elaborated and verified sequentially (Figure 5a). However, for a new large project, this NSU team grew from one to three for each role; it became a team having three designer and three testers. Thus, to optimize the development time, the project leader wanted to change the design process to allow parallel works.

To satisfy this requirement, the process designer applied the process pattern *Division of labour* (Figure 3a) to reorganize the workflow of *NSU Information Systems Design process* when keeping the fully specified elements of the original process.

Figure 5 shows such an application using *PatternApplying* relation.

(5a) NSU Information Systems Design process with sequential workflow



«ConcreteTask» DesignUserInterface
«ConcreteProduct» Navigation Map
«ConcreteTask» ReviewUserInterface
«ConcreteTask» DesignDatabase
«ConcreteProduct» DataModel
«ConcreteTask» ReviewDatabase
«ConcreteTask» DesignSystem
«ConcreteProduct» SystemDesign
«ConcreteTask» ReviewSystemDesign
«PatternApplying, replace »
<N→ 3,
T→{DesignUserInterface, DesignDatabase, DesignSystem}
P→{NavigationMap, DataModel, SystemModel}
N: Integer
T: Task[N]
P: Product[N]
«AbstractPattern» Division of labour
Setup
Design UserInterface
Design Database
DesignSystem
NavigationMap
DataModel
SystemModel
Review UserInterface
Review Database
Review SystemDesign
DesigneModel
Integration

(5b) Resulting model of the applying relation from *Division of labour* pattern to *NSU design process*. The modified process has the concurrent workflow.

Figure 5 An example of process pattern application for reorganizing process elements

Division of labour is defined as an abstract process pattern having three formal parameters: *N*-number of parallel tasks, *T* - list of tasks to be executed concurrently, *P*- list of products manipulated respectively by the tasks in the list *T*.

The *Division of labour* pattern was applied to these existing elements of *NSU DesginProcess*: *DesignUserInterface, DesignDatabase* and *DesignSystem*.

These elements played the pattern roles *T1*, *T2*, and *T3,* respectively. The actual parameter *NavigationMap*, *DataModel* and *SystemModel* substituted respectively *P1*, *P2* and *P3*.

Because the chosen applying mode was *replace*, the new elements *Setup* and *Integration* were added to the process; the new control flows were established between these new elements and the existing *DesignUserInterface,* *DesignDatabase*, and *DesignSystem* as described in the source pattern. The old control flows between *ReviewUserInterface* and *DesginDatabase*; between *ReviewDatabase* and *DesignSytem* were deleted because they did not conform to the new relationships established among the tasks *DesignUserInterface, DesignDatabase*, and *DesignSystem*.

The modified model of the *NSU Design Process* is shown in Figure 5b.

4. Conclusion

We have presented in this paper a solution to broaden the definition and the use of process pattern for exploiting better this concept in process modeling. This proposition is formalized in a process meta-model that defines rigorously the process pattern concept and two operators for process patterns reuse[5].

Compared with related works, our approach emphasizes process patterns reuse by abstraction mechanism. By distinguishing process elements at different abstraction levels, our meta-model allows representing various kinds of process knowledge encapsulated in process models. Based on this multi-levels abstraction, our process pattern definition covers a large variety of process patterns. By using parameterization to define process pattern concept and its reuse relations, we promote the use of process patterns and enable the pattern-based process model representation.

Especially, the two proposed applications of process patterns with the relations *binding* and *applying* provide complementary ways to exploit process patterns for different process modeling needs, which were ignored in previous works.

However, the proposed formalism just allows representing the process knowledge that can be expressed as a set of process elements with relations among them. Further studies on potential conflicts in applying process pattern(s) to a target model should be investigated to enable a more rigorous semantics of this operator.

[5] Due to space limit, we have not presented the OCL rules defining the semantic of the proposed concepts in our meta-model.

We are now attempting to formalize the actions to unfold the resulting process models of the proposed operators with OCL queries. Besides, a meta-process defining a systematic method for modeling processes based on patterns is currently developing.

To validate our work, we are implementing the proposed meta-model as a UML profile by using the *Objecteering ProfileBuilder*. The new profile will be imported into *Objecteering UMLModeler* to allow modeling processes based on process patterns.

In the future, we will develop a process modeling tool that supports the proposed meta-model and meta-process for constructing and tailoring processes by reuse process patterns.

5. References

1. Ambler, S. W., *Process Patterns: Building Large-Scale Systems Using Object Technology*, Newyork: SIGS Books/Cambridge University Press, 1998.
2. Derniame, JC., Kaba, BA., Graham Wastell, D. (Eds.), *Software Process: Principles, Methodology Technology*, LNCS1500, Springer-Verlag, 1999.
3. Fagan, M.E., “Advances in Software Inspections”. *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 7, Page 744-751, 1986.
4. Firesmith, D., Henderson-Sellers, B., *The OPEN Process Framework. An Introduction*. Addison-Wesley, 2001.
5. Franch, X., Ribó, J.M. “A UML-Based Approach to Enhance Reuse within Process Technology”, *EWSPT03,* 2003.
6. Gnatz, M., et al., *Proceedings of the 1st Workshop on Software Development Process Patterns*, *OOPSLA02*, Washington, 2002.
7. Gnatz, M. Marschall, F., Popp, G., Rausch, A., Schwerin, W. “The Living Software Development Process”. *Journal Software Quality Professional*, Volume 5, Issue 3, 2003.
8. Hagen, M., Gruhn, V., “Process Patterns - a Means to Describe Processes in a Flexible Way”, *ProSim04*, 2004.
9. Jacobson, I., Booch, G. and Rumbaugh, J., *The Unified Software Development Process*, Addison-Wesley, 1999.
10. Lonchamp J., “Process Model Patterns for Collaborative Work”, *Telecoop'98*, Austria. 1998.
11. OMG, *Software Process Engineering Metamodel 1.1*, 2005.
12. OMG, *SPEM 2.0 Draft Adopted Specification*, 2006
13. OMG, *UML 2.0 Infrastructure Specification*, 2005
14. OMG, *Meta Object Facility Core Specification 2.0*, 2006
15. Pavlov, V. and Malenko, D., “Mining MSF for Process Patterns: a SPEM-based Approach”, *Viking PLoP 04*, 2004.
16. Penker, M., Eriksson, H.E., *Business Modeling With UML: Business Patterns at Work*. John Wiley & Sons 2000.
17. Tran, D.T, Tran, H.N., Coulette B., Crégut, X., Dong T.B.T., “Topological properties for characterizing well-formedness of Process Components”. *Software Process: Improvement and Practice*, Wiley Interscience, V.10 N. 2, p. 217-247, may 2005.
18. Tran, H.N., Coulette B., Dong T.B.T, “A classification of Process Patterns”, *SWDC-REK 2005*. Reykjavik, 2005.
19. Tran, H.N., Coulette B., Dong T.B.T., “A UML based process meta-model integrating a rigorous process patterns definition”, *PROFES 2006*, Amsterdam, 2006.
20. Van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B. and Barros. A.P., “Workflow Patterns”, *Distributed and Parallel Databases*, 14(3), pages 5-51, July (2003)

A Framework for Tailoring Software Process

Lisandra M. Fontoura[1,2,3] and Roberto T. Price[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre RS, Brazil 91.501-970
[2] Centro Universitário Franciscano (UNIFRA)
Santa Maria RS, Brazil 97.010-032
[3]Universidade Regional Integrada do Alto Uruguai e Missões
Santiago RS, Brazil 97.700-000
e-mail: lisandra@inf.ufrgs.br, tomprice@inf.ufrgs.br

Abstract-- Tailored software processes are required when software projects have to adopt development methods and practices to their own specific needs. In this work, activities that can be performed in project processes within an organization, are structured in a global process framework (PRiMA-F), which also includes the process and organizational patterns used to describe risk-preventive and corrective actions. The basic framework structure, built by the organization, may allow different instantiations, creating processes strictly following agile or planned process paradigms, as well as combined approaches, or including activities to comply with quality models or standards, like CMM and others. PRiMA–F defines the software process of an organization, which is later tailored according to risks identified and project specific needs, originating the process that will be used in the project. Goal/Question/Metric Plans are defined to monitor risks. A tool to describe the process framework, with a database of patterns and risks, was developed, in order to aid the generation and adaptation of a process description. Some case studies were elaborated to validate the systematic proposed.

1. Introduction

Software projects differ from each other in relation to technology, complexity, clients, risks, formalism, team, size and other aspects; it is not worth defining a single process to be used applied to all the organizations' projects. It is necessary to adequate the software processes to the application domain and the specific demands of each project.

SW-CMM [9] proposes that an organization's standard software process model (OSSP) be defined, with the fundamental process elements that are expected in all projects and that process be tailored to each project, according to their characteristics and limitations, originating the defined software process for the project [9]. It is possible to constantly improve this process through process evaluation with an OSSP and all processes will benefit, since they are obtained from such OSSP.

Software projects involve risks. Risks can become problems and cause significant impact on time, cost, quality and client's satisfaction. It is necessary to manage risks.

This paper proposes a global framework (PRiMA-F), which integrates all the possible activities to be performed in the project processes, including preventive or corrective actions to risks. These activities are described as organizational and process pattern. Different processes can be instanced from the framework, such as: the software process of an organization and specific processes for the projects; by selecting process elements. This instantiation is done directly selecting process elements, or selecting process patterns associated to identified project risks . Each organization must define its own framework. PRiMA-F was used to create a case study framework, combining planned process activities from RUP [11], agile activities identified in XP[2], and a number of process and organizational patterns associated to risks collected from a number of authors [3,4,5]. Some specific processes were suggested to specific projects using analysis of the risks involved with the projects. Process and organizational patterns are ways to describe successful practices in software management and are used to elaborate software processes and to describe actions to prevent risks. In an associated work [8], process and organizational patterns related to project risks have been collected. The risks are used to identify preventive process and organizational patterns to include in the process for highly possible risks identified during the set up of the project. Tracking risks is necessary to guarantee that risk factors are within the accepted thresholds or to take actions if one of the risk factors goes beyond the allowed thresholds. The Goal/Question/Metric (GQM) paradigm [1] is used in this work to define the risk management plan.

The paper is outlined as it follows: in Section 2 we describe related works. Section 3 describes the PRiMA framework. Section 4 describes case studies. Section 5 describes PRiMA-tool and conclusions are provided in Section 6

2. Related Work

The proposed framework distinguishes itself from other risk management approaches [7,10] by proposing the description of a software process as the integration of organizational and process patterns and to tailor the software process of each project by identifying the risks associated to the project. Therefore the project process will be the combination of all process patterns associated with the identified project risks plus all the mandatory process activities of the organization's global process.

3. Process Framework

The framework is the skeleton of a process from which different processes can be instanced through the selection of process elements, previously defined by the organization, stored in a knowledge base. The organization's knowledge base is composed of a set of process elements used to define process models; a list of possible project risks, processes and organizational patterns, used to represent risk-preventive and corrective actions; risk resolution rules; and goals, questions and metrics used for risk tracking. Figure 1 shows the metamodel representing the main concepts of the knowledge base.

The *Activity*, *Worker*, *Artifact*, *Tool*, *ToolMentor*, *Discipline* and *Template* classes are used to represent the process elements required to model software processes. In the framework used to elaborate the case studies, the knowledge base stores the necessary elements to describe customization of the Rational Unified Process (RUP) [11] and the Extreme Programming (XP) [2]. This framework is just an example; each organization must create its own, according to its needs.

The *ProjectRisk* class represents risks instances that can happen take place in software projects. The Risks are associated to patterns (*Pattern* class) by means of risk resolution rules (*SelectionRule* class) set according to the project context. The context, used in this work, is similar to the one described by Cockburn [4] and by Boehm [3], where the following criteria are analyzed: defects criticality (possibility of loss caused by the defect), team size and team skills [8]. The Processes tailoring guided by the risks and associated preventive and corrective patterns may balance between the practices recommended by agile methods [2] and planned methods [11], and also risk associated process patterns, originating hybrid methods, according to the needs and uniqueness of each project [3]. The *PatternRelationship* class describes associations among patterns.

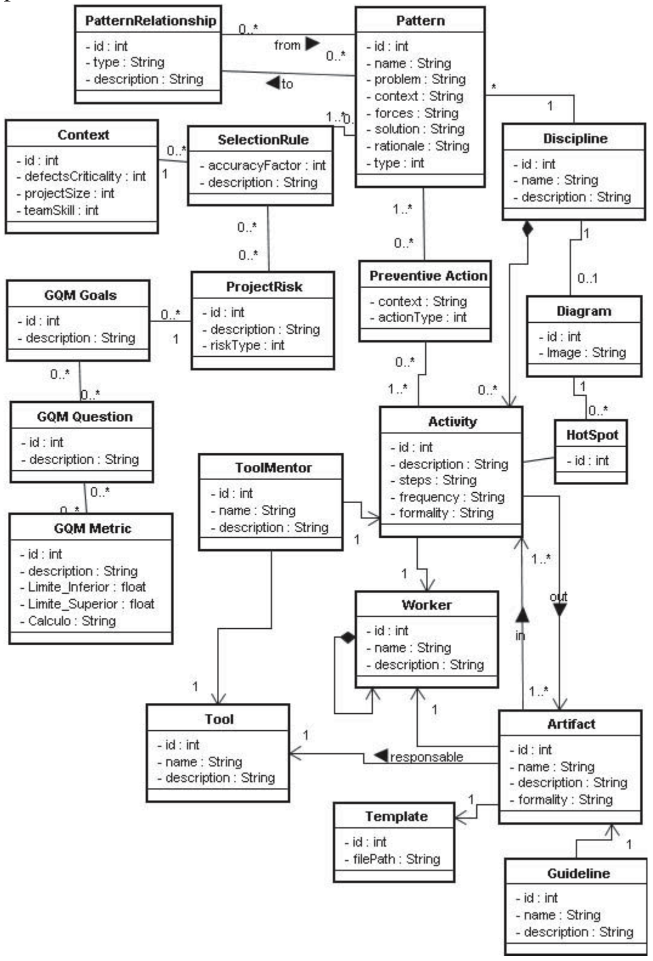


Figure 1. Knowledge Base Class Diagram

Table 1 shows examples of risk resolution rules. The examples of rules and of patterns have been built from practices described in existing published methodologies [2,9,11], and described in previous works [6,8].

Table 1. Example of Risk Resolution Rules

Risk	Process and Organizational Patterns
Misunderstanding the requirements	RequirementsAreValidated [9] ScenariosDefineProblem [5] PlanningGame [2] BuildPrototype [5]
Failure to manage end-user expectations	RequirementsAreValidated [9] BuildPrototype [5] ImpliedRequirements [5] EarlyAndRegularDelivery [5] PlanningGame [2]

Patterns describe solutions in high levels of abstraction. In order to insert patterns in software processes, they need to be described according to concepts (activities, papers, artifacts, etc.) used in process models. The association of patterns with the process elements required in the deployment of software process standards was based on the description of the pattern and on the practices described by models such as the CMM [9], and by processes such as RUP [11], XP [2], SCRUM, PMBOK or by authors as Pressman [12], Sommerville [13], among others.

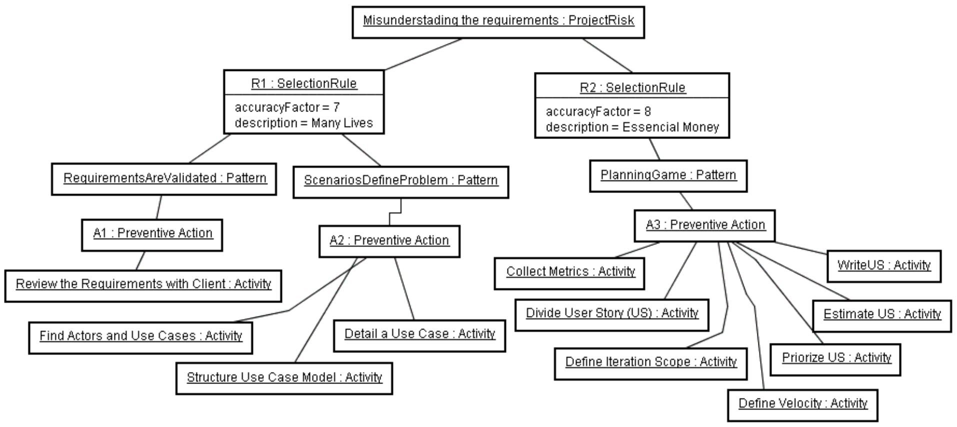

Misunderstading the requirements : ProjectRisk
R1 : SelectionRule
accuracyFactor = 7
description = Many Lives
R2 : SelectionRule
accuracyFactor = 8
description = Essencial Money
RequirementsAreValidated : Pattern
ScenariosDefineProblem : Pattern
PlanningGame : Pattern
A1 : Preventive Action
A2 : Preventive Action
A3 : Preventive Action
Review the Requirements with Client : Activity
Find Actors and Use Cases : Activity
Detail a Use Case : Activity
Structure Use Case Model : Activity
Collect Metrics : Activity
Divide User Story (US) : Activity
Define Iteration Scope : Activity
Define Velocity : Activity
Priorize US : Activity
Estimate US : Activity
WriteUS : Activity

Figure 2. Association of Patterns to Risks and Activities

The Goal/Question/Metric (GQM) Approach [1] is used to define the metrics to be used to track the progress of risk factors, making it possible for the project manager to take corrective actions, when necessary and in the appropriate moment. GQM plans are associated with risks and describe a group of goals (*GQMGoals* class) that aim to monitor risk factors. Goals are unfolded in quantifying questions (*GQMQuestion* class) and then to metrics (*GQMMetric* class) to be obtained to monitor the progress of the project risk factors. Table 2 shows, as an example, the GQM plan defined for the *Misunderstanding the Requirements* risk.

Table 2. GQM Plan: Misunderstanding the Requirements risk

Risk: Misunderstanding the Requirements	
Goal: Analyse the project with the purpose of monitoring the definition of requirements from the viewpoint of the development team in the following context.	
Question: Did the users validate the project requirements documents?	*Metric 1:* Percentage of validated requirements by the client RVRC = (amount of requirement documents validated/ total amount of requirement documents) *100

Activity diagrams show the flow of one activity to the other, making it possible to model the dynamic aspect of the process. An activity diagram (*Diagram* class) is made for each discipline to organize the possible activities to be executed in the software processes of an organization. The activities are associated with the activity diagram by means of extension spots (*HotSpot* class), previously defined for the diagram. Hot spots make it possible for the framework to be flexible and to instance processes for different projects. The activities were organized in activity diagrams by discipline. RUP`s activity diagram was extended to include activities proposed by processes, such as XP, CMM, SCRUM and PMBoK. Activities with common goals are instanced in the same hot spot.

The process designer is responsible for making the activities, roles and artifacts compatible within the processes in the moment designing the activity diagrams and of defining the hot spots, to ensure that the processes instanciated from the framework are consistent.

4. Case Studies

Two case studies were carried out to validate the proposed framework. The case studies were made based on two software projects developed at a university, which will be called Y University. The projects show different characteristics, being one of the projects goal to develop a financial system in one of the Y University campuses (Fin$oft Project), while the other project goal is to develop an academic system in a distributed way, by teams located at three different campuses of the same university (@cadSoft Project). Y University standard process is very simple, and it is based on a small subset of RUP activities.

The team that will develop the Fin$oft software is composed of a project manager, two developers, two trainees and the Computing Center Coordinator, who is the Senior Manager of this project. The team is classified as a high-skilled team, as developers are experienced in the programming language employed and have already developed similar systems. The defined risks for the Fin$oft project are: *failure to manage end user expectations*, *misunderstanding the requirements*, *conflict between user departments*, *scope and goals are not clearly defined*, and *non-realistic schedule and budget*. Among the pattern list suggested by the PRiMA Approach, the patterns selected were: *EarlyAndRegularDeliverXP*, *ConstantRefactoring*, *PlanningGame*, *OnSiteCustomer*, *BuildPrototype*, *DocumentedSoftwareEstimate, SizeTheSchedule Simple-Design* and *DocumentedConfigurationManagementPlan*.

The team that is developing Fin$oft financial software is small, with developers who are knowledged on the technology to be used in the project. The client is within the campus where the system will be developed, so the project configuration make possible to make use of the most agile methodologies. The main patterns suggested by the PRiMA Approach are those that aim to provide more agility to the organization standard process, based on RUP.

Another case study, @cadSoft academic software will be developed by three teams, totalizing 22 people: 3 project managers, 8 developers, 10 trainees and 1 senior manager, based on the Y University administrative area. The team is classified as average skilled due to the number of trainees in the team and the fact that the team does not master the technology chosen for the system development. The defined risks for the @cadSoft project are: *lack of a methodology for the project*, *lack of required knowledge/skill in the project*, *misunderstanding the requirements*, *introduction of new technology*, *wrong development of functions of user interfaces*, *unfeasible design*, and *lack of top management commitment to the project*. Among the suggested patterns list to prevent risks, the following were selected: *ConstantRefactoring*, *SoftwareLifeCycleIsDefined*, *EarlyRegularDeliverRUP*, *ScenariosDefineProblem*, *ProjectProcessIsDefined*, *ShunkWorks*, *ArchitectureTeam*, *PeerReviews*, *ApprenticeShip*, *SeniorManagementReview* and *DocumentedConfigurationManagementPlan.*

In the @cadSoft Academic System's process, the situation presented is opposed to that of the Fin$oft, because the team is distributed in different campuses, and the customer, in this case the University administrative area, is far from the development teams, the team is composed of many trainees, who can stay in the team only for short periods. The most relevant patterns, suggested by PRiMA-Tool, are patterns which aim to provide more planning and documentation to the software process. Considering the difficulty in face to face communication in distributed teams, documents are generated so that the teams can communicate and keep informed.

5. PRiMA-Tool

An experimental environment was developed composed of two tools, which are: Pattern-Based Methodology Tailoring Tool (PMT-Tool) and Project Risk Management Approach Tool (PRiMA-Tool). PMT-Tool module was developed by Júlio Hartmann [8]. PMT-Tool is responsible for cataloguing the process patterns and associating them with the software risks by means of preventive rules, as well as selecting the pattern to prevent prioritized risks in a specific project. PRIMA-Tool module is responsible for the elaboration of the project software process, from organization's standard process tailoring, inserting in it the selected patterns to prevent the project risks and defining the Goal/Question/Metric Plans to manage the project risks.

Having concluded the project process tailoring, Prima-Tool generates a website with the description of the defined process for the project to be consulted by developers, managers and process engineers. The tools are available for interested readers to play with at http://www.urisantiago.br/lisandra/prima/.

6. Summary and Conclusion

This article proposes a process framework, which makes it possible to instantiate development processes tailored according to the identified and prioritized risks of the development project. This is made easier trough a tool that supports the processes instantiation from the framework. The tailoring of the process is according to the risks of the project. The tailoring process is based on the information, rules and associations inserted in the knowledge base describing risks, organizational patterns, associations between risks and patterns and the global development process. The knowledge base described in the studies cases is a suggestion and works as an example. The knowledge base of any organization deploying the approach must be constantly revised and evaluated by the process engineers, aiming at continuous improvement. Results of postmortem analysis of projects may help in this task.

Future works include the use of a workflow management system (WMS) to generate environments to support the execution of processes defined from PRiMA-F, the association of activities with tools, and the automated of collection of metrics to monitor risks.

References

[1] Basili, V. R.; Seaman, C. The Experience Factory Organizational. IEEE Software, New York, v. 19, n. 3, p. 30-31, May 2002.

[2] Beck, Kent. Embracing Change with Extreme Programming. IEEE Computer, 32:70--77, Oct. 1999.

[3] Boehm, B.; Turner, R. Using Risk to Balance Agile and Plan-Driven Methods. IEEE Computer, June 2003.

[4] Cockburn, Alistair. Selecting a Project's Methodology.IEEE Software, July/August 2000.

[5] Coplien, James. Sofware Patterns. SIGS Books and multimedia. http://www1.bell-labs.com/user/cope/Patterns/WhitePaper/. 1996.

[6] Fontoura, L. M. PRiMA: Project Management Risk Approach. Tese de Doutorado, UFRGS, 2006.

[7] Kiper, J. D.; Feather, M. S. A Risk-based Approach to Strategic Decision-Making for Software Development. ICSS, 2005.

[8] Hartmann, Júlio; Fontoura, Lisandra M.; Price, Roberto T. Using Risk Analysis and Patterns to Tailor Software Processes. XIX Simpósio Brasileiro de Engenharia de Software, 2005.

[9] Paulk, Mark C. et. al. Key Practices of the Capability Maturity Model, Version 1.1, TR CMU/SEI-93-TR-025, 1993.

[10] Roy, G. C. A Risk Management Framework for Software Engineering Practice. ASWEC, 2004.

[11] Rational Software Corporation. Rational Unified Process, v. 2003. Cupertino, 2003.

[12] Pressman, R. Engenharia de Software, McGraw-Hill, 2006.

[13] Sommerville, I. Engenharia de Software, Prentice-Hall, 2003.

Analyzing Configuration Management Repository Data for Software Process Improvement

Shihong Huang
Computer Science & Engineering
Florida Atlantic University
shihong@cse.fau.edu

Christopher Lo
Computer Science & Engineering
Florida Atlantic University
clo1@fau.edu

ABSTRACT

The software development process is an incremental and iterative activity. Source code is constantly changed to reflect changing requirements, to respond to testing results, and to address problem reports. Proper software measurement that derives meaningful numeric values for some attributes of a software product or process can help in identifying problem areas and development bottlenecks. Objective assessment is needed of the current status of software development so that informed project managers can make decisions. This paper presents a methodology called VITA for applying software analysis techniques to configuration management repository data with the aim of identifying the impact on file changes due to change requests and problem reports. The repository data can be analyzed and visualized in a semi-automated manner according to user-selectable criteria. The approach is illustrated with a model problem concerning software process improvement of an embedded software system in the context of performing high-quality software maintenance.

Keywords: impact analysis, process improvement, knowledge management, repository, software maintenance, visualization

1. INTRODUCTION

Large-scale software systems must be continuously maintained and evolved to respond to shifting business requirements. Business decisions are often made based on the measurements of the software quality and priorities of business goals. The measurements should be able to check if the software had reached the required quality threshold, and highlight areas of the software development where special foci are needed. Software product measurements can be used to make general predictions about a system's attributes, such as the number of faults in the system, to identify anomalous components that are likely to have more errors that management can concentrate on these components during quality review process.

A number of large companies such as Hewlett-Packard [4], AT&T [1], and Nokia [9] have introduced metrics programs and are using collected metrics in their quality management processes. Most of the focus has been on collecting metrics on program defects and the verification and validation processes. The measurement used could be control measurements used by software process or predictor measurements used by software products [13], both of which could influence management to make decisions about the software systems.

However, some of software process external quality attributes often cannot be measured directly. Questions such as, "Are the developers distributed efficiently?" "Do the components require refactoring?", and "Where do we spend most of our development resources?" are affected by many factors; there is no simple way to answer them. One of the possible solutions to measure these external quality attributes is to use internal attributes that can be derived from entities of software development process, such as configuration management repository.

A configuration management repository includes abundant data of not only configuration items, but also the use of software components, proposed changes, components that could be affected by these changes, and number of changes per entry. However, some of the knowledge is implicitly hidden in the repository data and cannot be obtained by doing simple queries.

This paper presents an integrated approach to process improvement for identifying production problems areas and potential bottlenecks in development. As the first step of this project, software analysis techniques were used on configuration management repository data with the aim of identifying and visualizing the impact on file changes due to change of requests and problem reports. The approach realizes the goal of providing objective assessments of current software production processes and makes implicit information that are hidden in the configuration management repository explicit so that informed decision can be made by project managements.

The next section discusses some of the fundamental issues related to software process improvement and measurement techniques. Section 3 outlines an integrated approach to analyze and visualize configuration management repository and quantify qualitative data with the aims of helping identify the impacts of particular change requests and problems reports. Section 4 provides a real-world case study involving a large IT company's non-proprietary data to illustrate the approach and highlight the key steps of this methodology. Finally, Section 5 summarizes the paper and outlines possible avenues for future research.

2. RELATED WORK

A software process is a set of activities that leads to the production of a software product [13]. Process improvement is based on the assumption that the quality of the engineering process is critical to product quality. Normally, dramatic changes to existing process are not a viable solution; it can only generate turbulence within the organization. Therefore, sometimes a quiescent, non-invasive, and adoption-centric approach to process improvement is needed [7]. To identify software process problems areas and bottlenecks, objective assessment of the process and quantifying qualitative data is need. Software metrics, specially looking at configuration management repository data can provide valuable feedback of the system.

There have been many studies conducted on software metrics [8][10]. Metrics from repositories of configuration management are able to provide developers and analysts with large quantities of data regarding developers' work habits that might help analysts deduce which modules require the most amount of effort. Studies have been conducted that utilize metrics for effort estimation in several capacities. More often than not, the metrics that are acquired from previous efforts made to the system can help in estimating how much effort is required for similar changes in a related feature and module of similar size [5][8][10]. When one requires information about how to proceed into the future, it is useful to look to the past for guidance. In addition to metrics, the requirements documents of a project are also treasure troves of information regarding logical and natural design that occurs from requested features. The interaction between certain requirements provides an understanding of how to organize the large features and modules of the system [10].

To identifying development process bottleneck an problem areas, a set of metrics of the process need to be collected. Basili *et al* introduced the Goal Question Metric (GQM) approach for collecting data with a purpose [2]. By spending the proper amount of time on defining the business goals, questions can then be derived from these goals. Finally, the appropriate metrics can be collected. The three levels that GQM address correlates with the three parts of the approach are the conceptual, operational, and quantitative. These three levels relate accordingly to the goal, question, and metrics that can be obtained for improving new or existing processes.

3. VITA METHODOLOGY

The **V**isualization of **I**mpac**T** **A**nalysis (VITA) methodology incorporates the theories from Anaylitic hierarchy Process (AHP) [11], GQ(I)M, and DMADV/DFSS in order to analyze and visualize configuration management repository data. Analysts first determine the goals and questions that are to be asked of the system and the data. Next, these goals are classified as alternatives and thusly prioritized according AHP. From this step, an analyst is able to derive the most pressing goals and focus attention on gathering the metrics necessary to answer these questions. These metrics combined form indicators that will be used as gauges throughout the rest of the process. Thresholds will be determined in order to see if the indicators chosen have remained static or have breached the threshold. The time frame can be arbitrarily set to any range depending on the approach that is taken or at the analyst's discretion. The impacts of change of requests and problem reports are visualized in static and dynamic graphs. The entire process can be performed semi-automatedly and indefinitely in a recursive fashion or until focus is shifted to another goal. The following sections details the steps taken by VITA methodology.

3.1 Goals, Questions, and Metrics

GQM is used to guide the creation of different types of metrics that could be used to measure the current status of the development process. The goals are to be determined by two parties. The first is that of the data provider that acts as the customer in this situation. The customer can make suggestions as to their intentions as well as areas of focus. The second source of goals is to be derived from the researcher. The researcher should choose a set of goals which will satisfy their research [6]. After both parties have defined their goals, priorities should be assigned to these goals. In this particular project, there are several goals in mind. The first is that of determining the grouping of change requests in an effort to locate virtual clustering of files. The second is to identify and improve developer effort distribution. These two goals can be mapped into a set of questions that map to metrics. Samples of the mapping of the questions to be answered and metircs related to these questions are listed in the following tables. These metrics are generated by using GQM and AHP methodologies mentioned in Section 2.

Based on the GQM methodology, three metrics tables were generated to help managers to make informed decisions about the software development process. The first table is Developer Effort Distribution Focused. This table could be used to answer question such as "Are the Developers distributed Efficiently?" It includes 19 entrances of matrics such as Time spent per file, Time spent per branch etc. The second table is Problem Report Focuses. It could be used to answer question such as "Do the Components Require Reorganization?". It includes 14 entrances of matrics such as Number of files per branch, Branches with the most test hours etc. The third table is Change Request Focused. It coul be used to answer question such as "Where are the Clusters Reated to CRs and PRs?". It includes 12 entrances of metrics such as Time spent per file in change request (CR), Time spent per branch in CR etc.

3.2 Threshold

The threshold is the maximum imposed level a given metric can achieve before passing into a warning level that warrants further investigation. A threshold does not usually exist naturally unless a governing body such as the managerial staff of a project or a development team specifies it. At the initial stages, there is not a set of threshold gauges to compare to all measurements in each smaller iteration or phase. Therefore, the initial thresholds can be obtained by deriving the average and median from the whole of the data set. However, like any statistical analysis, there will be outliers. Hopefully, the difference between the average and median will be small enough to take the halfway point between the two numbers.

4. CASE STUDY

The previous section outlined an integrated approach (VITA) that uses configuration management repository data to identify software development problems areas and bottlenecks. This section illustrates this approach by using real-world data from a large industrial partner's non-proprietary data. The VITA method takes three-step approach [14]: gathering data from repository, analyzing and integrating data from different sources (i.e., knowledge management), and visualizing clustering information according to end-users' selection.

4.1 Data Gathering

A collection of non-proprietary repository data from a large industrial partner was used for this research. There are two sets of data that were acquired for this case study. The first set, denoted by *Repository Activity Table*, is a collection of day-to-day concurrency versioning system data that is collected as a developer checks out a file as well as relating it to the appropriate work order. Each working set entry contains information such as filename, working directory, branch directory, date, time, action taken, work order number, as well as developers.

The second set of data, denoted by *Working Order Table*, is comprised of a unique work order number, a work order type (problem report or change request), date submitted, date resolved, fix hours, and development test hours.

4.2 Knowledge Management

The information gathered from the first step was then parsed into a database with the logical organization of the pieces of the data in order to provide meaning to each row of data. A random section of data was selected for analysis.

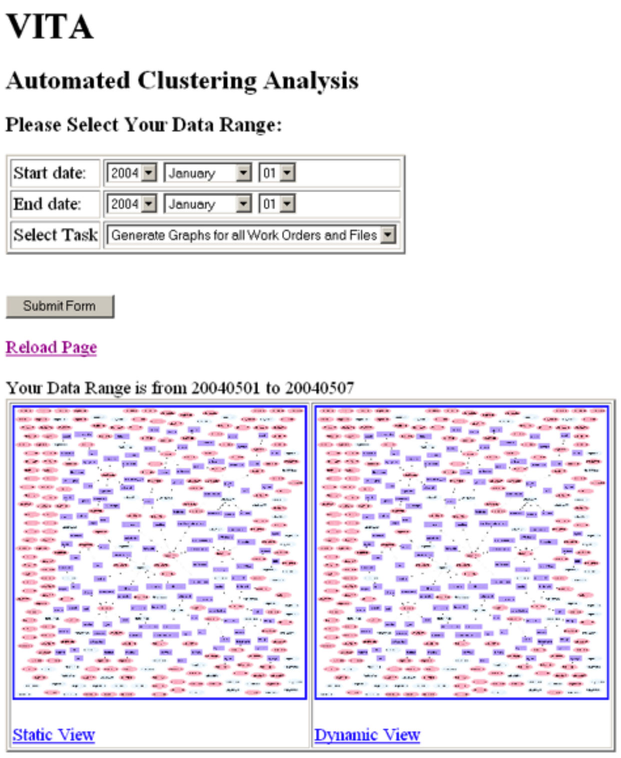
As an illustrative example of this approach, this paper uses the first two weeks of data provided by industrial partner to show the feasibility of the VITA methodology. The earliest two weeks were taken into analysis so that any hypotheses developed for these two weeks can then be applied to following two week sections of time in the data. It was taken into consideration the size of the organization from which this information was derived, therefore, two weeks represents a meaningful section of time in which requests can be made as well as significant bugs have been remedied for large pieces of the code.

4.3 Information Visualization

The last part of the methodology is information visualization. Information visualization often deals with data that are normally large, semi-structured, or multivariate. The visualization can display and interactively explore data that might be abstract and may not be intuitively comprehensible [3]. To make the VITA methodology easy to use by end-user and to show the results of clusters in an intuitive way, a configuration management repository visualization tool called VITA Toolkit has been developed. From the preliminary prototype, a user can select timeframe that they are interested and two types of graphic views, static view in JPEG, PNG, or GIF and dynamic view in SVG format. Examples of these two types of graphical view are shown in Figure 1.

4.3.1 Sample Clustering Visualization

By using the VITA Toolkit, the first two weeks' data from the industrial partner was analyzed and visualized. The first week of this two-week time chunk showed that there are many files that gravitate towards each other due to particular change requests and problem reports. The diagram in Figure 2 shows the excerpts generated by VITA Toolkit from the total clustering of the total of two weeks (1st and 2nd week) of analysis. Color codes are used to distinguish different artifacts in the analysis. For example, pink ovals represent problem reports (CRs), light blue ovals represent change requests (PRs), and slate blue boxes represent files. The edges that connect the boxes to ovals represent a relationship between a change request and a file or a problem report and a file.


VITA

Automated Clustering Analysis

Please Select Your Data Range:

Start date:	2004	January	01
End date:	2004	January	01
Select Task	Generate Graphs for all Work Orders and Files		

Your Data Range is from 20040501 to 20040507

Figure 1: VITA Clustering Analysis Toolkit

4.3.2 Result Analysis

Preliminary results from these two weeks of data suggest that there is certain clustering of files that gravitate to each other due to a change request. It is likely these files show a natural affinity due to the design. Certain change requests and problem reports had multiple files associated with them. In addition, the reverse was true where certain files saw multiple change requests and problem reports during the specified time frame. In addition to these, there are many files that exist without the attachment to any change requests or problem reports as well as change requests and problem reports that do not have any nodes connected to them. This could be due to the fact that the change request or problem report that required the work of these specific files exists outside of the desired time frame. Another possibility is that a change request or a problem report was made during the time frame but no action was taken yet.

By acquiring the knowledge of the files associated with a particular change request or problem report, an analyst would be able to make assumptions at a high level and keep track of all other files that were manipulated together so that future work on a particular file might raise a flag towards the developer so that considerations should be taken towards other files that were previously associated with the file in question. In addition, a file with the highest problem report to change request ratios could be deemed high risk and therefore warrant extra attention in the next stage of development.

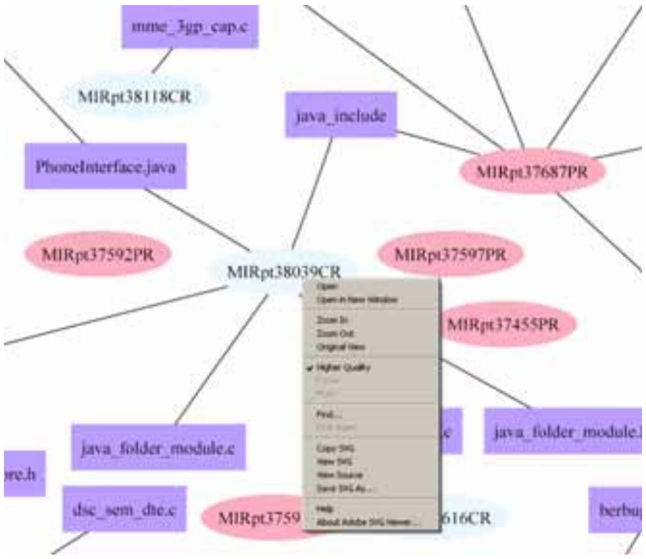


Figure 1: Clustering Analysis for Two Weeks Data (Partial Graph)

5. SUMMARY

This paper presented a methodology for software process improvement, specifically visualize the impact analysis by analyzing configuration management repository. As the first step in identifying the development bottleneck and problem areas, an impact analysis and visualization toolkit VITA of files changes due to change request and problem report was developed. A case study that applies this methodology by using real-world data from a large industrial partner illustrates the feasibility of the methodology.

Although this project does not involve analysis at the code level, it does provide meaningful input for further code level analysis because each artifact in the generated graphic view is clickable to the source related to it. Goal Question Indicator Metrics methodology combined with Six Sigma's DMADV methodology can prove to be a rather successful combination – specially if priorities are set using the Analytic Hierarchy Process. The advantage of a statistically-based prioritization and decision-making process is that it allows managers and individuals in the position to make decisions regarding a project to possess increased credibility regarding their reasons for certain choices. The GQ(I)M approach places emphasis on the goals and questions that should be asked of the process improvement initiatives.

Future work will include applying statistical methods, such as classification and regression tree algorithms (C&RT), to the repository data. Identifying and predicting continues variables or categorical variables for development process. For example, to identify the clusters of files that are likely to be changed together over the development process, to find the files with the highest Problem Report to Change Request ratio that might suggest that these files are prone to errors and bugs and warrant consideration as to redesigning [12]. In addition, statistical analysis of the files, problem reports, and change requests should be made in order to help focus the study.

REFERENCES

[1] Barnard, J. Price, A. "Managing Code Inspection Information." *IEEE Software*, 11(2), 59-69, 1994.

[2] Basili, V. R., Caldiera, G., Rombach, H. D. "The Goal Question Metric Approach." *Encyclopedia of Software Engineering,* Wiley&Sons Inc., 1994.

[3] Gansner, E., North, S. "An Open Graph Visualization System and Its Applications to Software Engineering." *Software: Practice and Experience*. 1999.

[4] Grady, B. "Practical Results from Measuring Software Quality." *Communication of the ACM*, 36(11), 62-68, 1993.

[5] Graves, T.L., Mockus, A. "Inferring change effort from configuration management data." *Metrics 98: Fifth International Symposium on Software Metrics*, pages 267-273, Bethesda, Maryland, November 1998.

[6] Hayes, J.H., Dekhtyar, A., Sundaram, S.K., Howard, S. "Helping analysts trace requirements: an objective look'" *Requirements Engineering Conference, 2004. Proceedings*. 12th IEEE International 2004. pp. 249 – 259

[7] Huang, S.; Tilley, S.; VanHilst, M.; Distante, D. "Adoption-Centric Software Maintenance Process Improvement via Information Integration." *Proceedings of the 13th IEEE International Conference on Software Technology and Engineering Practice* (STEP 2005: September 24-25, 2005; Budapest, Hungary). Los Alamitos, CA: IEEE Computer Society Press, 2006

[8] Huffman Hayes, J., Patel, S., and Zhao, L. "A Metrics-Based Software Maintenance Effort Model." *Proceedings of the 8th European Conference on Software Maintenance and Reengineering,* Tampere, Finland, March 2004. pp. 254-258. http://selab.netlab.uky.edu/Homepage/csmr_ameffmo_hayes_2004%5Eas_published.doc

[9] Kilpi, T. "Implementing a Software Metrics Program at Nokia." *IEEE Software*, 18(6), 72-77, 2001.

[10] Lehman, M.M., Perry, D.E., and Ramil, J.F. "Implications of Evolution Metrics on Software Maintenance." ICSM'98, November 1998. http://www.ece.utexas.edu/~perry/work/papers/feast2.pdf

[11] Mustafa, M.A., Al-Bahar, J.F. "Project risk assessment using the analytic hierarchy process." *IEEE Transactions on Engineering Management*. Volume 38, Issue 1, Feb. 1991 Page(s):46 – 52

[12] Sandusky, R.J., Gasser, L., Ripoche, G. "Bug Report Networks: Varieties, Strategies, and Impacts in a F/OSS Development Community." 2004.

[13] Sommerville, I. *Software Engineering* (8th Edition). Addison-Wesley, 2006.

[14] Tilley, S. *A Reverse-Engineering Environment Framework*. Software Engineering Institute, Carnegie Mellon University. Technical Report CMU/SEI-98-TR-005, 1998.

Smooth Quality Oriented Component Integration through Product Line Based Aspect-Oriented Component Adaptation

Yankui Feng, Xiaodong Liu and Jon Kerridge
School of Computing
Napier University
Edinburgh, UK
E-mail: {y.feng, x.liu, j.kerridge}@napier.ac.uk

Abstract

Mismatches in QoS (Quality of Service) often appears as a major but implicit hurdle to smooth component integration. This paper presents a solution to the above problem through product line based aspect oriented component adaptation. The approach enjoys high level of automation and capability of deep level adaptation, which is achieved in an aspect-oriented component adaptation framework by generating and then applying the adaptation aspects under designed weaving process. The aspect product line facilitates the creation of adaptation aspects that support specific adaptation requirements. An expandable repository of reusable adaptation aspects has been developed. The approach and its realization are integrally demonstrated and verified with a case study.

Keywords: Component-Based System, Software Product Line, Aspect-Oriented Programming, Software Reuse, Quality of Service.

1. Introduction

Component-Based Development (CBD) has been proved as an effective technology in supporting community-wide reuse of software assets [12]. Recently, the methodology of CBD has been expanded to build more challenging systems, such as dependable embedded software and web services, which consequently imposed more rigid requirements on the methodology itself, in particular, in the QoS (Quality of Service) aspect [5][6].

Due to the availability of components and the diversity of target applications, in many cases pre-qualified available components still have mismatches to the specific reuse requirements in particular applications. This problem has been a major hurdle of wider component reusability and smooth component composition. The mismatches may occur in aspect of functionality or QoS such as performance, security and safety, and degrade the target component-based system severely. Component adaptation has been researched over the years as a key solution to the above problem [2] [10]. Due to the complex nature of the mismatch problem, available approaches are either only capable for adaptation at simple levels such as wrappers [2], or inefficient to use due to the lack of automation in their adaptation process. To assure the QoS of the target component-based system, more efficient automated adaptation mechanisms are in need to eliminate component mismatches. Aspect oriented programming (AOP) [9][11] is an ideal technology for QoS oriented component adaptation due to its unique advantages in addressing non-functional problems such as security and performance.

Our approach, namely Generative Aspect-oriented component adaptatIoN (GAIN) [4][8] is proposed to achieve automated component adaptation at a rather deep level, particularly aiming at eliminating mismatches in QoS such as system performance and dependability. In this paper, a case study of an online testing system is presented to demonstrate and verify our approach and its implementation. Compared with traditional AOP[9][11][13][14], the weaving process of aspects in GAIN supports more complex control flow, i.e., not only sequence, but also switches, synchronization and multiple threads, in order to deal with adaptation in more complicated environments such as concurrent and real-time applications. To facilitate the reusability of adaptation knowledge, based on software product line techniques [1][3], an expandable repository of reusable adaptation aspects is developed.

The reminder of the paper is organized as follows: Section 2 describes the approach framework, including aspect-oriented generative component adaptation process, aspect repository and process-based component adaptation specification. Section 3 presents the prototype tool, and a case study to demonstrate and verify the approach and its implementation. Section 4 discusses the related work. Finally, section 5 presents the conclusion.

2. The GAIN approach

2.1 The approach process

Figure 1 describes the process of the aspect-oriented generative component adaptation. We presume that in a component based system, a component has been found potentially suitable for an application, but some mismatches still exist, therefore, the application developer wishes to have the component adapted.

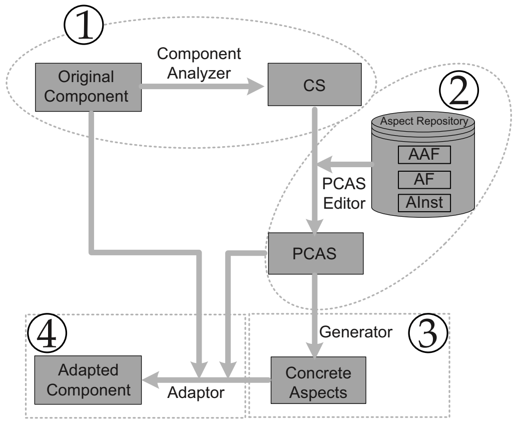


Figure 1: The process of GAIN Tool

The mismatch will be eliminated by applying aspect-oriented adaptation to the original component. First, as shown in figure 1, the component is analyzed with the component analyzer, which analyzes the source or binary code of the component and extracts component specification information, e.g. class names, method signatures and quality features. The component specification will be used to guide component adaptation. If the component already has well defined specification, this step can be skipped.

Then, based on the adaptation requirements, a Process-based Component Adaptation Specification (PCAS) will be created by selecting aspect types, namely Abstract Aspect Frames (AAF), from Aspect Repository and defining the weaving process of these aspects. The selection of aspects is actually the process to determine functional variation of a specific adaptation. The composition of PCAS is supported by an interactive IDE called PCAS Editor, which supports both graphical and XML source view of the PCAS.

In step 3, based on PCAS and the lower level aspect definition, namely Aspect Frame (AF) in the Aspect Repository, executable aspects instances (AInsts) are generated by the aspect generator according to different AOP implementation specifications. As a result, platform variation is achieved during aspect generation. The input for the aspect generator is XML formatted aspects and the output is concrete aspect instances in an executable form.

In the last step, to implement PCAS in the weaving process, a post-weaving technique is developed. The post-weaving tool gets class files for aspects generated by AOP platform such as AspectJ as input, and then modifies those class files to generate new class files that support complicated flow control and synchronization according to PCAS.

2.2 The aspect repository

The aspect repository is an embodiment of the proposed product line based reusable aspect model. Reusable aspects are defined at three abstraction levels and kept in the repository as AAF, AF, and AInst. The reusable assets in the repository include both primitive and composite aspect types, which come from the adaptation process in PCAS. More details of Aspect Repository is described in [4][8].

2.3 The PCAS

To satisfy the adaptation requirements for a particular reuse context, it often requires performing complex adaptation to multiple components with a set of generated aspects applied to these components under a specially designed process containing conditions, synchronization and other flow controls. Process-based Component Adaptation Specification (PCAS) is developed to describe the above complicated adaptation details.

If a PCAS is found common and reusable in the future, its process control part can be regarded as a composite aspect type. Composite aspects are supported in AAF level to achieve advanced reuse in typical aspect using cases.

A PCAS is an XML formatted document, which includes the details of component adaptation, such as the target component, the weaving process, and the abstract aspects to be applied. In a PCAS, sequence and switch structure are supported to achieve flexible adaptation on components. The adaptation process in PCAS is depicted with only the ID of the selected aspects. Full details of the aspects are still kept in Aspect Repository.

3. Tool Realisation and Case study

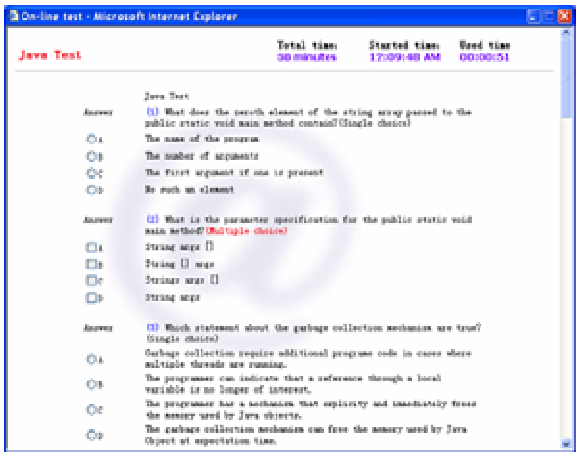
3.1 Tool realisation

A CASE tool has been developed to facilitate the proposed approach. With the tool, component developers define aspect weaving process by drag-and-drop in a graphical tool, namely PCAS Editor. They select candidate aspects and fill in necessary details in Aspect

Manager. The Aspect Generator generates concrete aspect automatically. According to the defined PCAS, Adaptor completes the aspect weaving and generates the final adapted components.

3.2 The problem of current system

Case studies have been done to verify the approach, in terms of its capability of improving QoS such as system performance in a component based system. In this section, we use the following case study to demonstrate how PCAS works and how to generate an executable aspect by mapping through the different abstraction views of the aspect in our framework.



On-line test - Microsoft Internet Explorer

Java Test

Total time:	Started time:	Used time
30 minutes	17:09:40 AM	00:00:51

Java Test

Answer (1) What does the zeroth element of the string array passed to the public static void main method contain? (Single choice)

○ a The name of the program
○ b The number of arguments
○ c The first argument if one is present
○ d No such an element

Answer (2) What is the parameter specification for the public static void main method? (Multiple choice)

☐ a String args []
☐ b String [] args
☐ c Strings args []
☐ d String args

Answer (3) Which statement about the garbage collection mechanism are true? (Single choice)

○ a Garbage collection require additional programe code in cases where multiple threads are running.
○ b The programmer can indicate that a reference through a local variable is no longer of interest.
○ c The programmer has a mechanism that explicity and immediately frees the memory used by Java objects.
○ d The garbage collection mechanism can free the memory used by Java Object at expectation time.

Figure 2: On-line testing component

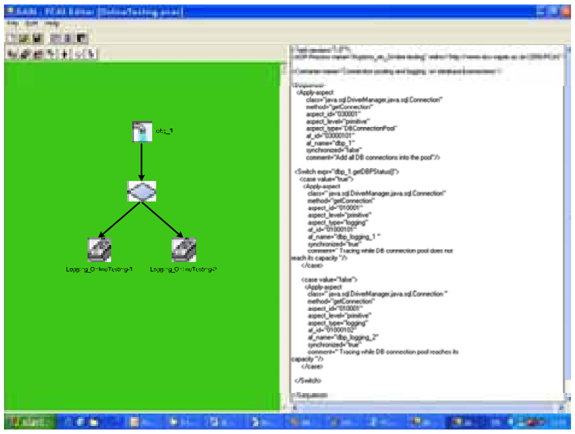
As shown in figure 2, the case study regards with an on-line testing component, which was developed by Component Source, a software company selling COTS components. The IT department of a university planned to build its own online assessment system and bought the component for integration as part of the system. However, the system developers identified that the heavy access load imposed by the large number of students has made the system performance poor.

3.3 The adaptation requirement

The development team decided that prior to integration of the online testing component three actions should be done to adapt the performance of the component. First, a database connection pool is to be introduced to the online testing system to improve system performance. Then, logging is used to monitor the usage of the connection pool. Finally, based on the logging information, the connection pool is tuned to achieve the best performance with reasonable resource cost such as memory consumption, by adjusting the parameters, including the capacity of connection pool and the time of expire of a connection instance.

3.4 Generative aspect-oriented adaptation

To meet the reuse requirements, the following three aspects are applied to the component to implement the above adaptation actions, namely database connection pool, logging if connection pool reaches its maximum capacity, and logging if connection pool does not reach its maximum capacity. These adaptation actions are then described in a Process-based Component Adaptation Specification (PCAS). The specification is created with the PCAS Editor shown in figure 3 by finding appropriate AAFs, i.e., either primitive types or composite types of aspects, and putting these AAFs into an adaptation process. Functional variation of adaptation is implemented through the composition of PCAS.



```
<?xml version="1.0"?>
<AOP-Process name="Aspects_on_On-line testing" xmlns="http://www.dcs.napier.ac.uk/2005/PCAS">
<Container name="Connection pooling and logging on database connections">
<Sequence>
  <Apply-aspect
     class="java.sql.DriverManager,java.sql.Connection"
     method="getConnection"
     aspect_id="030001"
     aspect_level="primitive"
     aspect_type="DBConnectionPool"
     af_id="03000101"
     af_name="dbp_1"
     synchronized="false"
     comment="Add all DB connections into the pool"/>
 <Switch expr="dbp_1.getDBPStatus()">
   <case value="true">
    <Apply-aspect
      class=" java.sql.DriverManager,java.sql.Connection"
      method="getConnection"
      aspect_id="010001"
      aspect_level="primitive"
      aspect_type="logging"
      af_id="01000101"
      af_name="dbp_logging_1 "
      synchronized="true"
      comment=" Tracing while DB connection pool does not
reach its capacity "/>
   </case>

   <case value="false">
    <Apply-aspect
      class=" java.sql.DriverManager,java.sql.Connection "
      method="getConnection"
      aspect_id="010001"
      aspect_level="primitive"
      aspect_type="logging"
      af_id="01000102"
      af_name="dbp_logging_2"
      synchronized="true"
      comment=" Tracing while DB connection pool reaches its
capacity "/>
   </case>

 </Switch>

</Sequence>
```

Figure 3: An environment to create PCAS (PCAS Editor)

The content of PCAS is shown in figure 4:

```
<?xml version="1.0"?>
<AOP-Process name="Aspects_on_On-line testing"
xmlns="http://www.dcs.napier.ac.uk/2005/PCAS">
<Container name="Connection pooling and logging
                       on database connections">
<Sequence>
  <Apply-aspect
       class="java.sql.DriverManager,java.sql.Connection"
       method="getConnection"
       aspect_id="030001"
       aspect_level="primitive"
       aspect_type="DBConnectionPool"
       af_id="03000101"
       af_name="dbp_1"
       synchronized="false"
       comment="Add all DB connections into the pool"/>
 <Switch expr="dbp_1.getDBPStatus()">
   <case value="true">
     <Apply-aspect
```

```
        class=" java.sql.DriverManager,java.sql.Connection"
        method="getConnection"
        aspect_id="010001"
        aspect_level="primitive"
        aspect_type="logging"
        af_id="01000101"
        af_name="dbp_logging_1 "
        synchronized="true"
        comment=" Tracing while DB connection pool does not
reach its capacity "/>
      </case>
      <case value="false">
       <Apply-aspect
        class=" java.sql.DriverManager,java.sql.Connection "
        method="getConnection"
        aspect_id="010001"
        aspect_level="primitive"
        aspect_type="logging"
        af_id="01000102"
        af_name="dbp_logging_2"
        synchronized="true"
        comment=" Tracing while DB connection pool reaches its
capacity "/>
      </case>
  </Switch>

    </Sequence>
  </Container>
  </AOP-Process>
```

Figure 4: A Process-based Component Adaptation Specification for Online testing system

The specification in PCAS is at a rather overview level and does not contain the details of individual aspects. Developers need to provide parameter values for each aspect. Common AFs can be saved into Aspect Repository for further reuse. As shown in figure 5, the Aspect Manager is a tool to manage reusable aspects in the aspect repository, and to present graphical views of aspects at various abstraction levels.

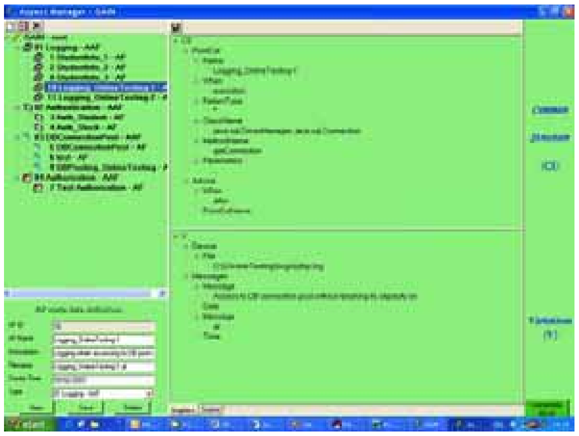


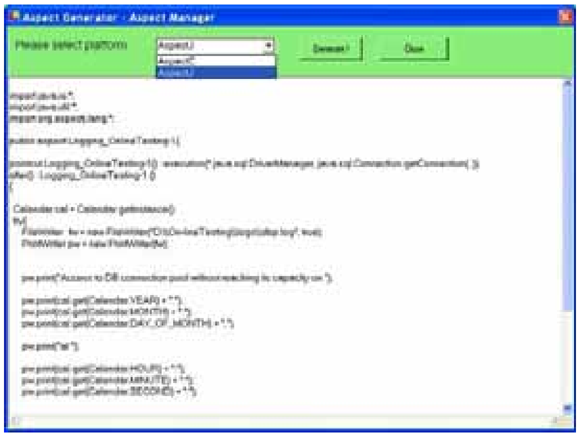
Figure 5: Manipulate aspects in Aspect Manager

In this example, three AFs will be generated for each of the above aspects accordingly. Due to the structural similarity of AFs of different aspects, we only give the AF for logging while DB connection pool does not reach its capacity in figure 6 as an example.

```
<?xml version="1.0" ?>
<Aspect name="dbp_logging_1">
 <!—Common Structure -->
 <CommonStructure>                                   CS
   <PointCut>
      <Name>dbp_pointcut_1</Name>
      <When>execution</When>
      <ReturnType>*</ReturnType>
      <ClassName>
            java.sql.DriverManager,java.sql.Connection
      </ClassName>
      <MethodName>getConnection</MethodName>
      <Parameters>..</Parameters>
   </PointCut>
   <Advice>
      <When>after</When>
      <PointCutName ref=" dbp_pointcut_1" />
   </Advice>
 </CommonStructure>

 <!-- Variations -->
 <Variation type="logging">
   <Output>
     <Device>
       <File>D:\On-lineTesting\logs\dbp.log</File>
     </Device>
     <Messages>
       <Message>Access to DB connection pool
              without reaching its capacity on
       </Message>
       <Date/>
       <Message>at </Message>
       <Time/>
     </Messages>
   </Output>                                          V
 </Variation>
</Aspect>
```

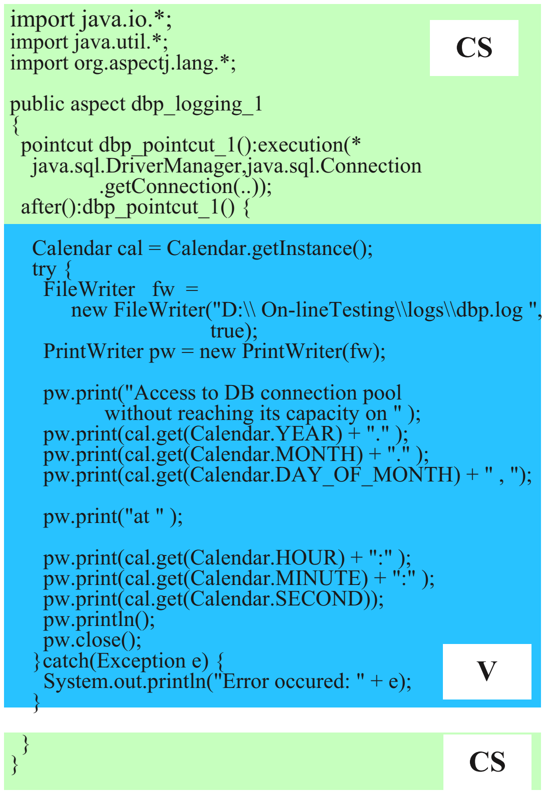
Figure 6: An Aspect Frame for online testing system

From the above AF in figure 6, Aspect Generator shown in figure 7 generates an aspect instance (AInst) that is specific to a selected AOP platform. The generated AInst of the AF in figure 6 is given in figure 8.



```
import java.io.*;
import java.util.*;
import org.aspectj.lang.*;

public aspect Logging_OnlineTesting 1 {

pointcut Logging_OnlineTesting1(): execution(* java.sql.DriverManager, java.sql.Connection.getConnection(..));
after(): Logging_OnlineTesting1()
{
 Calendar cal = Calendar.getInstance();
 try{
   FileWriter fw = new FileWriter("D:\\On-lineTesting\\logs\\dbp.log", true);
   PrintWriter pw = new PrintWriter(fw);


   pw.print("Access to DB connection pool without reaching its capacity on ");

   pw.print(cal.get(Calendar.YEAR) + "-");
   pw.print(cal.get(Calendar.MONTH) + "-");
   pw.print(cal.get(Calendar.DAY_OF_MONTH) + " ");

   pw.print("at");

   pw.print(cal.get(Calendar.HOUR) + ":");
   pw.print(cal.get(Calendar.MINUTE) + ":");
   pw.print(cal.get(Calendar.SECOND) + " ");
```

Figure 7: Generate aspects in Aspect Generator



```
import java.io.*;
import java.util.*;
import org.aspectj.lang.*;

public aspect dbp_logging_1
{
 pointcut dbp_pointcut_1():execution(*
   java.sql.DriverManager,java.sql.Connection
          .getConnection(..));
 after():dbp_pointcut_1() {

   Calendar cal = Calendar.getInstance();
   try {
    FileWriter   fw  =
        new FileWriter("D:\\ On-lineTesting\\logs\\dbp.log ",
                   true);
    PrintWriter pw = new PrintWriter(fw);

    pw.print("Access to DB connection pool
           without reaching its capacity on " );
    pw.print(cal.get(Calendar.YEAR) + "." );
    pw.print(cal.get(Calendar.MONTH) + "." );
    pw.print(cal.get(Calendar.DAY_OF_MONTH) + " , ");

    pw.print("at " );

    pw.print(cal.get(Calendar.HOUR) + ":" );
    pw.print(cal.get(Calendar.MINUTE) + ":" );
    pw.print(cal.get(Calendar.SECOND));
    pw.println();
    pw.close();
   }catch(Exception e) {
    System.out.println("Error occured: " + e);
   }

 }
}
```

Figure 8: An Aspect Instance for online testing system

The generated executable aspects are finally applied to the component by the aspect weaver. A new adapted version of the component is then created through aspect weaving. Since current AOP platform like AspectJ does not support complicated flow control such as switch in weaving process, post-processing is applied to enable process-based weaving in our framework.

The Aspect Weaver weaves the generated aspect instances into the original component according to the PCAS. The final adapted component source code is invisible to the developer. By deploying the adapted component, the targeted QoS requirements of system performance are fulfilled.

3.4 Conclusion

The new system with the adapted online testing component has been tested under various access loads. The results show that the system performance has been improved greatly, in particular while a large number of students using the system. The case study verifies that the GAIN approach is effective with good level of automation in improving QoS of components and component-based systems.

4. Related work

4.1 Binary Component Adaptation

Binary Component Adaptation (BCA) [10] has been proposed by R. Keller and U. Hölzle to support component adaptation in binary form and on-the-fly (during program loading). BCA rewrites component binaries before (or while) they are loaded, requires no source code access and guarantees release-to-release compatibility. That is, an adaptation is guaranteed to be compatible with a new binary release of the component as long as the new release itself is compatible with clients compiled using the earlier release.

However, together with the binary code adaptation, especially with “online” (on-the-fly) adaptations, extra processing time is required. As a result, the load-time overhead is a major problem. Consequently, when more adaptation processes are required, the load-time will be the bottleneck of the system performance.

4.2 Superimposition

Superimposition [2] is a novel black-box adaptation technique proposed by Bosch. Software developers are able to impose a number of predefined, but configurable types of functionality on reusable components. The notion of superimposition has been implemented in the Layered Object Model (LayOM), an extensible component object language model. The advantage of layers over traditional wrappers is that layers are transparent and provide reuse and customizability of adaptation behaviour. Due to lack of component information, modification is limited at simple level, such as conversion of parameters, and refinement of operations. Moreover, with more layers of code imposed on original code, the overhead of the adapted component increases heavily, which degrades system efficiency.

4.3 Aspectual Component

To achieve reusable aspects, Karl Lieberherr et al. introduced the concept of Aspectual Components [7]. Aspects are specified independently as a set of abstract join points. They believe that aspect-oriented programming means expressing each aspect separately, in terms of its own modular structure. Using this model, an aspect is described as a set of abstract join points which are used when an aspect is combined with the base-modules of a software system. In this way, the aspect-behaviour is kept separate from the core components, even at runtime.

4.4 Summary

Due to the diversity and level of component mismatch, available component adaptation approaches are either only capable for adaptation at simple levels, such as wrappers, or inefficient to use due the lack of automation in their adaptation process. Deep-level highly automated component adaptation can be achieved through AOP. Some AOP based frameworks have been developed to achieve reusable components. However, an AOP platform independent framework is still desired. Furthermore, current AOP techniques only support weaving aspects sequentially. To cope with complex adaptation, it often requires weaving aspects in more sophisticated control flow, e.g. dynamically deciding whether to invoke a particular aspect, and synchronizing in multi-thread applications.

5. Conclusions

Despite the success of component-based reuse, the mismatches in QoS between available pre-qualified components and the specific reuse context in individual applications continue to be a major factor hindering component reusability and smooth composition. The work presented in this paper is based on the observation that existing reuse approaches and tools are weak in providing a mechanism to adapt components in QoS aspect and meanwhile with sufficient automation. The aspect-oriented nature of our approach makes it particularly suitable for the improvement of non-functional features of the target component-based software, such as dependability and performance.

The proposed approach applies aspect-oriented generative adaptation to targeted components to correct the mismatch problem in QoS so that the components can be integrated into the target application smoothly. Automation and deep level adaptation are the benefits of the approach. This is achieved with the following key techniques in an aspect-oriented component adaptation framework: 1) the generation of adaptation aspects based on specific adaptation requirements; 2) the advanced aspect weaving process definition mechanism that supports switch and synchronization; 3) an expandable repository of reusable adaptation aspects.

The benefits of the approach include deeper adaptability, especially in non-functional aspects, higher automation and therefore smooth component integration and wider reusability. As consequence, the target component-based software will have better performance. Our case studies have shown that the approach and tool are promising in solving the mismatch problem in QoS.

6. References

[1] Batory, D., Johnson, C., MacDonald, B., & Heeder, D. V., "Achieving Extensibility Through Product-Lines and Domain-Specific Languages: A Case Study", *ACM Transactions on Software Engineering and Methodology (TOSEM)*, April 2002, Vol. 11(2), pp. 191-214.

[2] Bosch, J., "Superimposition: a component adaptation technique", *Information and Software Technology*, 1999, 41, 5 pp. 257-273.

[3] Diaz-Herrera, J.L., Knauber, P., & Succi, G. "Issues and Models in Software Product Lines," *International Journal on Software Engineering and Knowledge Engineering*, 2000, 10(4):527-539.

[4] Feng, Y., Liu, X., & Kerridge, J. "Achieving Smooth Component Integration with Generative Aspects and Component Adaptation", *Proceedings of 9th International Conference on Software Reuse*, Torino, Italy, June 11-15, 2006, LNCS 4039, pp. 260-272.

[5] Ingham, David B., Shrivastava, Santosh K., & Panzieri, Fabio, "Constructing dependable Web services", *IEEE Internet Computing*, 2000, 4(1), pp. 25-33.

[6] Jhumka, A., Hiller, M., & Suri, N, "Component-based synthesis of dependable embedded software", *Formal Techniques in Real-Time and Fault-Tolerant Systems. 7th International Symposium, FTRTFT 2002. Proceedings LNCS*, 2002, Vol.2469, pp 111-28

[7] Lieberherr, K., Lorenz, D., & Mezini, M, "Programming with Aspectual Components", *Technical Report, College of Computer and Information Science, Northeastern University, NU-CCS-99-01*, March, 1999.

[8] Liu, X., Feng, Y., & Kerridge, J. "Achieving Dependable Component-Based Systems Through Generative Aspect Oriented Component Adaptation", Proceedings of 30th Annual International Computer Software and Applications Conference, Chicago, September 18-21, 2006.

[9] Mezini, M., Ostermann, K., "A comparison of program generation with aspect-oriented programming", *Lecture Notes in Computer Science*, 2005, 3566, pp. 342-354.

[10] Keller, R., & Hölzle, U. "Binary Component Adaptation". *Proceedings of the 12th European Conference on Object-Oriented Programming*, July, 1998.

[11] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., & Griswold, W., "Getting Started with AspectJ", *Communications of the ACM*, October 2001, pp. 59-65.

[12] Sommerville, I. (2007). *Software Engineering (8th edition)*. Addison-Wesley. ISBN: 0-321-31379-8.

[13] Sullivan, G.T., "Aspect-oriented programming using reflection and meta object protocols - Providing programmers with the capability to modify the default behaviour of a programming language.", *Communications of the ACM*, OCT 2001, 44 (10), pp. 95-97.

[14] Viega, J., Voas, J., "Quality time - Can aspect-oriented programming lead to more reliable software?", *IEEE SOFTWARE*, Nov-Dec, 2000, 17(6), pp. 19-21.

Modular Specification of Aspect-oriented Systems and Aspect Conflicts Detection

Hui Liang
School of Computing
National University of Singapore
lianghui@comp.nus.edu.sg

Jing Sun
Department of Computer Science
The University of Auckland
j.sun@cs.auckland.ac.nz

Abstract

In this paper, we propose AspecTCOZ to provide a formal specification notation for aspect-oriented software design and analysis. Furthermore, we propose a formal specification-based approach for the early detection of semantic conflicts between aspects, which are caused by the superimposition of multiple aspects on the same join point.

1 Introduction

Aspect-oriented programming (AOP) [4] has been proposed as a new methodology to improve the separation of concerns in software development. By far, a few methodologies for aspect-oriented requirement analysis and architecture design have been proposed [1, 12, 13]. However, comprehensive technique supports for modeling and design are still far from sufficient.

In this paper, we propose AspecTCOZ, an aspect-orientated extension of the integrated formal notation TCOZ (Timed Communicating Object-Z) [7]. The class schema in TCOZ notation is an eligible candidate for specifying an aspect formally, because *aspect* is the unit of modularity, encapsulation, and abstraction in AOP, just in the same way as *class* in OOP. Meanwhile, the strength of TCSP in modeling process control and real-time interactions, which is preserved in TCOZ, provides a great mechanism for specifying the temporal order between pointcut and advice in an aspect-oriented context. Therefore, we extend TCOZ with the mechanisms for formally specifying the constructs of join point, pointcut, advice, and inter-type introduction, aiming to provide a formal specification notation for aspect-oriented software design and verification.

Meanwhile, in AOP, when multiple aspects are superimposed on the same join point, unexpected semantic conflicts between aspects may emerge. Therefore, we propose an approach for the early detection of semantic conflicts between aspects, based on the formal specification of the system written in the AspecTCOZ notation.

The rest of the paper is organized as follows. Section 2 introduces TCOZ notation. Section 3 presents AspecTCOZ notation. Section 4 presents a formal specification-based approach for the early detection of semantic conflicts between aspects. Section 5 reviews some related work. Finally, Section 6 concludes the paper.

2 TCOZ notation

Timed Communicating Object-Z (TCOZ) [7] is an integration and extension of the Object-Z [2, 10] and the Timed CSP [8, 9] formal modeling notation. The essence of this blending is the unification of Object-Z operation specification schemas with terminating CSP processes.

The basic structure of a TCOZ document is the same as that of an Object-Z document, consisting of a sequence of class definitions, and some type and constant definitions in the usual Z style. However, TCOZ varies remarkably from Object-Z in the definition of class schema. In TCOZ notation, operation schema expressions may appear wherever processes may appear in CSP and CSP process definitions may appear wherever operation definitions may appear in Object-Z. Furthermore, it is natural to allow to define TCOZ operations in terms of CSP primitives, such as event sequencing, as well as through the schema calculus. Thus, it becomes feasible to specify its temporal properties when describing the operation. Meanwhile, operation schemas take on the syntactic role of CSP processes, so they may be combined with other schemas and even CSP processes using the standard CSP process operators. A detailed introduction to TCOZ and its Timed CSP and Object-Z features may be found in [7]. The formal semantics of TCOZ is documented in [6].

3 AspecTCOZ - an aspect-oriented extension of TCOZ

Aspect is the central unit of modularity, encapsulation, and abstraction in AOP. It is defined very much like a class,

Table 1. Formal notation for join point model of AspectJ

Join point model	Formal notation
method call constructor call	ζ(list of *operation/ constructor signatures*)
method execution constructor execution handler execution advice execution object/class initialization	ξ(list of *operation/ advice/constructor/handler/ object initialization signature*)
field read access	Ⓡ(list of *class member names*)
field write access	Ⓢ(list of *class member names*)

Table 2. Formal notation for some pointcut designators of AspectJ

Pointcut designator	Formal notation
control-flow based pointcuts	$\blacktriangledown$(*executing/calling pointcuts*)
	$\triangledown$(*executing/calling pointcut*)
lexical-structure based pointcuts	$\bigcirc$(name of class)
	$\circledcirc$(name of operation)
execution object poincuts	$\oplus$(name of class/object)
	$\odot$(name of class/object)
argument pointcuts	$\mathcal{A}$(type/name of parameters)
conditional check pointcuts	predicates from TCOZ

and can contain methods, fields, nested class members, and initializers, just like a normal OOP class. Moreover, just like class inheritance in OOP, there is a mechanism for aspects inheritance with which aspects can not only extend other aspects, but also extend classes and implement interfaces in AOP. Therefore, the class schema in TCOZ notation might be an eligible mechanism for specifying aspects formally. However, as the basic units for implementing aspect-oriented crosscutting concerns, aspects must contain the constructs that express the weaving rules for both dynamic and static crosscutting, namely pointcuts, advice, and inter-type declarations and so on. There are no such mechanisms in TCOZ that can specify them competently. Therefore, we extend TCOZ notation with some constructs so that it will be capable of specifying an aspect-oriented system formally.

3.1 Join point

Join points are events in the control flow of a program; and they are identifiable point in the execution of a program. The join points defined by AspectJ include: method and constructor calls or executions, field accesses, object and class initializations, handler and advice executions, and so on [5]. In AspecTCOZ, we introduce formal notations, which is as shown in Table 1, for the join point model of AspectJ.

3.2 Pointcut

A pointcut is a program construct that captures a set of join points by matching certain characteristics. In addition to the join points which have been presented in Section 3.1, the pointcut designators in AspectJ can also capture join points based on matching the circumstances under which they occur, such as control flow, lexical scope, and conditional checks. Table 2 presents the proposed formal notation for pointcut designators.

Complex matching rules can be formed by combining simple pointcuts. AspecTCOZ provides a unary negation operator $\neg$ and two binary operators $\wedge$ and $\vee$ to build powerful pointcuts from the simple building blocks of existing and primitive pointcuts.

- unary negation operator $\neg$ allows the matching of all join points except those specified by the pointcuts.
- $\wedge$ and $\vee$ are provided to combine pointcuts. Combining two pointcuts with the $\vee$ operator causes the selection of join points that match either of the pointcuts, whereas combining them with $\wedge$ operator causes the selection of join points matching both the pointcuts.

To provide a sound formal notation, it is demanded that all the pointcuts have a name in AspecTCOZ. Thus, the syntax of a pointcut declaration is as follows:

$$\mathit{PointcutName}\ [(\mathit{ParameterTypeList})] \mathrel{\widehat=} \mathit{PrimitivePointcut}\ \{\neg \mid \wedge \mid \vee\ (\mathit{PrimitivePointcut} \mid \mathit{PointcutName})\}$$

Literally, the pointcut can have zero or a list of parameters; and it can be a primitive pointcut or the combination of existing pointcuts and/or primitive poincuts with operators (i.e. $\neg, \wedge, \vee$).

3.3 Advice

Advice is a method-like construct that expresses what to do at the join points captured by a pointcut. The operation schema in TCOZ work greatly in describing "what to do". However, it is not capable enough of formally specifying advice because each piece of advice must be associated with a pointcut in AOP. The implicit invocation of advice can happen *before*, *after* or *around* the join points matched by its pointcut. The strength of TCSP in modeling process control and real-time interactions, which is preserved in TCOZ, provides a great mechanism for specifying the temporal order between pointcut and advice.

Join points captured by the pointcuts are essentially events in the execution of a program. Therefore, we can define a process *PCprocess* for every pointcut *PointCut* as follows:

$$PCprocess = e : PointCut \rightarrow SKIP$$

Furthermore, in TCOZ notation, operation schemas is identified with CSP processes that perform only state update events; and operation schema expressions may appear wherever processes may appear in CSP. Therefore, assuming *OP* is the operation schema describing "what to do" with the advice and *PCprocess* is the process corresponding to the relevant pointcut, we can specify the *before advice* and *after advice* as the sequential composition of two processes, namely *PCprocess* and *OP*, as follows:

before advice $\Rightarrow$ *OP*; *PCprocess*
after advice $\Rightarrow$ *PCprocess*; *OP*

3.4 Inter-type declaration

Inter-type declarations are statements that an aspect takes complete responsibility for certain capabilities on behalf of the "targets" of the inter-type declarations. The most basic forms of inter-type declarations are for methods, fields, and constructor. The mechanism provided by AspecTCOZ for specifying inter-type declaration of fields, methods and constructors are similar to those mechanism for normal declarations, but with the exception that the targets of the declarations are attached with sign '$\propto$'. There are two purposes to do so: 1) to distinguish the inter-type declarations from the normal ones; 2) to show clearly what the target modules of the inter-type declarations are. The following two schemas are respectively the general form of inter-type declaration of state variables and operations in AspecTCOZ.

$\propto$
$\{NameOfVariable \propto TargetClassName : Type\}_1$
$[Predicates\ \ on\ \ Variables]$

$OpName \propto TargetClassName$
$[ListOfToBeChanged]$
$[NameOfVariable \propto TargetClassName : Type]$
$[Predicates\ \ on\ \ Variables]$

3.5 Aspect

Having proposed the extension to TCOZ notation for formally specifying join point, pointcut, advice, and inter-type introduction, now we can formally define an aspect in AspecTCOZ notation. The general form of an aspect schema is shown in Figure 1.

AspectName
[*Inheritance*]
[*LocalDefinition*]
[*StateSchema*]
[*InitialSchema*]
[*OperationSchema*]
[*OperationExpressionDefinition*]
[*PointcutDefinition*]
[*AdviceDefinition*]
[*IntertypeStateVariableSchema*]
[*IntertypeStateVariableInitialSchema*]
[*IntertypeOperationSchema*]

Figure 1. An aspect schema

4 Formal specification-based detection of semantic conflicts

As multiple aspects are superimposed on the same join point, the aspects might well interfere with each other in a potentially undesired manner. The potential semantic conflict is one of the critical and intrinsic issues in aspect-oriented software development. While those aspects are syntactically sound and can be compiled without any problems, this kind of conflicts exhibit themselves only when the composed application executes.

In AspecTCOZ, the aspect is defined by the association of operation schema and pointcut. In the operation schema, the input and output parameters are clearly laid out, and the variables/objects that will be changed by the operation are also explicitly laid out in the *ListOfToBeChanged*. Thus, based on the formal specification, we can figure out whether there might be any data-dependent conflicts between aspects. Assume that a system has been specified in AspecTCOZ notation, and that there are some join points which are superimposed by a few aspects. For each join point, if it is superimposed then for each pair of the aspects superimposing on it (A_1, A_2), for each advice a_1 which is an advice included in aspect A_1, we check whether there exists an advice a_2 in aspect A_2 such that advice a_2 is the same kind of advice as a_1 and there are variables which are included both in the input variable list of the operation schema associated with a_1 and in the output variable list or the *ListOfToBeChanged* of the operation schema associated with a_2. If yes, there will be conflicts between the two aspects A_1 and A_2. We also check whether there are common elements between the *ListOfToBeChanged* of the operation schema associated with a_1 and the *ListOfToBeChanged* of the operation schema associated with a_2. If yes, we declare that there will be conflicts between A_1 and A_2.

5 Related work

Some researchers have tried to extend mathematics and/or logic based formal specification notations to support aspect-oriented program design and verification. The work most close to ours is what have been proposed by Yu *et al.* [14, 15]. They proposed AspectZ [15], an aspect-oriented extension to Z. In a similar way, Yu *et al.* [14] introduced the concept of join point, pointcut, advice and aspect to Object-Z. The formal specification notation proposed by us, AspecTCOZ, is different from the work by Yu *et al.* in two main ways. Firstly, in AspecTCOZ, advice is defined with the assistance of operation schema, which provides the system designers with the ability to abstract and encapsulate the description of what to do at the join point, and the ability to reuse this formal description. Secondly, with the strength of TCSP in modeling process control and real-time interactions, which is preserved in TCOZ, the temporal order between pointcut and advice can be described clearly and concisely in AspecTCOZ notation.

The detection of semantic conflicts has received some attention from researchers. Durr *et al.* [3] proposed a detecting approach that defines the semantics of advice in terms of operations on a resource model.Tessier *et al.* [11] proposed a formal way to detect semantic conflicts between aspects based on extended UML class diagram model. In our approach, the conflicts are detected based on the analysis of data flow between the formal specifications of two pieces of advice.

6 Conclusions and future work

In this paper, we have proposed AspecTCOZ notation. It is an aspect-orientated extension to TCOZ with the mechanisms for formally specifying the constructs of join point, pointcut, advice, and inter-type introduction. Furthermore, based on the formal specification of the system, which is written in AspecTCOZ notation, we proposed an approach for early detection of the semantic conflicts between aspects. It detects the conflicts in the design and modelling phases of system development, therefore, makes it possible to reduce the development cost while promising a high quality software. To develop the mechanism for formally specifying *around*-advice is a part of our future work. And, we will also extend our conflicts detection technique with the analysis of control dependence based on available formal specifications, so that more implicit conflicts will be detected[1].

[1]An extended version of this paper can be found at http://www.comp.nus.edu.sg/∼lianghui/AspecTCOZ.pdf.

References

[1] E. Baniassad and S. Clarke. Theme: An approach for aspect-oriented analysis and design. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 158–167, 2004.

[2] R. Duke and G. Rose. *Formal Object Oriented Specification Using Object-Z.* Cornerstones of Computing. Macmillan, March 2000.

[3] P. Durr, L. Bergmans, and M. Aksit. Reasoning about semantic conflicts between aspects. In *Proceedings of the First Aspect, Dependencies, and Interactions Workshop*, pages 10–18, 2006.

[4] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings European Conference on Object-Oriented Programming*, pages 220–242, 1997.

[5] R. Laddad. *AspectJ in Action: Practical Aspect-Oriented Programming*. Manning Publications Co., Greenwich, CT, USA, 2003.

[6] B. Mahony and J. S. Dong. Overview of the semantics of TCOZ. In *IFM '99: Proceedings of the 1st International Conference on Integrated Formal Methods*, pages 66–85, 1999.

[7] B. Mahony and J. S. Dong. Timed Communicating Object Z. *IEEE Transactions on Software Engineering*, 26(2):150–177, Feb. 2000.

[8] S. Schneider and J. Davies. A brief history of Timed CSP. *Theoretical Computer Science*, 138(2):243–271, 1995.

[9] S. Schneider, J. Davies, D. M. Jackson, G. M. Reed, J. N. Reed, and A. W. Roscoe. Timed CSP: Theory and practice. In J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, pages 640–675. Springer-Verlag, 1992.

[10] G. Smith. *The Object-Z Specification Language*. Advances in Formal Methods. Kluwer Academic Publishers, 2000.

[11] F. Tessier, L. Badri, and M. Badri. A model-based detection of semantic conflicts between crosscutting concerns: Towards a formal approach. In *Proceedings of International Workshop on Aspect-Oriented Software Development*, 2004.

[12] D. X. Xu and K. E. Nygard. Threat-driven modeling and verification of secure software using aspect-oriented petri nets. *IEEE Transactions on Software Engineering*, 32(4):265–278, 2006.

[13] D. X. Xu and W. F. Xu. State-based incremental testing of aspect-oriented programs. In *Proceedings of the 5th International Conference on Aspect-Oriented Software Development*, pages 180–189, 2006.

[14] H. Yu, D. Liu, Z. Shao, and X. He. Modeling complex software systems using an aspect extension of Object-Z. In *Proceedings of 18th International Conference on Software Engineering and Knowledge Engineering, 2006*, pages 11–16, 2006.

[15] H. Yu, D. Liu, L. Yang, and X. He. Formal aspect-oriented modeling and analysis by AspectZ. In *Proceedings of 17th International Conference on Software Engineering and Knowledge Engineering, 2005*, pages 175–180, 2005.

Avoiding Bad Smells in Aspect-Oriented Software

Eduardo K. Piveta[1], Marcelo Hecht[1], Ana Moreira[2], Marcelo S. Pimenta[1],
João Araújo[2], Pedro Guerreiro[2], R. Tom Price[1]
[1] Inst. Informática, Univ. Federal do Rio Grande do Sul - Porto Alegre, Brazil
[2] Dept. Informática, FCT, Universidade Nova de Lisboa - 2829-516 Caparica, Portugal
{epiveta, mvhecht, mpimenta}@inf.ufrgs.br, {amm, ja, pg}@di.fct.unl.pt, tomprice@terra.com.br

Abstract

Different kinds of shortcomings can appear during software development. Some of them can be detected using static analysis and be removed using refactoring techniques. A more efficient strategy is to avoid them before they appear in the software. Recent work has been done on the identification of bad smells in the context of aspect-oriented software development. However, mechanisms to help the software engineer to avoid them are not sufficient. In this paper, based on a collection of bad smells that can occur in aspect-oriented software, we propose a set of guidelines to reduce the occurrence of bad smells in aspect-oriented software artefacts and show the benefits of using, through examples, the guidelines hereby described.

1. Introduction

Aspect-Oriented Software Development [3] is being used to improve the modularisation of crosscutting concerns. Nonetheless, the use of aspects can introduce shortcomings either particular to the use of aspects or similar to those found in object-orientation, such as pieces of code abandoned in a class or in an aspect, code duplication, and classes or aspects with too many or too few responsibilities [15]. These shortcomings, called bad smells [4], can decrease reuse throughout all stages of the development process and can be minimized by the identification of their symptoms and the removal of their causes.

There are bad smells that appear in object-oriented software, indicating refactoring opportunities for code extraction from objects to aspects [12], or bad smells for aspect-oriented software [15] and detection algorithms for the AspectJ [9] language. However, these authors do not deal with mechanisms to avoid the occurrence of these bad smells in aspect-oriented software.

Also, guidelines for software design and for programming style for aspect-oriented software have been proposed [1][5][2], mostly based on guidelines for object-oriented software. These guidelines help the process of designing software in terms of developing more readable and reusable aspects. However, they do not address all the issues raised by the bad smells that can appear in aspect-oriented software.

In this paper we describe additional aspect-oriented design guidelines specifically targeted to avoid the occurrence of bad smells in aspect-oriented software. For each guideline, we provide a general description, a motivation and examples. We use the AspectJ Design Patterns [6] and other source code samples [2][7] as examples to demonstrate the use of the proposed guidelines.

This paper is organized as follows. Section 2 describes bad smells that can occur in aspect-oriented software. Section 3 presents a set of guidelines to aid the software engineer in conceiving aspect-oriented software. In Section 4 we discuss the relation between bad smells and guidelines. Section 5 discusses related work and Section 6 includes some concluding remarks.

2. Background

Bad smells are shortcomings in software artefacts, detectable by a set of symptoms and that can be indicatives of potential improvements through refactoring. In this section, we briefly describe bad smells that can appear in aspect-oriented software. For more details, one can refer to our past work in [15] and [16] or refer to the work of Monteiro and Fernandes [12]. The following bad smells are taken into consideration in this paper:

Code Duplication. Duplication of code among methods, advices, fields and pointcut definitions can increase the problems that occur in maintenance activities. These duplications can occur as a result of copy and paste programming or as a result of miscommunication (when a developer independently writes code that is very similar to what exists elsewhere).

Anonymous Pointcut Definition. Using the pointcut definition predicate directly on the advice may reduce the advice legibility and hide the predicate intention. Also, a pointcut definition not encapsulated in a pointcut cannot be reused by other constructions.

Lazy Aspect. This situation, initially defined in [12], happens when an aspect has few responsibilities, and its elimination result in benefits in terms of maintenance. Sometimes, this responsibility reduction is related to previous refactoring or to unexpected changes in requirements (planned changes that never occurred, for instance). After one or more refactoring patterns are applied, some classes or aspects can lose responsibilities.

Divergent Changes. Another shortcoming occurs when the pointcut definitions of an aspect are almost identical, varying only in their modifiers or in small parts of their predicate. Every time a pointcut is modified, all the other pointcuts have go through the same modifications.

3. Avoiding the Occurrence of Bad Smells

In this section we define guidelines dealing with bad smells in aspect-oriented software. Each guideline is followed by a brief motivation and by an example chosen from a well-known set of aspect-oriented implementations of design patterns [6] and other cases from [2] and [7]. We show both constructions that use the guidelines described in this section and constructions that do not follow the guidelines.

3.1. Use abstract aspects

Guideline. Design toward abstract aspects, whose behaviour is defined completely by its advices, and its relationship with classes or other aspects is accomplished by specialization.

Motivation. The use of abstract aspects can help in developing more reusable aspects, by postponing implementation decision and leaving the definition of concrete pointcut definitions to the sub-aspects. Also, the behaviour defined in abstract aspects can be reused to different target applications. Each application can create sub-aspects that capture the specific points that will activate the aspect behaviour.

Example. In the *Observer* pattern [6] below there is a *ScreenObserver* aspect that extends the reusable abstract aspect *ObserverProtocol* (line 1) and defines that both roles (*Subject* and *Observer*) will be played by the *Screen* class (lines 2 and 3). It also defines when the subject state changes (line 4) and what should be done to update the observers (lines 5-7). This example complies with this guideline, defining an abstract aspect implementing the logic for the *Observer* pattern and leaving for the sub-aspects to bind the *Subject* and *Observer* roles and the changes in the *Subject* with the classes that will play these roles.

```
1  public aspect ScreenObserver extends ObserverProtocol{
2    declare parents: Screen implements Subject;
3    declare parents: Screen implements Observer;
4    pointcut subjectChange(Subject sub): call(void Screen.
         display(String)) && target(sub);
5    void updateObserver(Subject sub, Observer obs) {
6      ((Screen)obs).display("Updated");
7    }
8  }
```

In the *Decorator* pattern [6], the *BracketDecorator* (line 1) and the *StarDecorator* (line 5) aspects have a duplicated pointcut named *printCall* (lines 2 and 7). In the example below, the implementation does not follow the *use abstract aspects* guideline.

```
1  public aspect BracketDecorator {
2    protected pointcut printCall(String s):
3      call(public void ConcreteOutput.print(String)) &&
           args(s);
4  }
5  public aspect StarDecorator {
6    declare precedence: StarDecorator, BracketDecorator;
7    protected pointcut printCall(String s):
8      call(public void ConcreteOutput.print(String)) &&
           args(s);
9  }
```

This implementation can be improved by introducing a super-aspect containing the common pointcut, eliminating the duplication (in red).

3.2. Use named pointcuts

Guideline. Use named pointcuts to provide hot spots for extension and to use the terms of the problem domain in hand.

Motivation. The definition of named pointcuts allows the reuse of the predicate associated with this pointcut. Also, a new term is added to the vocabulary regarding the concern being encapsulated by the aspect. The use of names for pointcuts enables the designer to use these names as a terminology for the development of the system. It also allows that these pointcuts are subjected to future refinement, by defining concrete pointcuts on sub-aspects.

Example. In the *Factory Method* design pattern [6] there is an aspect that changes the behaviour of a *Factory Method*, using an around advice. With this approach it is possible to have the factories create different products depending on the aspects woven into the project. In the *AlternateLabelCreatorImplementation* (line 1), the developer defines a pointcut named *labelCreation* (line 2), used in the around advice. If a before or an after advice is needed, the developer can reuse the definition. This definition serves also to define composition of pointcut predicates, and can be used by other pointcuts in the aspect. Other examples of the application of this guideline may be seen in Kiczales et al [8].

```
1  public aspect AlternateLabelCreatorImplementation {
2     pointcut labelCreation(): execution(JComponent
          LabelCreator.createComponent());
3     JComponent around(): labelCreation(){
4       JLabel label = (JLabel) proceed();
5         label.setText("...␣alternate␣JLabel");
6         return label;
7       }
8   }
```

In the *Builder* design pattern [6], this guideline is not followed in an aspect named *CreatorImplementation*, as shown below. The pointcut predicate is defined directly in the declare construction (line 2) and therefore cannot be reused in other pointcuts.

```
1   public aspect CreatorImplementation {
2       declare error: (set(public String Creator+.
            representation)
3           || get(public String Creator+.representation)) &&
              ! (within(Creator+)
4           || within(CreatorImplementation)): "variable␣
              result␣is␣aspect␣protected...";
5   }
```

Other examples of unnamed pointcuts can be seen in the *QueueStateAspect*, that implements the state transitions for the *State* pattern, in the *SortingStrategy* aspect, implementing part of the *Strategy* pattern and in the *SingletonProtocol* aspect, that defines the general behaviour of the *Singleton* pattern [6].

3.3. Use semantic based pointcuts

Guideline. Avoid relying only on method or class names for pointcut composition. Instead, use annotation or inheritance mechanisms, to associate semantics to class members. You can use inheritance or interface implementation references in the pointcuts to provide clearly defined semantics.

Motivation. Sometimes, a naming convention is adopted during the development of a system. However, these conventions are not always obeyed, or they are inadequate when dealing with the representation of join points collections to be affected by an aspect. Naming mechanisms increase the coupling between the base system and aspects and naming conventions are not checked and therefore are not guaranteed to be followed [10].

Example. You can define semantic based pointcuts using inheritance, for example, as used in the *Singleton* design pattern [6]. In the *SingletonInstance* aspect the developer defines a pointcut named *protectionExclusions* (line 3) defining a predicate that is true every time an instance of *PrinterSubclass* (line 4) or one of its sub-classes is created. Another possibility is to associate predicates to interfaces, using the advantages of this construct.

```
1   public aspect SingletonInstance extends
        SingletonProtocol {
2       declare parents: Printer implements Singleton;
3       protected pointcut protectionExclusions():
4       call((PrinterSubclass+).new(..));
5   }
```

In the *Proxy* design pattern example [6], the *unsafeRequest* method (line 6) is directly mentioned in the *requests* pointcut predicate (line 2). As this example was done before annotation support in *AspectJ 5* it did not rely in the constructions regarding annotations. Using these constructs, the developer can define an annotation to define a method's security status (@*safety*, for example) in order to be available to the *requests* pointcut.

```
1   public aspect RequestBlocking extends ProxyProtocol {
2       protected pointcut requests():
3           call(* OutputImplementation.unsafeRequest(..));
4   }
5   public class OutputImplementation {
6       public void unsafeRequest(String s) {
7         System.out.println("[OutputImplementation.
              unsafeRequest()]:␣"+s);
8       }
9   }
```

3.4. Favour pointcut composition

Guideline. Every time that the predicate of a pointcut contains join points without a strong semantic relationship, favour specifying pointcuts as the combination of two or more distinct pointcuts, one for each well-defined set of join points.

Motivation. Sometimes, when defining a pointcut, the developer put together a set of heterogeneous join points in the same predicate. The developer should focus on grouping related join points in separated pointcuts and composing these pointcuts in order to achieve the desired combination.

Example. Alwis et al [2] employ an example that illustrates this principle, by separating the pointcut's predicate in different sets. Initially, there was a pointcut named *lowLevelDataOperations* (line 1) containing several method calls in its predicate. Operations related to an *ASCII* channel, to a binary channel, and to a list of commands are specified in a single pointcut definition, as seen below.

```
1   pointcut lowLevelDataOperations():
2         (target(AsciiDataChannel) && (call(String readLine
              (..)) || call(void writeLine(..))))
3       || (target(BinaryDataChannel) && (call(long read(..))
            || call(void write(..))))
4       || (target(ListCommand) && call(void writeFileInfo
            (..))) ;
```

The separation of these definitions improves the pointcut readability and allows the developer to reuse the new defined pointcuts. It is also easier to evolve the aspect, as each set of related method calls are defined in a separated pointcut. Consider the example below. The *lowLevelDataOperations* (line 1) is now composed by several pointcut definitions: *asciiDataOps*, *binaryDataOps* and *listCommandDataOps* (lines 2-4). Below, there is the code complying with this guideline.

```
1   pointcut lowLevelDataOperations() : asciiDataOps() ||
        binaryDataOps() || listCommandDataOps();
2   pointcut asciiDataOps(): ... ;
```

Pointcuts related to a single class (such as *asciiDataChannelDataOps*, line 1) can be moved to inside this class.

This would simplify the maintenance of both the class and the aspect. The disadvantage of this approach is the difficulty in the comprehension of the aspect just by its definition; it becomes necessary to find pointcuts in other classes of the system.

In the Facade pattern [6] implementation, the *FacadePolicyEnforcement* aspect defines a declare warning construction that uses a composite pointcut to describe that a warning should be raised every time a encapsulated method is called outside the facade (line 5). This helps to avoid developers to call the methods encapsulated by the facade directly.

```
1  public aspect FacadePolicyEnforcement {
2     pointcut encapsulatedMethods(): call(* (Decoration ||
            RegularScreen || StringTransformer).*(..));
3     pointcut facade(): within(OutputFacade);
4     declare warning: encapsulatedMethods() && !facade():
           "Calling␣encapsulated␣method␣directly";
5  }
```

Another example, the *SingletonProtocol* aspect [6] aims to compose pointcut predicates, but do not define a separated pointcut definition to a singleton construction (line 3). Extracting this piece of predicate helps to clarify the aspect.

```
1  public abstract aspect SingletonProtocol {
2     protected pointcut protectionExclusions();
3     Object around(): call((Singleton+).new(..)) && !
           protectionExclusions() {
                ...
4     }
5  }
```

If a new pointcut definition is created, a pointcut composition of *singletonCreation* and *protectionExclusions* can be used. In the next example, the use of the composite predicate is illustrated. Note the extracted pointcut *sigletonCreation* in line 2.

```
1  public abstract aspect SingletonProtocol {
2     pointcut singletonCreation(): call((Singleton+).new
           (..));
3     Object around(): singletonCreation() && !
           protectionExclusions() {...}
4  }
```

3.5. One concern per aspect

Guideline. Design aspects so that they provide functionality to only one concern of the application. If this aspect deals with more than one concern, try to divide it in two or more aspects or classes, maybe forming a generalization hierarchy.

Motivation. When an aspect handles more than one concern, it should be divided into smaller aspects, each responsible for a single concern. This often happens with advice with diverging purposes or with attributes and inter-type declarations without connection with the rest of the aspect.

Example. Consider a *Debug* aspect (part of an example named *Space War* - a spaceship and asteroids game [7]), that defines advices dealing with different concerns simultaneously. This aspect collects points regarding user interface modification (line 2), regarding changes in the registry contents (line 3), and regarding ship collisions (line 4), among other concerns omitted in the example. Although all of these features are related to system debugging, they could be divided in several aspects, each one with a different perspective on debugging.

```
1  aspect Debug {
2    after() returning (SWFrame frame): call(SWFrame+.new
          (..)){...}
3    after(Registry registry) returning : target(registry)
         && (call(void register(..)) || call(void
         unregister(..))){...}
4    after(Ship ship, SpaceObject obj) returning : call(
         void Ship.handleCollision(SpaceObject)) && target
         (ship) && args(obj){...}
5  }
```

Another possibility is to make the different extracted aspects inherit from the same super-class (or super-aspect), which could be the *Debug* aspect itself.

In the following example, we used a sub-aspect of *Debug* containing an advice responsible for manipulating the debugging of ship collisions (line 2).

```
1   aspect Collision extends Debug{
2     after(Ship ship, SpaceObject obj) returning: call(
           void Ship.handleCollision(SpaceObject)) &&
           target(ship) && args(obj){...}
3   }
```

Defining a *Collision* aspect enables the developer to separate the debugging responsibilities, focusing, in this case, only in the collision specific requirements. This separation also makes easier to reuse the *Debug* aspect, since it contains only the basic debugging functions.

4. Discussion

The use of these guidelines can avoid the occurrence of *anonymous pointcut definitions* and several occurrences of *large aspects*, *lazy aspects*, *code duplication* and *divergent changes* [15][16].

Anonymous pointcut definitions can be avoided using the following guidelines: *use named pointcuts* and *favour pointcut composition*. *Large aspects* and *lazy aspects* can be avoided using the *one concern per aspect* guideline. *Code duplications* can be diminished by using *abstract aspects* and *named pointcuts*. The *favour pointcut composition* guideline can be used to overcome occurrences of *divergent changes*.

For the first guideline (*use abstract aspects*), it is not necessary that an abstract aspect is created for every aspect in the application, just as not every pointcut have necessarily to be defined by an abstract aspect. Abstract aspects are the core of aspect reuse. They are heavily used in the aspect-oriented design patterns and they allow the definition of a reusable implementation of several design patterns. The use of this principle is exemplified by the design patterns described by Hannemann and Kiczales [6], by the access and authentication mechanisms in Vanhaute et al [18] and by the implementation of distribution and persistence components in Soares et al [17]. Vanhaute et al [18] states that the use of abstract pointcuts and inheritance between aspects helps a lot in the generalization required for implementing aspect-based frameworks. Some further examples of the usage of abstract aspects can be seen in the implementation of the following patterns: *Chain Of Responsibility*, *Command*, *Mediator*, *Observer*, *Flyweight*, *Memento* among others. Usually the user binds the roles in the patterns using aspect inheritance.

The use of *named pointcuts* can be found in several aspects in the pattern library [6]. They are used as a key resource to the definition of reusable aspects. Examples of aspects using named pointcuts are: *ChainOfResponsibilityProtocol*, *CommandProtocol*, *MediatorProtocol*, *Decorator* aspects and *ObserverProtocol*. Mahrenholz et al [11] remind furthermore that using named pointcuts allow one to create formal arguments (parameters) for the occurrences referred by the predicate defined in the pointcut. Alwis et al [2] assert that using relevant names for pointcuts may promote the reuse of the aspects. The names of the pointcuts should describe, at high level, which kinds of operations (or other more complex join points) fit the pointcuts context, instead of the working details of the operation.

The use of *semantic based pointcuts* provides mechanisms to use annotations when there is the need to group different structures in a set of classes or aspects. This grouping is performed in a way to consider semantic aspects above syntactic ones. Even the application of renaming refactoring patterns may cause pointcuts to affect a different set of join points.

The *favour pointcut composition* guideline contributes to the clarity of the specification, and at the same time allows pointcuts to be reused individually. Following this guideline may help avoiding bad smells with respect to *divergent changes* and *code duplication* in aspects [15].

The *one concern per aspect* guideline can be used when, in modelling for example, the concerns encapsulated by an aspect involve an inheritance relationship, this should be made explicit by moving duplicated members to a super-aspect. After applying a set of refactoring patterns, aspects with few responsibilities can appear. Aspects without sufficient responsibilities can be merged with another aspect or class. Empty aspects or aspects that does not contain advices, pointcuts or inter-type declarations can be converted to classes or merged with existent aspects.

5. Related Work

Hanenberg and Unland [5] describe a similar guideline than *use abstract aspects*, called *separated pointcut definition*. However, for them, every time an aspect is created, a super-aspect should be implemented containing all the required pointcut declarations. We believe that not *every* aspect should have an abstract super aspect and we propose using abstract aspects when they can be used to provide extension points for applications.

Alwis et al [2] presents code styles for the AspectJ programming language, stating that even small design decisions concerning the naming of pointcuts may have a significant impact on the readability and modularisation of an application's code. Our work differ from that in the sense that we complement the suggestions provided by Alwis et al by proposing a guideline to name all anonymous pointcut definitions.

Chavez and Lucena [1] describes a set of guidelines for aspect-oriented design, discussing how the use of aspects may contribute in achieving some of the principles sought by object-oriented design. Aspect-oriented software can help to promote the stability of classes, by allowing non-invasive modifications in their structure or behaviour using static or dynamic crosscutting constructs, such as: advices and inter-type declarations. We focus more on guidelines regarding the use of pointcuts and inheritance and on guidelines to avoid bad smells in aspect-oriented software

Koppen and Stoerzen [10] describe the fragile pointcut problem and provide a tool to detect differences in pointcut semantics. Their work focuses on finding differences on the points matched by aspects when the software evolves and help on preventing unwanted side effects when modifying code in the presence of aspects. Our work focuses on favouring the use of semantic based pointcuts as a way to prevent the occurrence of these side effects (instead of detecting them after the problem occurs).

Monteiro and Fernandes [13] [12] present a catalogue of refactoring patterns to help in aspect extraction from legacy object-oriented code and discuss bad smells that might appear in aspect-oriented software, including one bad smell that occurs only in aspects. In [14], they emphasize the importance of the development of both refactoring patterns and bad smells catalogues. In this work we use *Large Aspect* as one of the bad smells to be minimized when using the guidelines described here.

Piveta et al [16] define algorithms to automatically detect five types of bad smells that occur in aspect-oriented software development, more specifically those written using the AspectJ language and provide a prototype implementation to evaluate the detection algorithms in a case study. Even using the guidelines described in this paper, some bad smells can appear in the resulting software. Piveta et al work can be used as a complement to this paper, detecting eventually remaining bad smells.

6. Conclusions

In this paper we provide a set of guidelines to help on avoiding the occurrence of bad smells in aspect-oriented software. These guidelines are exemplified and discussed using a set of AspectJ examples from different sources.

The guidelines listed in this paper can assist software designers in presence of crosscutting concerns and complements the ones defined by Chavez and Lucena [1], Hanenberg and Unland [5] and Alwis et al [2], refining considerations regarding the problems found specifically when dealing with crosscutting concerns.

The developer should focus on avoiding that these bad smells appear in the first place. Following some basic design guidelines, several bad smells can be minimized. Although the guidelines discussed in this paper are expressed in examples of a specific language, they can be adapted to other aspect-oriented languages. Some guidelines can be used directly, whereas some of them are not available in a chosen language, for example. As the AspectJ model is the basis for several aspect languages, it can be seen as a good starting point to the definition of bad smells and design guidelines for a large set of aspect-oriented software.

7 Acknowledgments

This work has been partially supported by CNPq under grant No.140046/2006-2 for Eduardo K. Piveta and by Capes-Grices project grant No. 2051-05-2 - *Identification of Crosscutting Concerns and Refactoring in Aspect-Oriented Systems*. It is also funded by Project PROSUL-Proc. no. 490478/2006-9 - *Latin-America Research Network on Aspect-Oriented Software Development (Latin-AOSD)*.

References

[1] C. V. F. G. Chavez and C. J. P. d. Lucena. *Guidelines for Aspect-Oriented Design*. First Brazilian Workshop on Aspect Oriented Software Development. Brasilia, Brazil, 2004.

[2] B. De Alwis, S. Gudmundson, G. Smolyn, and G. Kiczales. Coding issues in AspectJ. In P. Tarr, L. Bergmans, M. Griss, and H. Ossher, editors, *Workshop on Advanced Separation of Concerns (OOPSLA 2000)*, Oct. 2000.

[3] T. Elrad, R. E. Filman, and A. Bader. Aspect-oriented programming. *Comm. ACM*, 44(10):29–32, Oct. 2001.

[4] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: improving the design of existing code*. Object Technology Series. Addison-Wesley, 2000.

[5] S. Hanenberg and R. Unland. Using and reusing aspects in AspectJ. In K. De Volder, M. Glandrup, S. Clarke, and R. Filman, editors, *Workshop on Advanced Separation of Concerns in Object-Oriented Systems (OOPSLA 2001)*, Oct. 2001.

[6] J. Hannemann and G. Kiczales. Design pattern implementation in Java and AspectJ. In *Proceedings of the 17th ACM conference on Object-oriented programming, systems, languages, and applications*, pages 161–173. ACM Press, 2002.

[7] E. Hilsdale and G. Kiczales. Aspect-oriented programming with AspectJ, 2001. Proceedings of the 16th Object Oriented Programming Systems Languages and Applications (OOPSLA'2001). *Tutorial*. Tampa Bay, FL, USA.

[8] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. Getting started with AspectJ. *Comm. ACM*, 44(10):59–65, Oct. 2001.

[9] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. In J. L. Knudsen, editor, *Proc. ECOOP 2001, LNCS 2072*, pages 327–353, Berlin, June 2001. Springer-Verlag.

[10] C. Koppen and M. Störzer. PCDiff: Attacking the fragile pointcut problem. In K. Gybels, S. Hanenberg, S. Herrmann, and J. Wloka, editors, *European Interactive Workshop on Aspects in Software (EIWAS)*, Sept. 2004.

[11] D. Mahrenholz, O. Spinczyk, and W. Schroder-Preikschat. Program instrumentation for debugging and monitoring with aspectc++. In *Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, page 0249. IEEE Press, 2002.

[12] M. Monteiro and J. Fernandes. Towards a catalog of aspect-oriented refactorings. In P. Tarr, editor, *Proc. 4rd Int' Conf. on Aspect-Oriented Software Development (AOSD-2005)*, pages 111–122. ACM Press, Mar. 2005.

[13] M. P. Monteiro and J. M. Fernandes. Object-to-aspect refactorings for feature extraction. In *Proceedings of the 3rd International Conference on Aspect-Oriented Software Development (AOSD'2004)*. ACM Press, March 2004.

[14] M. P. Monteiro and J. M. Fernandes. The search for aspect-oriented refactorings must go on. In T. Tourwé, A. Kellens, M. Ceccato, and D. Shepherd, editors, *Linking Aspect Technology and Evolution*, Mar. 2005.

[15] E. Piveta, M. Hecht, M. Pimenta, and R. T. Price. Bad smells em sistemas orientados a aspectos (in portuguese). *Brazilian Symposium on Software Engineering, SBES 2005, Uberlandia - Brasil*, 2005.

[16] E. Piveta, M. Hecht, M. Pimenta, and R. T. Price. Detecting bad smells in aspectj. *Journal of Universal Computer Science (a ser publicado - setembro), JUCS 2006*, 2006.

[17] S. Soares, E. Laureano, and P. Borba. Implementing distribution and persistence aspects with AspectJ. In *Proceedings of the 17th ACM conference on Object-oriented programming, systems, languages, and applications*, pages 174–190. ACM Press, 2002.

[18] B. Vanhaute, B. De Win, and B. De Decker. Building frameworks in AspectJ. In L. Bergmans, M. Glandrup, J. Brichau, and S. Clarke, editors, *Workshop on Advanced Separation of Concerns (ECOOP 2001)*, June 2001.

METRICS OF CREDIBILITY AND INTERACTION QUALITY: DESIGN AND EVALUATION

Nilda Pérez Otero, Marcelo Pérez Ibarra, Sandra Mendez, Adelina García, Ma. del Pilar Gálvez Díaz, Viviana E. Quincoces, Héctor Liberatori, Beatriz Fiorito, Cecilia María Lasserre
FI - UNJu - S.S. de Jujuy, Argentina - E-mail: lasserre@imagine.com.ar

ABSTRACT

Web Engineering is the process through which high quality WebApps are created. Although Web Engineering incorporates many of the concepts and fundamental principles of Software Engineering, the sites and WebApps have their own characteristics which are rather different from those of the traditional software. Like any other software product, the development of WebApps needs to include quality assurance activities. So that quality could be assured, evaluation and control of the intermediate products up to the final products must be planned. WebApps applied to distance education gave rise to what is called e-learning. Considering that a great deal of the proposals on education software evaluation are related to the quality or their need for expert evaluators due to its complexity, the Group of Software Engineering belonging to the Faculty of Engineering of the Universidad Nacional de Jujuy, Argentina developed a Metrics to determine the Credibility and Interaction Quality of Tele training (MECACIN). In this paper the development process followed for obtaining MECACIN based on the Model of Credibility and Interaction Quality of Tele training Courses is presented. Also, the design and reliability analysis process of MECACIN's questionnaire is related. The results of the experiment show that MECACIN's questionnaire has internal consistency.

KEY WORDS:

Quality, Metrics, e-learning, questionnaire reliability

I. INTRODUCTION

Web Engineering (IWeb) is the process through which high quality Web applications (WebApps) are created [1]. Although Web Engineering incorporates many of the concepts and fundamental principles of Software Engineering, the sites and Web Apps as product or product in use have their own characteristics which are rather different from those of the traditional software [2]. Based on these characteristics, a great variety of Web applications are identified: informative, downloading, personalizing, of interaction, user's input, transaction-oriented, service-oriented, portal, database access and data storing [3].

Software product quality is an issue that has been dealt with since several years ago but, in general, for each project different quality models are adopted. [4]. In 1991 a first edition of the quality standard ISO/IEC 9126 [5] was published, the idea was that it would become the reference point for all later developments. This standard, which is made up of four parts (ISO/IEC 9126-1, 2001; ISO/IEC 9126-2, 2003; ISO/IEC 9126-3, 2003; ISO/IEC DTR 9126-4) [6, 7, 8, 9], gives different models for evaluating the external quality, the internal quality and quality in use of a software product.

Just as any other software product, the development of WebApps needs to include quality assurance activities. So that quality be assured, evaluation and the control of the intermediate products up to the final products must be planned. ISO/IEC 9126, is used for the formulation of quality models for different Web applications [2, 10, 11]. WebApps applied to distance education gave rise to what is called e-learning. According to [12], e-learning is defined as the systematic use of multimedia technologies based on computers so as to upgrade students, improve learning, linking students with people and resources that would help to their needs, and to put together learning with performance and individual targets with organization targets.

Most of the works done on Web applications quality are oriented to electronic business or to some other types of sites such as magazines, television channels or entertainment sites; generally funded by large organizations. The issue of the evaluation of education learning software in particular, has been studied and documented by several authors related to education, giving evaluation measures on the subject [13, 14, 15, 16, 17, 18, 19, 20, 21, 22]. However, considering that a great deal of the proposals on education software evaluation are related to the quality or their need for expert evaluators due to its complexity, the Group of Software Engineering (GIS) belonging to the Faculty of Engineering of the Universidad Nacional de Jujuy, Argentina developed a Metrics to determine the Credibility and Interaction Quality of Tele training[1] (MECACIN), which is easy to use by people who have already finished high school, and who are not necessarily specialized in comput-

[1] Tele training is a system of distance training supported on Computing and Communication Technologies (technologies, telecommunication webs, video-conferences, digital TV, multimedia materials), which combines several pedagogical elements: classical teaching (attending the class or self-study), practice work, real time contacts (present, videoconferences or chats) and differed contacts (tutorials, forums, e-mail) (FUNDESCO, 1998).

ing nor tele training [23, 24, 25], but with knowledge on INTERNET surfing.

The goal of this work is to state the development process which the GIS followed for the MECACIN design and evaluation and the model it has. The work is structured as follows: in part 2 the Model of Credibility and Interaction Quality of Tele training Courses (MOCACIN) is presented which supports the metrics; in part 3 the design and refinement of the measuring instrument is presented; in part 4 the results of the evaluation of the Internal Consistency of the measuring instrument are presented; in part 5 the conclusions and finally in part 6 the references.

II. Model of Credibility and Interaction Quality of Tele training Courses

When working on the Articulation of high-school education with the university during the years 2004-2005 in the province of Jujuy, Argentina, a group of teachers belonging to high school education came up to the Universidad Nacional de Jujuy with the following situation: when they tried to include tele training courses in their curricula, they were not able to establish certain aspects such as: qualifications of the authors of the contents, veracity of the contents, updating of contents, learning sequence for the user, surfing easiness, correctness of the edition.

All these aspects would allow them to prepare a fast selection of courses of easy and accessible surfing and coming from a reliable source; so as to later analyze deeply if they are adequate or not to be included in the curricula. After this problem was presented, the University asked the GIS to solve the above situation.

In order to start with the formulation, the GIS gathered together information regarding the e-learning applications, distance education applications, quality standards in existence, publications on Web Engineering (e-commerce, entertainment, etc.) and formulation techniques and question evaluation.

The material which was studied allowed us to determine:

1. that e-learning applications, which were considered interesting by those teachers are classified within the Information WebApps category, since they only present a reading content with simple links and surfing [3],
2. that an e-learning course is a software product which can be applied to models of quality as established in the standard ISO/IEC9126, adapted to the IWeb process and
3. that the ISO/IEC 14598 [26] standard presents a set of steps oriented to quality evaluation.

The ISO/IEC9126 standard does not consider in its quality model the content category which is considered as essential for what is meant to be measured, therefore, following the steps given in the standard ISO/IEC14598, the GIS formulated a mixed model of quality (MOCACIN) which considers:

1. some categories and sub-categories of external quality which are defined in the ISO/IEC 9126 standard,
2. the content category which Olsina [27] and Fernandez Nodarse [11] included while working on the WebApps, which, in the context of this work, refer only to text, images (graphs, figures, photographs, etc) and audio, and
3. an accessibility subcategory which is considered essential within the specific type of WebApp to which we pointed (tele training courses).

Table 1 presents the categories and subcategories considered in the model or requirement's tree formulation. Below we define each of the named categories:

Functionality: the capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.

TABLE 1. Categories and subcategories of MOCACIN

Category	Subcategory
Functionality	Accuracy Suitability Task assurance
Usability	Understandability Operability
Efficiency	Time behavior Resource utilization Accessibility
Content	Information accuracy Information suitability Legal compliance

Usability: the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions.

Efficiency: the capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.

Content: the software product ability to provide exact, adequate, easy to access and legal information.

Once the categories and subcategories of the tree are considered, the following concepts to be applied are explained:

Information need: to determine in tele training courses the qualifications of the authors of the content, the veracity of the contents, the updating of the contents, the learning sequence for the user, easiness of surfing and correctness of the edition.

Calculable concept: credibility and interaction of tele training courses. By credibility quality meaning the possibility to accurately determine who obtains credits for the course, who is its responsible person and if the edition regulations were followed (spelling, grammar, speaking, lawfulness), and by interaction quality the easiness surfing in the course (access speed to the pages, indenting how to go back to the previous page, to the next, links with other pages) and understanding its structure (titles, targets, audience).

Metrics: direct metrics are used to determine if attributes are present or absent and indirect metrics to determine global quality.

For each subcategory of the requirement's tree the attributes are identified which will be measured during the quality evaluation of the courses of tele training.

Once the information need is defined, the computing concept is defined and the interest attributes are identified, a measuring instrument was chosen: a questionnaire with the following characteristics. Direct, pre-codified, of closed questions (1, 0 or X), multidimensional (five dimensions) and of simple marks/score or not balanced.

III. MEASURING INSTRUMENT

Based on the opinion of experts on the development of distance education courses, available standards and the guideline for the design of e-commerce made by Alexander and Tate [28], a questionnaire was designed having 71 questions, based on heuristics [24].

Two of the attributes corresponding to the subcategory Operability, category Usability and their associated heuristics are shown below (Table 2).

TABLE 2. Category-Subcategory-Attribute-Heuristics

		ATTRIBUTE	QUESTION
CATEGORY Usability	SUBCATEGORY Operability	Link Back and Forward	For hierarchy structured sites, is there an included link on the page that links to the page which is on the superior and/or inferior level of the hierarchy?
		Link to Site Map, Index or Table of Contents	Does the page include a link to a site map, index or table of contents?

A preliminary test and research was made on the initial questionnaire in order to identify the following: more adequate type of questions, if the questions are clear and correct, if the questions have an adequate length, if the answers have an appropriate categorization, if there are any psychological issues against some of the questions, if there is logic in the order, if the time for answering is accepted by the people being questioned.

The preliminary test was done giving the questionnaire to a group of experts on distance education and on semi-present education, and a survey on how questions were understood. This research established that some questions required, from the users, a knowledge of computing terminology. In addition, some members of the group pointed out that the interpretation of some of the questions was associated to personal appreciations (for example: Is it clear which are the topics included in the course?) or they evaluated more than one attribute (for example, if there are graphs, tables or charts, are they labeled and easy to read?).

This situation led us to reformulate the questionnaire and to modify the way in which questions were presented according to the problems given. Some footnotes were also added in order to make clear the meaning of the technical terms. Table 3 shows some examples of the problems and the modifications made to the initial questionnaire.

As a result of the previous reformulation we got a questionnaire made up of 60 questions. This questionnaire was tested in order to evaluate if it had the properties that can assure its measuring ability. Those properties are defined below and the reliability analysis is pointed out.

A. *Scale properties*

By *Scale* we mean any type of measuring instrument, being the questionnaire one of the most common. An instrument of this type is a set of questions which have a series of metrics properties, through the measuring we study and quantify those properties [29]. Every scale must have a correction method, both of the instructions for its application and also of the guideline for interpreting the marks/score. That is to say, of properties that assure its ability for measuring, which are grouped in: Reliability, Validity and Feasibility [29].

TABLE 3. Problems and reformulation of the initial questionnaire.

ORIGINAL QUESTION	REPORTED PROBLEM	REFORMULATED QUESTION/FOOTNOTE
If there are graphs, tables or charts, are they labeled and easy to read?	Evaluation of more than one attribute. Interpretation subject to personal appreciation	If there are images (graphs, figures, photographs, etc) do they have their corresponding epigraph? If there are images, can you see them clearly? If there are tables, do they have their corresponding epigraph If there are tables, can you read them clearly?
Is there a link included on the subordinated/secondary pages to the main page?	Computing term. Main Page	*Main page*: this is the page from which you can access to all the items of the course (bibliography, contents, activities, evaluation, etc) It can be preceded by a presentation page.
Is it clear which topics are included in the course?	Interpretation subject to personal appreciation	The subjects/themes included in the course, is it clearly indicated.

Reliability is the scale's ability for measuring in a consistent way, accurately and without mistake the characteristic that is to be measured. When the scale is applied to the same objects in two different situations, the same measurement must be obtained.

Validity is the instrument's ability for measuring what it says it measures and not other aspects.

Feasibility is the easiness that the instrument has to be applied in several situations and group of individuals.

B. *Reliability Analysis*

Reliability Analysis allows us to determine the extent in which the elements in the questionnaire are related among them, to obtain a global index of replication (internal consistency of the scale in its whole) and identify problematic elements which should be taken away from the scale [29]. Reliability in a measuring instrument is valued through Stability, Internal consistency and Reliability among the examiners [30].

- *Stability*: it is meant to evaluate up to what extent the marks obtained by an individual in a test are free of measuring mistakes caused by at random temporal fluctuations. They are recommended for measuring the Test-Retest Method and the Equivalent Shape Method [31].
- *Internal Consistency*: degree in which different parts or elements from the test measure the same variable. For this type of measurement the Equivalent Shape Method, Half Partition and Analysis of the Variance of Items are the ones that are proposed (Cronbach's Alfa Coefficient or Kuder Richardson 20).
- *Reliability Inter-Examiners*: it is meant to evaluate to what extent the measurement of a feature through an instrument is independent from the tester subjectivity. That is to say, that the people examined by a test have the same marks/scores in their executions, not considering who the examiner is. The suggested procedure for this type of evaluation is the Agreement among Judges Method.

This work only presents the analysis of reliability with regard to internal consistency

IV. Internal Consistency Evaluation of the Measurement Instrument

There are several models which allow us to get different reliability coefficients, the GIS used Cronbach's Alpha which is an internal consistency model based on the average inter-elements correlation. It assumes that the scale is made up by homogenous elements that measure the same characteristic and that the internal consistency of the scale can be evaluated through the existing correlating among all its elements. In addition, it assumes that a scale is reliable when the variation of the marks observed can be attributed to the existing differences among the individuals. The coefficient values range between 0 and 1, and it is considered that there is a good internal consistency when the alpha value is over 0.7 (in some special cases it is accepted a value below 0.6 [32]. This, as the other statistical data which are used in the reliability analysis process, assumes that the scale elements are combined by adding, that is to say that global scoring of the scale is obtained by adding the marks of its elements. In the case of multidimensional scales (questions are grouped by the way in which the questions measure a dimension and others a different dimension) the coefficient calculation of reliability is made for each of the dimensions. The Cronbach's Alpha coefficient was made with the SPSS software, version 12.

The execution of the reliability evaluation made to the questionnaire of 60 questions, regarding the consistency aspect reported a Cronbach's Alpha of 0.740 for a group of experts. When applying it to the group of teachers who started the research, the alpha was below 0.7. After analyzing the results, it was concluded that the problems found could be due to the questionnaire which included questions regarding the technical aspects of INTERNET which are unknown to the specified users (for example, site map, browser title, etc).

This led us to choose two work lines:

- Quality Model for Credibility and Interaction for users of tele training courses.
- Quality Model for Credibility and Interaction for content developers of tele training courses.

Working with the first line, a questionnaire of 41 questions was chosen, aiming to people with training equal to high school, and not necessarily specialized in computing nor tele training specifically, but who do know how to surf on INTERNET. This version had three evaluations in groups with the mentioned characteristics, improving the Cronbach's Alpha. These evaluations allowed a new adjustment to the questionnaire which ended up in a new one with 39 questions divided in 5 (five) dimensions (Table 4).

TABLE 4. Definite Questionnaire

N°	Type	Heuristic
1	D1	Is there an indication of the companies or organizations supporting/guaranteeing the course?
2	D1	How can you contact the organizations or companies supporting the course? (is there a phone number, a Web address, an official e-mail, or a post-address)?
3	D1	Are the names and surnames of the authors of the course indicated?
4	D1	Is there any indication regarding the background of the author/s of the course? Do you consider that is enough, is there at least indicated the background of one author?
5	D1	Are the authors genuine/suited2 for the subjects dealt with in the course? Is it enough that at least one of the authors be considered genuine/suited?
6	D1	Is the author protected by author's rights?
7	D1	Is there any way to contact the authors? (a phone number, a Web-address, an official e-mail or a post address). Is it considered enough that at least one of the contact ways to one of the authors is indicated?
8	D1	If there is a contact way established, is there any indication as when the answer will be received?
9	D1	Is there any indication that the course has been revised by an editor3?
10	D1	Is there any indication that the course has been revised by peers?
11	D2	Is the title the appropriate one?
12	D2	Are the targets of the course duly stated?
13	D3	Is there any indication regarding the audience?
14	D4	Is there any indication as to the order in which

[2] Genuine/suited: having the ability to do something, apt.

[3] Editor: in the sense of the person responsible for the edition (spelling, grammar, images …?

N°	TYPE	HEURISTIC
		the contents of the course must be learnt?
15	D5	Is the language used to express the contents of the course, concise4, correct and concrete (not ambiguous)?
16	D1	If there are images (graphs, figures, photographs, etc), do they have their corresponding epigraph5?
17	D1	If there are images, can they clearly be seen?
18	D1	If there are tables, do they have their corresponding epigraph?
19	D1	If there are tables, can they be read?
20	D1	If the course includes statistical data, are the dates when they were collected clearly indicated? If there is one data missing, this would mean for the answer to get a zero.
21	D1	Regarding the course contents: is there any spelling mistake? If there is just one mistake it is enough for giving a zero to the answer.
22	D1	Regarding the contents of the course: are they free from syntax mistakes? If there is any mistake this means a zero.
23	D1	Regarding the contents of the course, are they free from printing mistakes? If there is one mistake this means a zero.
24	D1	If the course has a speaking part, is it free from pronouncing mistakes? If there is one mistake this means a zero.
25	D1	If there is bibliography, are the names and surnames of the authors, the title of the work, publishing house, date of publishing and city where it was published indicated? (in case of cities having the same name you must put the country) ALL the data stated in the heuristic must be put for each one of the bibliographic entries.
26	D1	Is there any indication related to the date in which it was created or of its modification (or updating date of the course)? It is considered as enough when the year is stated.
27	D2	The title of the main page[6], is it concise?
28	D2	The title of each one of the related pages, does it correspond to the material developed in them?
29	D2	Related to the subordinated pages, do they have a link to the main page?
30	D2	If links are used to go to the previous or next page (Back and Forward), is the location kept in all the pages that are presented?
31	D2	If a linking bar and/or linking key is used, is it put on the same place on all the pages presented?
32	D2	If a linking bar and/or linking key is used, is it the same on all the pages presented?
33	D2	Do all the links have access to pages (or sections within a same page) to the ones to which they refer?
34	D2	The course: does it have an index or table of contents?
35	D2	Do all the pages have a link to the index or table of content? Is it considered enough at least one of the indicated alternatives?
36	D1	The back colors and sources (letters) used in the course, do they allow to read easily the contents?
37	D2	Does it keep the same back presentation for each type of activity proposed by the course? (theory, practice, exercises, evaluation, etc.)
38	D2	Is there any indication as to the characteristics of the equipment and programs which are necessary for the correct functioning of the course?
39	D2	Within the course, Does the unfolding of a page not take more than 20 (twenty) seconds? This time will not be taken into account when a link must be done to related subjects or sites.

[4] Concise: brief related to the way of expressing ideas

[5] Epigraph: label, explanation of an image, table, etc.

[6] Main page: The page from which you can have access to all the items of the course (bibliography, contents, activities, evaluation, etc.) It might be preceded by a presentation page.

The last version had two new tests. The first with a group of four persons and the second one with a group of eleven persons which the mentioned characteristics. Table 5 shows the results that were obtained.

TABLE 5. Results obtained with the questionnaire of 39 heuristics.

DIMENSION	QUANTITY OF HEURISTICS	α OF CRONBACH	
		G1	G2
D1	22	0,967	0,770
D2	14	0,820	0,562
D3	1	*	*
D4	1	*	*
D5	1	*	*
Global	39	0,925	0,779

* since it is a dimension with just one question, the Cronbach Alpha cannot be calculated.

V. CONCLUSIONS

The results shown on table 5 allow us to assure that the MECACIN's questionnaire has an internal consistency. This is because the global Cronbach's Alpha value in both experiences is over the threshold of 0.7, although one of the groups presents, in D2 dimension, an Alpha which is below 0.7. Since D3, D4 and D5 dimensions have just one heuristics it was impossible to apply a Cronbach's Alpha, yet these dimensions present, in both experiences, homogenous answers for all the examiners, which can assure that these three questions are well formulated.

Once the internal consistency is proved, the GIS will approach to the determination of the Stability and Reliability Inter Examiners so as to assure the reliability of the scale. Later we shall focus on the study of validity and feasibility. We can foresee that, based on the experiments that were made, this last aspect will be presented as positive.

REFERENCES

[1] Pressman, Roger S. (2005), Ingeniería del Software. Un enfoque práctico. Sexta Edición. McGraw Hill Interamericana. México.

[2] Covella, G. J. (2005). "Medición y Evaluación de Calidad en

Uso de Aplicaciones Web". Presentación de Tesis de Maestría en la Facultad de Informática de la UNLP. La Plata.

[3] Dart, S. (1999). "Contaning the Web Crisis Using Configuration Management". En Proc. First ICSE Workshop on Web Engineering, ACM. Los Ángeles. http://www.fistserv.macarthur.uws.edu.au/san/icse99-WebE/ICSE99-WebE-Proc/default.htm (22/02/07)

[4] Pfleeger, S. L. (2002). Ingeniería de Software. Teoría y Práctica. Primera Edición. Prentice Hall y Pearson Educación. São Paulo SP.

[5] ISO/IEC 9126 (1991). Information technology - Software product evaluation - Quality characteristics and guidelines for their use. Ginebra

[6] ISO/IEC 9126-1 (2001), Software Engineering – Product quality. Part 1: Quality Model, Secretaría General de ISO, Ginebra.

[7] ISO/IEC 9126-2 (2003). Software Engineering – Product quality. Part 2: External Metrics. Secretaría General de ISO. Ginebra.

[8] ISO/IEC 9126-3 (2003). Software Engineering – Product quality. Part 3: Internal Metrics. Secretaría General de ISO. Ginebra.

[9] ISO/IEC 9126-4 (2005). Software Engineering – Product Quality. Part 4: Quality In Use Metrics. Ginebra.

[10] Olsina, L. A. (2000). Metodología Cuantitativa para la Evaluación y Comparación de la Calidad de Sitios Web. Presentación de Tesis Doctoral en la Facultad de Ciencias Exactas de la UNLP. La Plata.

[11] Fernandez Nodarse, F.; N. Soubal y S. Lima Montenegro. (2002) "Experiencias en la Concepción de una metodología para el desarrollo y control de calidad de productos y servicios informáticos orientadas a la Educación a distancia y el Comercio electrónico en Internet" En Actas I Congreso Internacional de Tecnologías y Contenidos Multimediales en Ambientes Digitales.http://espejos.unesco.or.uy/simplac2002/ad.html.

[12] Goodyear, P. (2000). "e-Learning, knowledge work and working knowledge". IST2000 Event, e-Learning Futures session, Nice.

[13] Barroso, J.; J. L. Mendel; and J. Valdeverde. (1998). "Evaluación de los medios informáticos: una escala de evaluación para el software educativo". En Cebrián, M. et al. "Creación de materiales para la innovación con nuevas tecnologías": EDUTEC 97, 355-358. ICE Universidad. Málaga. http://www.ice.uma.es/edutec97/edu97-c3/2-3-08.htm (22/02/07).

[14] Del Moral, E. (1998). "El desarrollo de la creatividad y las nuevas herramientas tecnológicas". En Comunicación Educativa y Nuevas Tecnologías". pp 51-66. Praxis. Barcelona. http://www.eafit.edu.co/articulos/evaISE.htm (22/02/07)

[15] Galvis, A. (2000). "Evaluación de MECs por juicio de expertos", Capitulo 10 del libro: "Ingeniería de software educativo" 2da. reimpresión. Universidad de Los Andes. Colombia.

[16] Gómez, M. T. (1997). "Un ejemplo de evaluación de software educativo multimedia". En Cebrián, M. et al. "Creación de materiales para la innovación con nuevas tecnologías": EDUTEC97. ICE Universidad. Málaga. http://www.ice.uma.es/edutec97/edu97_c3/2-3-03.htm (22/02/07)

[17] González, M. (1999). "Evaluación de software educativo. Orientaciones para su uso" Proyecto Conexiones. Universidad de EAFIT.

[18] Gros, B. (Coord.); A. Bernardo; M. Lizano; C. Martínez; M. Panadés y I. Ruiz. (1997) "Diseños y programas educativos, pautas pedagógicas para la elaboración de software". Editorial Ariel S.A. Barcelona

[19] Marqués, P. (1998). "La evaluación de programas didácticos". En Comunicación y Pedagogía (149). pp 53-58. Barcelona. http://www.xtec.es/~pmarques/tecnoedu. htm (22/02/07)

[20] MVU, Michigan Virtual University. (2002). "Standard Quality on-line courses". http://ideos.mivu.org/standards (22/02/07).

[21] PEMGU. (1999). "Pedagogical evaluation methods & guidelines for multimedia applications" Partners: Epral (Portugal), Colegio Irabia (Spain) and Holbaek Technical College (Denmark), DEL (Denmark), the University of Cologne (Germany) and Olivetti (Italy). http://www.irabia.org/pemgu/ (22/02/07).

[22] Stephen, B. (1998). "Evaluating checklist. Evaluating training software". Lancaster University. http://www.keele.ac.uk/depts/cs/Stephen_Bostock/docs/evaluationch.ecklist2.html (22/02/07).

[23] Lasserre, C.M. y V.E. Quincoces. (2005). Quality in E-Learning: a Heuristic Evaluation. En Proceeding Simposio ASSE 2005 (34JAIIO). Rosario.

[24] Quincoces, V.E. y H. Liberatori. (2005). "Métrica para evaluación de Cursos de Tele training". Cuadernos 26, FHyCS, Suplemento. VIII Jornadas Regionales de Investigación en Humanidades y Ciencias Sociales. ISSN 0327-1471. San Salvador de Jujuy.

[25] Gálvez, M. P; H. Liberatori; V. Quincoces; A. García y C. Lasserre. (2006). "Evaluación de Sitios Educativos: Comparación de Heurísticas". En Investigaciones Docentes en Ingeniería. Facultad de Tecnología y Ciencias Aplicadas. Catamarca.

[26] ISO/IEC 14598-1 (1999). Information Technology – Software Product Evaluation. Part 1: General Overview. Secretaría General de ISO. Ginebra.

[27] Olsina, L. A.; G. Covella y G. Rossi. (2006). "Web Quality".·En Web Engineering. E. Mendes & N. Mosley (Eds). pp 109-142.. Lecture Notes in Computer Science of Springer, ISBN 3-540-28-28-196-7.

[28] Alexander, J. E. and M. A. Tate. (1999). WEB Wisdom. How to Evaluate and Create Information Quality on the WEB. Wolfgram Memorial Library Widener University. Laurence Erlbaum Associates Publishens. Lodon. Mahwah. New Jersey. Apéndice A y B.

[29] Pérez López, C. L. (2005). "Métodos estadísticos avanzados con SPSS". Editorial Paraninfo. Madrid.

[30] Tornimbeni, S.; E. Pérez; F. Olaz y A. Fernández. (2004). Introducción a los Tests Psicológicos. 3° Ed. Editorial Brujas. Argentina. APA (American Psychological Association). (1999). Standards for psychological and educational tests. Washington, D.C.

[31] Aron A. y E. N. Aron. (2001). Estadística para psicología. 1ª edición. Pearson Education. Buenos Aires.

Predicting Order of Likelihood of Defective Software Modules

Rattikorn Hewett*, Phongphun Kijsanayothin*, and Alta van der Merwe+
*Department of Computer Science, Texas Tech University
+School of Computing, University of South Africa
Rattikorn.Hewett@cs.ttu.edu, kphongph@gmail.com, vdmeraj@unisa.ac.za

Abstract

The ability to quickly identify defective software modules can help expedite development of dependable software. Much empirical research has focused on accurate prediction of defective modules from data of software previously developed under similar environments. While this is useful, time wasted on investigating wrong modules can be critical when dealing with extremely large and complex systems. Is it possible to rank predicted modules in order of their susceptibility to defectiveness? Unfortunately, the likelihood of defectiveness is neither entirely dependent on nor linear to the number of defects in software modules. This paper presents an algorithm for predicting if a newly developed software module is likely to be defective, and rank those predicted to be defective in order of their likelihood. We apply the algorithm to five benchmarked data sets of NASA software application projects. The experiments show highly competitive results to other well-established approaches giving an average of 85.3% accuracy.

1. Introduction

Software dependability demands high quality assurance, which requires rigorous and costly assessments, ranging from time-consuming manual inspections to automated formal verification [3, 4, 7, 11]. The ability to quickly identify defective software modules can help expedite development of dependable software. One way to quickly identify defective software modules is by automated construction of models that can accurately predict which of the software modules under development are defective.

Such predictive models are of great benefits in various software practices [1, 2, 4]. When time and resources are not adequate for complete testing of an entire system, software developers can use the resulting predictions to focus the testing on parts of the system that are likely to have defects. Many real-time software systems (e.g., robotics, mission control and planning) increasingly rely on dynamic code synthesis that can adapt code generation to satisfy runtime requirements and changing operating conditions. To ensure that the systems provide functionality and perform satisfactorily in real-time, quick assessments of software being developed in real-time become necessary. These assessments give feedbacks to improve software dependability by allowing modification of the mission objectives, defect masking at runtime or proactively real-time configuration of software [3]. Models that can predict defective modules of newly developed software based on dynamically measured metrics provide effective means for assessing dependability of real-time software. The term 'defective module' refers to those modules where the number of defects is at least one, while 'defects' refer to faults in software that require fixing.

Much empirical research has focused on techniques for obtaining accurate prediction of defective modules from data of software previously developed under similar environments [2–4, 7, 11]. These techniques are applied under the assumption that behaviors of software defects are contributed by the same underlying development factors. Thus, experience from defect or characteristic data of previously developed software can be used for predicting future defect behaviors of software produced under the same development or similar environments. While this has shown to be beneficial, it can be even more advantageous to know which module is more susceptible to defectiveness so that time wasted on investigating wrong modules is reduced. This can be particularly crucial for extremely large and complex software or real-time systems where time spent on fixing wrong modules can degrade the system performance. One remedy is to rank predicted modules in order of their susceptibility to defectiveness.

Unfortunately, the likelihood of defectiveness of each module does not entirely depend on the number of defects in the module alone [4]. Furthermore, modules with high number of defects are not necessarily likely to be more defective in later development stage than those with low number of defects as they may have had all defects tested out [4]. Similarly, modules with few defects found prerelease may end up being defective post release because of poor testing efforts. The likelihood of defectiveness is not linearly associated with number of defects in software and thus, ranking of the probability of defectiveness of software modules must account for multiple contributing factors. However, finding an appropriate scoring function for this ranking is challenging and the problem of predicting likelihood order of defectiveness of software modules has not been widely addressed.

To study the above issue, this paper presents an algorithm for (1) predicting whether a newly developed software module is likely to be defective, and (2) ranking those predicted to be defective in order of their likelihood. The algorithm is based on the Martingale boosting technique [12]. We apply the algorithm to five popular benchmarked data sets of NASA software that are publicly available [13].

Section 2 describes related work and Section 3 gives the details of application data. Section 4 presents our ranking algorithm, MDR (Multi-Dimensional Ranking) followed by empirical studies and results in Section 5. Section 6 gives concluding remarks.

2. Related work

There has been significant increase in empirical software engineering research that aims to construct models for accurate prediction of defective-prone software [1, 4, 9]. This is partly due to recent availability of public software modeling tools such as Weka [13], and software data repositories such as the PROMISE repository [13].

Many recent studies [2, 3, 7, 11] have used the same benchmarked data sets as this study. Most of them apply *existing* methods either to improve prediction accuracy [3, 7] or gain understanding about the data [2, 11]. For example, Guo et al. [7] demonstrated how random forrest, a well-known ensemble learning technique for accuracy improvement, can have the same impact on software data. Challagulla et al. [3] applied various machine learning and statistical techniques readily available in public tools and reported their evaluation of these techniques for prediction of defective modules. Boetticher [2] exploits *k*-nearest neighbor, a widely used clustering algorithm for partitioning testing data into those that are close to defective and non-defective training instances to enhance understanding of defective modules in different levels of grain sizes. Koru and Liu [11] show that the larger size of software modules yields better prediction accuracy.

Unlike most previous efforts, our study presents a new aspect of prediction of defective models that also produces ranking order of likelihood of defectiveness. Although our focus was not to evaluate natures of software metrics but to use them for building predictive models, while doing so, our results led to observations about data characteristics that are strongly correlated with quality of defective ranking prediction. Our finding that module sizes can influence prediction accuracy (details are not presented in this paper) is consistent with previous observations [1, 10].

Table 1. Summary of NASA software data sets.

Project	Application	#of modules	#of defect. modules	Lines of Codes	Code
CM1	Spacecraft instrum.	498	49	14,763	C
JM1	Real-time predictive ground systems	10,885	2,109	457,346	C
KC1	Storage manage. for rec. and proc. data	2,109	326	42,391	C++
KC2	Science data proc.	522	107	19,259	C++
PC1	Flight software for earth orb. satellite	1,109	77	25,924	C

The proposed MDR algorithm is most similar to the MartiRank algorithm employed in the ROAM system [6]. Both are based on the Martingale Boosting [12] but the MDR employs different heuristic evaluation functions and partitioning technique for constructing predictive models. Section 4 describes these features in more detail.

3. Software characteristics and data

This section gives a background of the software data under our study. We apply the MDR algorithm to data sets obtained from five NASA mission-critical software projects. The data are publicly available from the NASA's Metric Data Program. In particular, we use its popular version in the PROMISE repository [13], mainly for comparison purpose.

Table 1 summarizes partial characteristics, such as a number of software modules (or data instances), a number of defective modules and lines of codes. The projects are concerned with different functionalities. KC1 and KC2 deal with data processing, storage and management, whereas the rest involve space specific functions. JM1 is the largest project in terms of number of modules, number of defective modules, and a total counts of lines of codes.

Each data set contains one class attribute and 21 condition attributes representing different types of software characteristics including static, dynamic code attributes. Table 2 categorizes these attributes into five metric types: *McCabe*, *Derived Halstead*, *Line Count*, *Operator* (or *Basic Halstead*) and *Branch Count*. Halsted metrics measure program size and are useful for estimating programming efforts, while *McCabe* metrics measure program complexity based on structures of control flows. All except the class attribute have numeric values. The *defective?* class attribute has binary values (one for defective and zero for non-defective). Detailed definitions and discussions of these metrics can be found in [3, 7, 13].

Table 2. NASA software metrics and attributes.

Types	**Attribute Descriptions**
McCabe	1. Line count of code 2. Cyclomatic complexity 3. Essential complexity 4. Design complexity
Derived Halstead	5. Number of operators & operands 6. Volume 7. Program length 8. Program difficulty 9. Intelligent Count 10. Effort 11. Effort Estimate 12. Programming Time
Line Count	13. Lines of code 14. Lines of Comment 15. Blank Lines 16. Lines of code and comment
Operator	17. Unique operators 18. Unique operands 19. Total operators 20. Total operands
Branch Count	21. Total Branch Count
Target Class	22. Defective?

```
inputs  :  L, a list of defective status (i.e., class attribute
           values) of each of n training instances (of soft-
           ware modules) T, a number of boosting rounds
outputs :  M, a list of triplets of the form $(A,s,p)$, where
           A, s and p, respectively, represents an attribute,
           a sorting order type (e.g., Inc for increasing,
           Dec for decreasing) and a ratio of a partition
           size to n, for each partition in each boosting
           round.
Begin
M ← Empty;
For each boosting round $t \leftarrow 1$ to T do
    Partition L into t sub-lists: $L_1, \ldots, L_t$ such that each
        sub-list has (1) approximately the same number of
        defective modules, and (2) approximately the same
        ratio of the number of defective modules to its size;
    For each sub-list $L_i$ do
        $\alpha$ = a set of all condition attributes;
        $L_i^{A,s}$ = a list obtained by sorting $L_i$ in the same order
            as its corresponding A attribute values sorted in
            s ordering type, where $s \in \{Inc, Dec\}$;
        $min\{R(L_i^{A,s}) | A \in \alpha, s \in \{Inc, Dec\}\} = R(L_i^{A^*,s^*})$;
        If $R(L_i^{A^*,s^*}) < R(L_i)$ then
            $Add(A^*, s^*, |L_i|/n)$ to M;
            Sort instances in $L_i$ in the same order as in $L_i^{A^*,s^*}$
        else $Add(NULL, NULL, |L_i|/n)$ to M
    End For
End For
End
```

Figure 1. Basic steps of MDR.

4. The MDR algorithm

We present a novel ranking algorithm, MDR (Multi-Dimensional Ranking). MDR is a machine learning algorithm for constructing a model for predicting if a newly developed software module is likely to be defective, and ranking those predicted to be defective in order of their likelihood. The algorithm is based on the Martingale boosting technique [12]. Intuitively, boosting iteratively uses results from one learner to tune the training instances so that learning in the next boosting round can incrementally improve prediction accuracy. Figure 1 shows basic steps of MDR.

In each boosting round, MDR performs a greedy search to select appropriate attributes, corresponding types of sorting order (e.g., increasing or decreasing order) and partitions of training modules in order to improve ranking patterns of defective modules in a given training data set. The selection is biased towards the ideal ranking, where modules are ordered from those with highest susceptibility to defectiveness to the lowest. During the search, MDR uses a heuristic evaluation function, R as a rating function to evaluate how good the selected sorted list of values of the selected attribute in a selected partition is compared to the ideal ranking. As shown in Figure 1, the two partitioning conditions are required to make sure that the class attribute values (i.e., defective or non-defective modules) are well distributed in each partition for training.

We now define our heuristic to rate L, a list of defective status of each of n software modules in the training set, i.e., $L = [d_1, d_2, ..., d_n]$, where $d_i = 1$ if software module i is defective otherwise it is 0. Formally, the rating of L is defined as follows:

$$R(L) = \sum_{i=1}^{n} r_i, \text{ where } r_i = \begin{cases} 0 & \text{, if } d_i = 1 \\ \sum_{k=i+1}^{n} d_k/n & \text{, otherwise} \end{cases}$$

The rating is designed to be biased to ranking patterns that have defective modules on top (or leftmost of the list) and non-defective modules at the bottom. Ratings can range from zero to $d(n-d)/n$ where d is a total number of defective modules and smaller rating values are better. For example, consider lists $L_1 = [0,0,1,1]$, $L_2 = [0,1,0,1]$, and $L_3 = [1,1,0,0]$. It is clear that L_3 is most desirable since defective modules are all towards the top of the list and L_1 is least desirable. This is consistent with our heuristic values: $R(L_1) = 1$, $R(L_2) = 3/4$, and $R(L_3) = 0$. As expected, patterns in order of preference are L_3, L_2, and L_1, respectively.

The output model is a list of triplets of the form (A,s,p), where A, s and p represents an attribute, a sorting order type (e.g., *Inc* for increasing, *Dec* for decreasing) and a ratio of a partition size to a number of training instances, respectively. Figure 2 shows an example of the model generated by MDR on NASA project CM1 when a total number of boosting round is specified to be three. The list of triplets for each partition in each boosting round is shown. A_i represents attribute i, as shown in Table 2. To apply the model to a given set of testing data instances, the ranking of likelihood of defectiveness can be predicted by repeatedly sort-

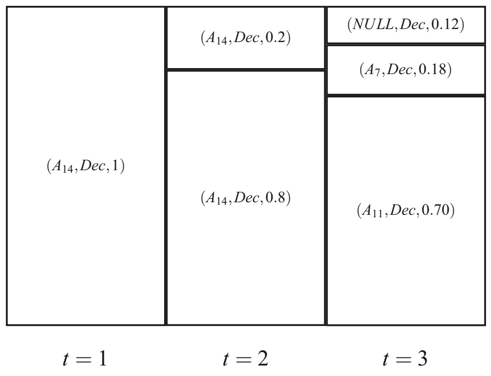


Figure 2. Example of a model produced by MDR.

ing the list according to the attribute selected by the model. For example, if we use the model in Figure 2, we will first sort the whole list according to attribute A_{14} (lines of comments) in decreasing order. Next (in round 2), we partition the list into two parts with the top part containing 20% of the total number of testing modules. We sort both top and bottom partitions according to attribute A_{14} in decreasing order. Finally, in round 3, we partition the list into three parts: the top and middle parts contain 12% and 18% of the total number of testing instances, respectively. The rest are in the last partition. In the top partition, NULL signifies that no attribute is selected and therefore there is no sorting required. Thus, this top partition remains unchanged. We then sort middle and bottom partitions in decreasing order according to attributes A_7 (program length) and A_{11} (effort estimate), respectively. The resulting list gives the ranking predictions.

Unlike most machine learners, MDR is specifically designed for producing ranking models. MDR differs from MartiRank in that it uses a different heuristic evaluation function as described. Furthermore, MDR records each partition boundary in terms of a ratio of a partition size to a total number of training instances, whereas MartiRank uses absolute boundary location. Consequently, MDR is more general than MartiRank in that we can apply MDR to a testing data set of any size whereas MartiRank can only be applied to those with the same size as the training data set.

We have implemented MDR in Java with capabilities to interface, at certain levels, with Weka [14]. This is to facilitate valid comparisons of results obtained from MDR and other machine learners provided by Weka.

5. Application and results

This section illustrates how MDR can effectively be used for constructing models for predicting and ranking defective models in order of their likelihood. We apply MDR to five data sets described in Section 3.

5.1. Evaluation metrics

Traditionally, most machine learners use *accuracy* (percentage of a ratio of a number of correct predictions to a number of testing instances) as a means to evaluate prediction quality. However, accuracy is not adequate for measuring ranking quality because it does not take mispredictions into account. Recent studies [2, 7] have employed metrics such as F-measure, and ROC (Receiver Operating Characteristics) curve [8] from signal detect theory [5]. However, like accuracy, these metrics do not have sufficient discriminating power for ranking evaluation.

Previous study has shown the *Area Under ROC Curve (AUC)* to be equivalent to the Wilcoxon statistic rank test [5] is a suitable means for evaluating ranking quality. To understand AUC, we first define the ROC curve. ROC curve is a plot between the *true positive rate (TP)* of the predictions on the Y-axis against the *false positive rate (FP)* of the predictions on the X-axis. *TP* is a ratio of positives correctly predicted to total positives, whereas *FP* is a ratio of negatives incorrectly predicted to total negatives [8]. *TP* and *FP* are also referred to as *sensitivity* and $1-$ *specificity*, respectively. In our context, positives refer to defective modules.

Given a list of predictions, a *cutoff* provides a boundary of accumulated instances tested so far to allow counting for defective modules correctly predicted and non-defective modules incorrectly predicted for determining *TP* and *FP*. Thus, each cutoff yields a single point on ROC curve (and one accuracy value). As cutoff incrementally moves down the list, we can obtain points on ROC curve toward the right side of the ROC space. Thus, ROC curve can be used to effectively evaluate (ranking) predictions. Unfortunately, comparing two learners based on their ROC curves is not always conclusive. This issue can be resolved by AUC, which represents each ROC curve with a single value (area under its curve). AUC can be interpreted as the probability that a randomly chosen example of defective module will have a higher estimated probability of being predicted as defective than a randomly chosen example of non-defective module. Thus, predictions with higher AUC are better. AUC has been shown theoretically and empirically to be a better measure for evaluating machine learners than accuracy. See more details in [8].

5.2. Methods

For efficiency control, MDR was designed so that a user can specify a number of boosting rounds. Although we have not proved this, we conjecture that effects of the number of boosting rounds (T) to the model accuracy will be similar to the effects of size of a training set. In other words, as T gets larger, the accuracy increases up until a certain point where accuracy will reach a plateau. This optimal accuracy may be reached at certain level of T (analogous to reaching a certain size of the training set). For practical purpose of our study here, we use $T = 8$ as applied in [6]. Further study is required to create a method for selecting an appropriate T.

To avoid overfitting in model construction, *n-fold cross-validation* [10, 14], a standard re-sampling accuracy estimation technique, is used. In n-fold cross-validation, a data set is randomly partitioned into n approximately equally sized subsets (or folds or tests). The learning algorithm is executed n times; each time it is trained on the data that is outside one of the subsets and the generated model (classifier) is tested on that subset. The estimated accuracy for each cross-validation test is a random variable that depends on the random partitioning of the data. The estimated accuracy is computed as the average accuracy over the n test sets. The n-fold cross-validations are typically repeated several times to assure data randomness and the estimated accuracy is an average over these n-fold cross-validations. For the experiments in this paper, we use AUC instead of accuracy and $n = 10$ as suggested in the study by Kohavi [10] to be theoretically acceptable.

We ran 10-fold cross-validations on all five NASA software data sets using our MDR ranking algorithm and four other machine learning techniques, which are available on Weka [14]. These four machine learners are *ZeroR*, *J48*, *Bayes* and *NN*. ZeroR is a majority learner, which is commonly used to measure a baseline performance in machine learning. J48 is a decision tree learner, Bayes is a well-known Naive Bayes classifier, and NN is a neural net approach using back propagation learning algorithm [14]. The last three learners are selected for comparison study because they cover a variety of techniques that use different representational models, namely decision tree models for J48, Bayesian probabilistic models for Bayes, and neural network for NN.

5.3. Results

Figure 3 shows a comparison of ROC curves obtained from ZeroR, J48, Bayes, NN and MDR for project CM1. As expected, ZeroR shows a baseline performance equivalent to random guessing. Therefore, it yields a no-discrimination straight line of 45 degrees from the horizontal [8]. The ROC curve obtained from J48 gives a better performance than random guessing. Majority of the ROC curve obtained from Bayes dominates that of J48. It is clear that both NN and MDR dominate the rest but there is no way to tell exactly, which of these two performs better by just observing their corresponding ROC curves alone.

Table 3 shows comparisons of percentage averages of AUCs, over 10-fold cross-validations, which are obtained from various learners and MDR. For each project, Table 3 shows the highest average AUC in bold. MDR performs

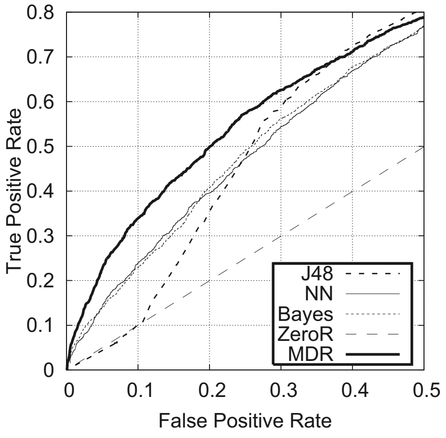


Figure 3. Comparison of ROC curves for CM1.

best on three of the five projects, namely CM1, JM1 and PC1. In these projects, the differences between the average AUCs obtained from MDR and the next best results range from close to 2% to 4%. Bayes performs best on KC1 and KC2. However, MDR has the highest mean of average AUCs over all five projects. Furthermore, average AUCs obtained from MDR are only 1.38% and 1.13% less than Bayes' result in KC1 and KC2, respectively. In spite of all these promising results, since the differences may not be statistically significant in each project, we conclude that MDR is highly competitive to other learners. In addition, recall that MDR provides additional information of ranking predictions of susceptibility to defectiveness, whereas others do not. Thus, MDR is more informative.

Table 3. AUC comparisons.

Project	ZeroR	J48	Bayes	NN	MDR
CM1	48.98	55.83	65.82	73.36	**75.12**
JM1	49.94	65.30	67.93	68.95	**71.97**
KC1	49.57	68.91	**78.99**	77.07	77.61
KC2	48.71	70.42	**83.43**	82.76	82.20
PC1	48.56	66.80	64.97	72.26	**76.25**
Avg.	49.15	65.45	72.23	74.88	**76.63**

It is quite remarkable to see that J48 performs rather poorly given a common knowledge that J48 along with other decision tree learners are among top performers when evaluated by accuracy. To see the magnitude difference between the two metrics, we compare average accuracy with average AUCs obtained from different learners on the same project. On project CM1, an average accuracy obtained from ZeroR, J48, Bayes and NN is 90.16%, 88.96%, 85.54% and 89.56%, respectively. These are a lot more optimistic than the average AUCs obtained from corresponding learners (as shown in the first row of Table 3). For project CM1, accuracy measures (in percentages) are about 16 to 41 higher than AUC measures (in percentages). In fact, if we apply a 5% cutoff to our ranking predictions from MDR, an average accuracy over five NASA projects obtained is 85.25%, whereas the average AUC (as shown in Table 3) is only 76.63%. Thus, using AUC as an evaluation metric can be more discriminating [7] but less optimistic than accuracy.

6. Conclusions

There has been significant amount of research in applying machine learners to predictions of defective software. What distinguish this work from previous efforts are the followings.

Firstly, we propose a different aspect of defective prediction by presenting an algorithm for ranking predicted defective modules in order of their likelihood. This gives stronger and more useful results than prediction of defective modules alone. Even though most machine learners provide some certainty measures of the predictions, obtaining ranking from them would require difficult ad-hoc post-processing, as they are not specifically designed for ranking purpose [6].

Secondly, we apply the proposed algorithm using AUC, an evaluation metric that is different from most previous studies in software defective prediction, particularly for the set of data under our study. AUC has been shown theoretically and empirically to be better than accuracy [8]. However, it has not been used widely in evaluating software defective prediction. Although previous work has employed related evaluation metrics such as probabilities of detections and false alarm, they are hard to use for comparison. AUC is superior to these metrics and ROC that does not provide a single measurement for easy comparison. Using AUC instead of accuracy to evaluate and interpret results of software data analysis obtained by machine learners could change previous findings about software characteristics and best prediction techniques to use in practice.

References

[1] R. M. Bell. Predicting the location and number of faults in large software systems. *IEEE Trans. Softw. Eng.*, 31(4):340–355, 2005. Member-Thomas J. Ostrand and Fellow-Elaine J. Weyuker.

[2] G. D. Boetticher. Nearest neighbor sampling for better defect prediction. In *PROMISE '05: Proceedings of the 2005 workshop on Predictor models in software engineering*, pages 1–6, New York, NY, USA, 2005. ACM Press.

[3] V. U. B. Challagulla, F. B. Bastani, I.-L. Yen, and R. A. Paul. Empirical assessment of machine learning based software defect prediction techniques. In *WORDS '05: Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 263–270, Washington, DC, USA, 2005. IEEE Computer Society.

[4] N. E. Fenton and N. Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Trans. Softw. Eng.*, 26(8):797–814, 2000.

[5] D. M. Green and J. A. Swets. *Signal Detection Theory and Psychophysics*. Wiley, New York, 1966.

[6] P. Gross, A. Boulanger, M. Arias, D. L. Waltz, P. M. Long, C. Lawson, R. Anderson, M. Koenig, M. Mastrocinque, W. Fairechio, J. A. Johnson, S. Lee, F. Doherty, and A. Kressner. Predicting electricity distribution feeder failures using machine learning susceptibility analysis. In *AAAI*, 2006.

[7] L. Guo, Y. Ma, B. Cukic, and H. Singh. Robust prediction of fault-proneness by random forests. In *ISSRE '04: Proceedings of the 15th International Symposium on Software Reliability Engineering (ISSRE'04)*, pages 417–428, Washington, DC, USA, 2004. IEEE Computer Society.

[8] J. Huang and C. X. Ling. Using auc and accuracy in evaluating learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 17(3):299–310, 2005.

[9] T. Khoshgoftaar and E. Allen. Predicting the order of fault-prone modules in legacy software. In *ISSRE '98: Proceedings of the The Ninth International Symposium on Software Reliability Engineering*, pages 344–353, Washington, DC, USA, 1998. IEEE Computer Society.

[10] R. Kohavi. The power of decision tables. In *ECML '95: Proceedings of the 8th European Conference on Machine Learning*, pages 174–189, London, UK, 1995. Springer-Verlag.

[11] A. G. Koru and H. Liu. An investigation of the effect of module size on defect prediction using static measures. In *PROMISE '05: Proceedings of the 2005 workshop on Predictor models in software engineering*, pages 1–5, New York, NY, USA, 2005. ACM Press.

[12] P. M. Long and R. A. Servedio. Martingale boosting. In *COLT*, pages 79–94, 2005.

[13] S. J. Shirabad and T. J. Menzies. The promise repository of software engineering databases. School of Information Technology and Engineering, University of Ottawa, Canada, 2005.

[14] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, October 1999.

Automated Test Code Generation from UML Protocol State Machines

Dianxiang Xu, Weifeng Xu

Department of Computer Science
North Dakota State University
Fargo, ND 58105, USA
{dianxiang.xu, weifeng.xu }@ndsu.edu

W. Eric Wong

Department of Computer Science
University of Texas at Dallas
Richardson, TX 75803, USA
ewong@utdallas.edu

Abstract

This paper presents a framework for automated generation of executable test code from UML 2.0 protocol state machines. It supports several coverage criteria for state models, including state coverage, transition coverage, and basic and extended round-trip coverage. It transforms the state invariants and transition postconditions of a state model into executable assertions to be verified against the actual object states by runtime code instrumentation. Hand-crafted test data are reused from one development version to the next due to the change of requirements. This reduces the working load for test regeneration of modified models. Our framework also reports the complexity of generated test suites, which can facilitate empirical evaluation of different coverage criteria for state models.

Keywords: Software testing, finite state machines, UML, object-oriented programming, coverage criterion.

1. Introduction

Finite state machines (e.g., UML Statecharts [10]) are widely used to document the design of object-oriented systems. Test generation from the state models of object behaviors has gained much attention in the past decade. Several coverage criteria (e.g., state coverage and transition coverage) have been proposed for state-based testing. Existing testing methods, however, often use them to measure how much of the state model is covered by a given test suite, rather than *automatically generating executable test code for the criteria*.

In general, model-based testing requires some level of human intervention in order to produce executable tests. A major problem is the extent to which the manual work can be reused from one development version to the next. This is similar to regression testing [9], which involves three issues: (1) selecting from the current test suite those tests that remain valid for the modified program; (2) removing obsolete tests from the current test suite; (3) identifying additional tests. Issues (1) and (2) together are also called the regression test selection problem, whereas issue (3) is called the coverage identification problem [9][11]. Although each of them is significant, code-based regression testing is by and large limited to the test selection problem. For model-based test generation, all tests are newly generated for the modified models. New (obsolete) tests are added (excluded) automatically. As such, hand-crafted test data need to be carried from one development version to the next.

This paper presents a framework for automated generation of executable test code from UML 2.0 protocol state machines. It is fully implemented in the MACT (Model-based Aspect/Class Testing) toolkit. MACT first generates a transition tree from a state model for the chosen coverage criterion. The tester can edit detailed test parameters if necessary. When the state model is modified due to requirements change, the hand-crafted test parameters are automatically reused. Once the required parameters are provided, MACT can generate executable test code, including test methods and state wrapper classes. The state wrapper classes provide a mapping from the state invariants in a state model to the concrete object states. They are used by runtime code instrumentation for verifying actual object states against expected states.

The rest of this paper is organized as follows. Section 2 describes how UML protocol state machines are used for class modeling and provides an overview of the automated testing process. Section 3 presents test generation from state machines. Section 4 discusses automated reuse of test data for modified models. Section 5 presents test metrics and our experiments. Section 6 reviews related work. Section 7 concludes the paper.

2. MACT: An Automated Test Framework

2.1 UML Protocol State Machines for Class Modeling

In UML 2.0, a protocol state machine specifies which operations can be called in which state and under which conditions, thus specifying the allowed call sequences on the operations. A main difference between protocol state machines and Statecharts is that transitions in a protocol state machine are associated with a precondition (guard) and postcondition, but not actions. As a blackbox testing strategy, model-based testing is concerned with the effect of the transition, rather than the procedural process. Therefore, protocol state machines provide an appropriate level of abstraction for test generation from state models.

We exploit protocol state machines to capture intra-object behaviors and inter-object effects. A protocol state model *M* consists of states *S*, events *E*, and transitions *T*. Transition $(s_i, e[p, q], s_j) \in T$ (precondition *p* and postcondition *q* are optional) means that, when event (method) $e \in E$ is triggered in the state $s_i \in S$, when p holds, then the state $s_j \in S$ must be reached under *q*. For a class state model, *S, E,* and *T* represent the possible states of objects, public constructors/methods, and functionality implemented by the constructors/methods, respectively.

A state $s \in S$ can be a concrete object state or a state invariant. For example, the state *OPEN* of a *BankAccount* object may refer to the following: `getClosed()==false && getBalance() >= 0 && getFrozen()==false`. Such states are specified in a state model for the purposes of test generation. They imply a link between the SUT and the generation of executable test code. The use of state invariants provides a high level of abstraction of object behaviors. A pre- or post-condition is a logical formula constructed by using constants, instance variables, and functions. A transition $(s_i, e[,q], s_j)$ without precondition means that the transition is unconditional: event *e* under state s_i always results in state s_j and *q* (if it exists). For convenience, we use α to denote the state before an object is created (as in [1]) and the *new* event to represent the constructor. Usually, a class model includes α in *S* and *new* in *E.* The object creation transition, $(\alpha, new[p, q], s_0) \in T$, if condition *p* holds, constructs an object with the initial state s_0 and achieves the postcondition *q*. Figure 1 shows the state model of a *BankAccount* class, where, for simplicity, *b* denotes *getBalance()*. The model consists of three state invariants: *OPEN*, *FROZEN* and *CLOSED*. The events are *deposit*, *withdraw, getBalance, close*, *freeze*, and *unfreeze*.

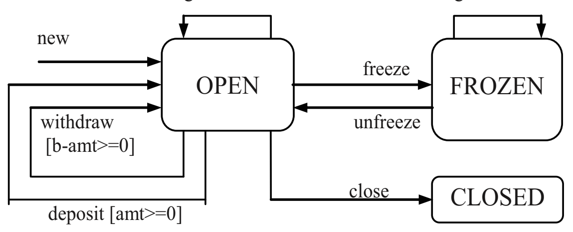


Figure 1. The state model of a *BankAccount* class

A test sequence is a sequence of transitions $(\alpha, new[p_0, q_0], s_0), (s_o, e_1[p_1, q_1], s_1),\ldots,(s_{n-1}, e_n[p_n, q_n], s_n)$. It starts with object creation, invokes methods on the object, and leads the object and other interacting objects to the respective states. Such a test sequence exercises not only individual constructors and methods, but also interactions between them.

2.2 Automated Testing Process

The automated testing process is shown in Figure 2. It starts with the tester building the state models for the classes under test and selecting a coverage criterion for test generation. The supported coverage criteria include state coverage, transition coverage, basic round trip, and extended round trip. For a given state model and coverage criterion, MACT generates a transition tree: the root represents the α state; each non-root node represents the resultant state and postcondition of a transition from the state in the parent node. As such, each path from the root to a leaf is a test sequence as described before. Section 3 will elaborate on the coverage criteria and the algorithms for generating tests that achieve the criteria.

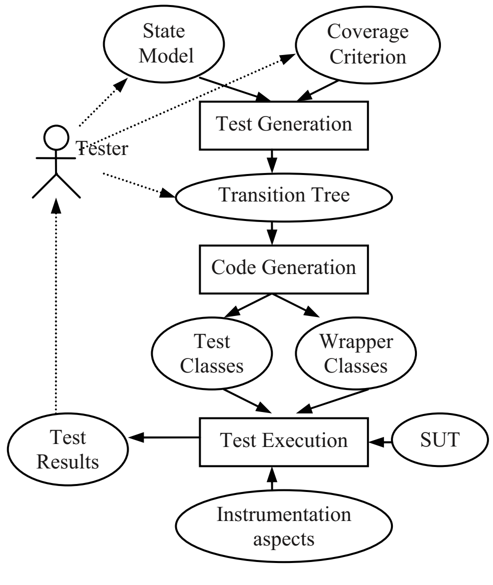
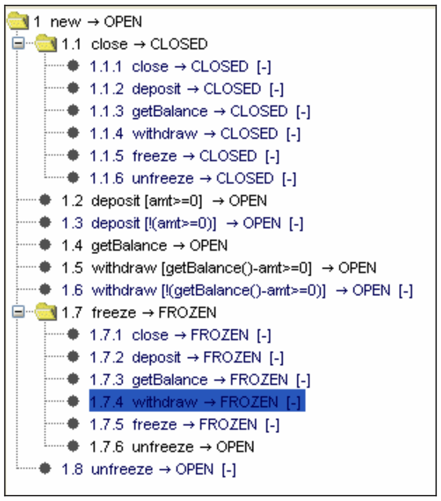


Figure 2. Automated testing process

Figure 3 shows the generated transition tree for the basic round-trip coverage of the *BankAccount* model in Figure 1. Negative test sequences, whose leaf nodes are illegal transitions, are marked with "[-]". For instance, the test sequence along the path $1 \rightarrow 1.7 \rightarrow 1.7.4$ consists of object creation and method invocations <*new*, *freeze*, *withdraw*> (the test input of the test sequence) as well as a sequence of expected resultant states <*OPEN*, *FROZEN*, *FROZEN*> (the oracle values of the test sequence). It is a negative test because one cannot *withdraw* money from a

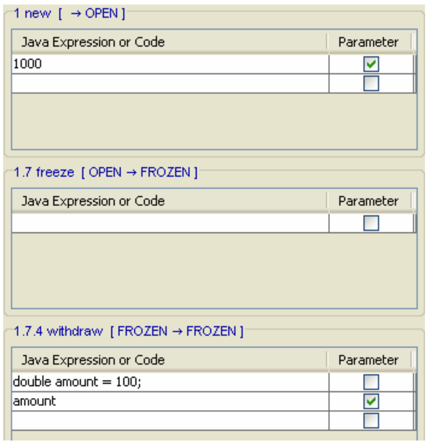
*FROZEN BankAccoun*t. This test is to check whether or not the *BankAccount* class implementation would actually prohibit such an operation.



```
1  new → OPEN
  1.1  close → CLOSED
      1.1.1  close → CLOSED  [-]
      1.1.2  deposit → CLOSED  [-]
      1.1.3  getBalance → CLOSED  [-]
      1.1.4  withdraw → CLOSED  [-]
      1.1.5  freeze → CLOSED  [-]
      1.1.6  unfreeze → CLOSED  [-]
  1.2  deposit [amt>=0]  → OPEN
  1.3  deposit [!(amt>=0)]  → OPEN  [-]
  1.4  getBalance → OPEN
  1.5  withdraw [getBalance()-amt>=0]  → OPEN
  1.6  withdraw [!(getBalance()-amt>=0)]  → OPEN  [-]
  1.7  freeze → FROZEN
      1.7.1  close → FROZEN  [-]
      1.7.2  deposit → FROZEN  [-]
      1.7.3  getBalance → FROZEN  [-]
      1.7.4  withdraw → FROZEN  [-]
      1.7.5  freeze → FROZEN  [-]
      1.7.6  unfreeze → OPEN
  1.8  unfreeze → OPEN  [-]
```

Figure 3. A transition tree for the model in Figure 1

Actual parameters have to be assigned to *new* and *withdraw* before the above test sequence becomes an executable test case (i.e., without compile-time errors). MACT provides a user-friendly interface for the tester to define such parameters. Once the tester clicks on a leaf node, the whole path from the root to the leaf is presented as a list of tables for editing. Figure 4 shows an editing session for the aforementioned test sequence. The user first inputs a value 1000 and uses it as the actual parameter for *new* by checking the parameter checkbox. The method *freeze* needs no parameter. For the invocation to *withdraw*, the user first provides a Java statement defining a double variable *amount* with value 100 (in this example, the parameter checkbox is not checked), and then uses *amount* as the actual parameter of *withdraw*.



1 new [→ OPEN]

Java Expression or Code	Parameter
1000	✓
	☐

1.7 freeze [OPEN → FROZEN]

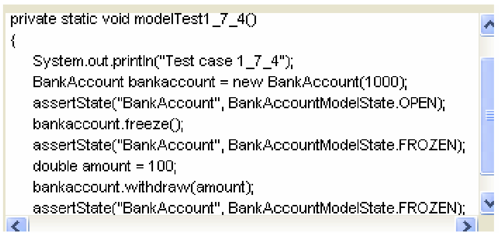
Java Expression or Code	Parameter
	☐

1.7.4 withdraw [FROZEN → FROZEN]

Java Expression or Code	Parameter
double amount = 100;	☐
amount	✓
	☐

Figure 4. A sample editing session

The ability to insert Java statements makes it possible to define runtime context for a specific testing task and set up and clean up test fixtures (e.g., establish and close a database or network connection before/after object creation or method invocations). Because the details of business logic (e.g., for *deposit* and *withdraw*) are often abstracted away in the state models, the tester is responsible for the satisfaction of method preconditions (e.g., getBalance()-amt<=0) when presenting actual test parameters. This is a non-trivial task, though. To alleviate this challenge, our future work will consider adapting a constraint satisfaction solver. This would require an executable language in place for specifying the detailed business logic (e.g., how *deposit* and *withdraw* operate).



```
private static void modelTest1_7_4()
{
    System.out.println("Test case 1_7_4");
    BankAccount bankaccount = new BankAccount(1000);
    assertState("BankAccount", BankAccountModelState.OPEN);
    bankaccount.freeze();
    assertState("BankAccount", BankAccountModelState.FROZEN);
    double amount = 100;
    bankaccount.withdraw(amount);
    assertState("BankAccount", BankAccountModelState.FROZEN);
```

Figure 5. A generated executable test method

After the required test parameters and additional code are completed, the test code generated by MACT is executable. If no method needs actual parameters, the generated test code is immediately executable. The general idea of code generation is as follows: a test method is created for each test sequence in a transition tree. Figure 5 shows the generated Java method for the aforementioned test sequence. The input value 1000 is used as the actual parameter for creating a bank account object: BankAccount bankaccount = new BankAccount(1000); The user-defined statement double amount=100; is inserted before the call to *withdraw*, and *amount* is used as the actual parameter of the call.

After each object creation and method invocation, an assertion is created to verify if the class under test has reached the expected state. For example, is the *bankaccount* object in the *FROZEN* state after the invocation to *freeze*? The user-defined code (e.g., double amount = 100;) is inserted either before or after the method invocation, depending on the order in which it occurs. Transition postconditions are also transformed into assertions. Once all test methods for the entire transition tree are created, MACT wraps them up into a test class and defines a main method that invokes all the corresponding test methods. This test class thus becomes a test suite that satisfies the selected coverage criterion.

MACT also generates a state wrapper class for each class involved in a state model. It consists of constants representing the state invariants in a state model and a getModelState method evaluating when the runtime object states achieve these state invariants. It thus builds a bridge between state invariants and runtime object states and facilitates determining whether tests pass or fail. The following code shows the state wrapper class for the *BankAccount* class. It is generated from the state definitions in the *BankAccount* state model. For example, the state invariant *OPEN* is defined as a constant. It represents a *bankaccount* object state that satisfies getClosed()==false && getBalance() >= 0 && getFrozen()==false.

```
public class BankAccountModelState{
    public static final String OPEN="OPEN";
    public static final String FROZEN="FROZEN";
    public static final String CLOSED="CLOSED";
    public String getModelState(BankAccount bankaccount){
        if (bankaccount.getClosed()==false
          && bankaccount.getBalance() >= 0
          && bankaccount.getFrozen()==false){
            return OPEN;     }
        else if(bankaccount.getFrozen()==true){
            return FROZEN;      }
        else if(bankaccount.getClosed()==true){
            return CLOSED;      }
        else return "Wrong state";
    }
}
```

The test execution infrastructure is supported by a collection of AspectJ aspects that instrument additional code to the class under test at runtime. AspectJ [6] is a Java-based aspect-oriented language. The aspects monitor runtime object states and compare them with the expected states. In brief, the generated test class and state wrapper classes, the code instrumentation aspects, and the SUT together form an executable system under the AspectJ running environment. Due to the limited space, this paper will not elaborate on the code instrumentation aspects. Interested readers can contact the authors for more details.

In addition, MACT provides a number of utilities for test management, such as saving/importing/merging test data and adding/modifying/deleting/cloning a node in a transition tree.

3. Automated Test Generation

3.1 Coverage Criteria for State Models

Our approach supports the state coverage, transition coverage, basic and extended round-trip coverage for automated test generation from state models. A test suite is said to achieve the state (or the transition) coverage if it covers each of the states (or the transitions) at least once. The basic round trip coverage refers to the Binder's round-trip path testing [1]. A basic round-trip test suite consists of a set of test sequences such that the resultant object state of each sequence has occurred at least once in some other sequence. An extended round-trip test suite consists of a set of test sequences such that the resultant object state and postcondition of each sequence is present at least once in some other sequence.

Let *A>B* represent that coverage criterion *A* subsumes coverage criterion *B* (i.e., a test suite that achieves A also achieves *B*). Then we have: extended round-trip > basic round-trip > transition coverage > state coverage. For example, the transition coverage subsumes the state coverage because a test suite of the transition coverage must cover all the states. The extended and basic round-trip coverage criteria are equivalent for a state model where no transitions have postconditions.

3.2 Test Generation Algorithms

Now we describe how the transition tree for a given coverage is generated. The root of a transition tree always represents the α state. The transition tree generation starts with the root and expands it.

The transition tree generation algorithm for the state coverage expands a node as follows: (1) find the transitions that start with the state represented by the current node (they are the object creation transitions if the current node is the root); (2) for each of these transitions, create a child node of the current node if its precondition can be satisfied and its resultant state is not yet traversed. The new child node represents the resultant state of the transition (it also contains a reference to the transition. This is similar for the other algorithms below). This state is marked as traversed; and (3) expand the new node.

The generation algorithm for the transition coverage expands a node as follows: (1) find the transitions that start with the state represented by the current node; (2) for each of these transitions, create a child node of the current node if the transition is not yet covered and its precondition can be satisfied. The new node represents the resultant state of the transition. The transition is marked as traversed; and (3) expand the new node.

The generation algorithm for the basic round-trip coverage expands a node as follows: (1) for each event, find the transitions that start with the state represented by the current node; (2) for each of the found transitions for the given event, create a child node of the current node if its precondition can be satisfied. The new child node represents the resultant state of the transition. Expand the new node if the resultant state has not appeared anywhere in the tree; (3) if no transition for the given event is found in the step (1) or the disjunction of the transition preconditions in step (2) is not a tautology (always true), create a new child node for the event (the event is illegal at the current state). The state of the new node is set to the state of the current node under expansion (i.e., an illegal event does not change object state). The precondition of the transition referenced by the new node is either null or the negation of the disjunction. Therefore the new node indicates a negative test. For example, the node 1.3 *deposit [!(amt>=0)]* $\rightarrow$ *OPEN [-]* in Figure 3 is a negative node. It is generated because *deposit* at the *OPEN* state is legal only when *amt>=0*. The extended round-trip coverage is similar to the basic one except for, in Step (2), expanding the new node if the resultant state and the transition postcondition are not contained by any node in the tree.

4. Reuse of Test Data for Modified Models

Frequent requirements change has been a norm in software development. To deal with requirements change, the design and implementation have to be modified. In the context of automated test generation, hand-crafted test data must be carried from one development version to the next. Consider the *BankAccount* model in Figure 1. Suppose a new banking policy allows overdrafts of up to $1,000. This requirements change is reflected in the modified *BankAccount* state model in Figure 6. The new *OVERDRAWN* state represents the balance of a *BankAccount* object is in the range of (0, -1000]. New transitions with respect to *deposit*, *withdraw* and *getBalance* are introduced.

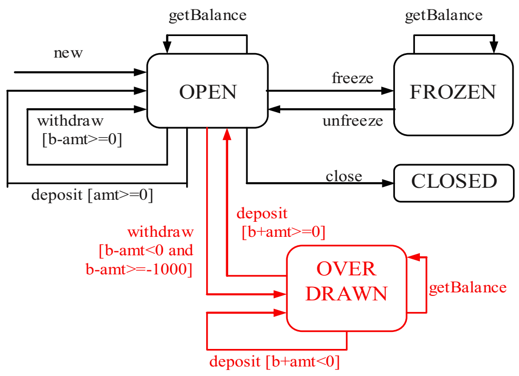


Figure 6. The modified *BankAccount* model

Let *M* and *M'* denote the models before and after modification, *TS* and *TS'* are their test suites, respectively. Each test sequence, *ts'*, in *TS'* belongs to one of the following situations:

(1) *ts'* needs no test parameters. In this case, its executable code can be generated immediately;

(2) *ts'* needs test parameters and is also a valid test sequence *ts* in *TS*. In this case, the user-defined test parameters for *ts* are all valid for *ts'*.

(3) *ts'* needs test parameters and it is also part of a valid test sequence *ts* in *TS*. In this case, the user-defined test parameters for the common part are all valid for *ts'*.

(4) *ts'* needs test parameters and it subsumes a valid test sequence *ts* in *TS* (i.e., *ts* is a sub-sequence of *ts'*). In this case, the user-defined test parameters for *ts* are all valid for *ts'*.

The situation (2) addresses the test selection problem of regression testing. It is not concerned about whether the tests in *TS* are valid or invalid. Obsolete tests in *TS* are not used in (i.e. automatically excluded from) the new test suite. Situations (3) and (4) deal with the coverage identification problem. They adopt existing test data, even if the test sequences containing these test data in *TS* have become obsolete. MACT offers an efficient algorithm for carrying test data from the test suite of one model to the next. Instead of comparing individual test sequences, it works directly on the two transition trees and associated test parameters.

5. Test Metrics and Experiments

MACT provides the following statistical information on the complexity of generated test suites:

- test methods (#M),
- test methods with negative tests (#N),
- constructor and method calls (#CM),
- assertions in the test methods (#A),
- parameters used in the tests (#P),
- parameters inputted by the tester (#PI),
- statements used in the tests (#S),
- statements provided by the tester (#SI).

We have applied MACT to the generation of executable test code for several applications. Duo to the limited space, here we only report the test metrics for the two *BankAccount* models in Figure 1 (denoted by *BA1*) and Figure 6 (denoted by *BA2*). Let *a, b, c* and *d* be the extended round trip, basic round-trip, transition coverage, and state coverage, respectively. As neither of *BA1* and *BA2* has postconditions, there is no difference between the extended and basic round-trip. The number of assertions (#A) is also the same as the number of constructor and method calls (#CM).

Table 1 shows the metrics of the executable test suites for *BA1*. For the round-trip coverage, there are 18 test methods (#M); 14 of them contain negative tests (#N); a total of 26 parameters (#P) are used in the test methods; and only nine (#PI) are direct inputs by the tester. Table 2 shows the test metrics of the (non-executable) test suites when they are first generated by reusing the test parameters for *BA1*. For the round-trip coverage, eight inputs (#PI) for *BA1* are carried into the test suite of *BA2* for 32 total parameters (#P, See Table 2). Table 3 shows the metrics of the executable test suites for *BA2*. For the round-trip coverage, a total of 13 tester-input parameters (#PI) are expected. The tester needs to provide five more parameter inputs after reuse. Similarly, for the transition/state coverage, the tester needs to input three and one more parameters, respectively.

Table 1. Metrics of the executable tests for *BA1*

	#M/N	#CM	#P/PI	#S/SI
BA1-a/b	18/14	66	26/9	1/1
BA1-c	2/0	10	4/3	0/0
BA1-d	2/0	6	0/0	0/0

Table 2. Metrics of *BA2* non-executable test suites generated by reusing *BA1* test parameters

	#M/N	#CM	#P/PI	#S/SI
BA2-a/b	25/17	94	32/8	1/1
BA2-c	3/0	21	7/3	0/0
BA2-d	3/0	9	3/1	0/0

Table 3. Metrics of the executable tests for *BA2*

	#M/N	#CM	#P/PI	#S/SI
BA2-a/b	25/17	94	43/13	1/1
BA2-c	3/0	21	11/6	0/0
BA2-d	3/0	9	4/2	0/0

6. Related Work

Significant research effort has been directed at the generation of test sequences from state models [6]. For example, the W-method [3] and Wp-method [4] construct a transition tree and traverse the transition tree so that each path is covered by the test cases. Many state-based test generation methods also use a state model to represent the SUT and then test whether or not the implementation and design models conform to each other. These methods have been extensively studied in the context of protocol testing [6]. However, none of them targets the testing of object-oriented programs. For example, events in the state machines are different from parameterized methods in object-oriented programming.

State models have also been used for model-based testing of object-oriented systems. The round-trip path testing [1] as the most referenced and applied technique is an adaptation of the W-method for deriving tests from a FREE state model (i.e., flattened Statechart) that describes the behavior of a single class or a cluster of classes. It replaces the identification sequence with a call to a state invariant checking method and requires the SUT to have a trusted ability to report the resultant states. Briand et al. [2] have recently conducted a series of controlled experiments evaluating the cost-effectiveness of the round-trip path testing, and they have showed that it can be enhanced by category partition. Hong et al. [5] provide a way to derive extended state machines from Statecharts to devise test criteria based on control and data flow analysis. Offutt et al. [8] provide definitions for such test criteria as all transitions, all transition pairs, and full-predicate. These criteria are used to evaluate how much of the state model is covered by a given test suite. Our approach uses the coverage criteria to drive test generation, i.e., generate tests that satisfy the criteria.

7. Conclusions

We have presented the framework for automated generation of executable test code from protocol state models. It supports four test coverage criteria. Reuse of hand-crafted test data for subsequently modified models can reduce the workload of creating new tests. MACT can also facilitate empirical evaluation of the cost-effectiveness (e.g., correlation of fault detection capability and testing costs) of various coverage criteria for test generation from state models. Such evaluation by hand would be tedious and error-prone without tool support.

Our future work will integrate a rigorous constraint language for specifying pre- and post-conditions in state models and a constraint problem solver for generation of test parameters. We will also investigate the test selection problem for model-based regression testing – how test sequences of modified models should be selected or prioritized.

8. Acknowledgement

This work was supported in part by the ND EPSCoR IIP-SG via NSF Grant EPS-047679 and the NASA OSMA/SARP Center Initiatives Program.

9. References

[1] Binder, R. V. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison-Wesley, 2000.

[2] Briand,L.C, Di Penta, M., Labiche, Y. "Assessing and Improving State-based Class Testing: A Series of Experiments", *IEEE Trans. on Software Engineering*, vol. 30, no. 11, pp. 770-793, Nov. 2004.

[3] Chow, T.S. "Testing Software Design Modeled by Finite-State Machines", *IEEE Trans. on Software Engineering*, vol. SE-4, May 1978, pp. 178-187.

[4] Fujiwara, S., Bochmann, G. v., Khendek, F., Amalou, M., Ghedamsi, A. "Test Selection Based on Finite State Models", *IEEE Trans. on Software Engineering*, vol. 17, no. 6, pp. 591-603, June, 1991.

[5] Hong, H.S., Kim, Y.G., Cha, S. D., Bae, D.H., Ural, H. "A Test Sequence Selection Method for Statecharts", *Journal of Software Testing, Verification and Reliability*, vol.10, no.4, pp.203-227,2000.

[6] Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J. and Griswold, W.G., An overview of AspectJ. *Proc. of ECOOP'01*, pp. 327-353, 2001.

[7] Mathur, A.P. *Foundations of Software Testing*, Draft v2.1, Purdue University, 2005.

[8] Offutt, J., Liu, S., Abdurazik, A., and Ammann, P. "Generating Test Data from State-Based Specifications". *Journal of Software Testing, Verification and Reliability*, vol.13, no.1,pp. 25-53, 2003.

[9] Rothermel, G. and Harrold, M.J. "Analyzing Regression Test Selection Techniques", *IEEE Trans. on Software Engineering*, vol. 22, no. 8, pp. 529-551, 1996.

[10] *UML 2.0 Specification*, http://www.omg.org/technology/documents/formal/uml.htm.

[11] Wong, W.E., Horgan, J.R., London, S., and Agrawal, H. "A Study of Effective Regression Testing in Practice", In *Proc. of the Eighth IEEE International Symposium on Software Reliability Engineering*, pp. 522-528, November 1997.

Validating A Layered Decision Framework for Cost-Effective Network Defense

Huaqiang Wei, Jim Alves-Foss
Department of Computer Science
University of Idaho
{wei3004, jimaf}@uidaho.edu

Du Zhang
Department of Computer Science
California State University, Sacramento
zhangd@ecs.csus.edu

Abstract

Cost-effective network defense includes at least three decision layers: security policies, defense strategies, and real-time defense tactics for countering immediate threats. A layered decision model (LDM) has been proposed to capture this decision process, and help us select cost-effective defense mechanisms to safeguard computer networks. This paper describes our efforts in validating the rationality and consistency of the LDM through simulation. The results indicate that the LDM is rational and consistent in cost-effective network defense.

Keywords: cost-effective network defense, layered decision model, simulation, rationality, consistency.

1. Introduction

How to safeguard computer networks is a critical and complex issue. The decision making process of network safeguarding is concerned with not only what security goals to be fulfilled, but also how to cost-effectively fulfill these goals. To fully understand the decision process we must develop proper means to obtain insight into the structure and security relationships of entities and decision types involved in network defense. One way to accomplish that is to develop a model that represents and captures the real-world decision process. Though several studies have applied modeling and simulation approaches to the investigation of network security and information assurance [1-9, 11-14], these researches focus on specific information security systems, phenomena, or network attack/defense processes at specific levels of protection (i.e. strategic level or tactic level), and do not combine defense mechanisms with security policies, security requirements, business goals and cost-benefit analysis into a single coherent framework.

To address these issues, we developed a layered decision model (LDM) for cost-effective network defense [15, 18]. The LDM includes three essential decision layers (security policies, defense strategies and real-time defense tactics) used to define decision parameters, establish explicit relationships between decision types, and support and record decisions made. The decision making process in the model is driven by a return-on-investment (ROI) cost-benefit analysis.

The focus of this paper is on the issue of the LDM's rationality and consistency. We describe a simulation based approach to validate the LDM's rationality and consistency. The reminder of the paper is organized as follows. Section 2 gives a brief overview of the LDM. Section 3 discusses the simulation of the LDM. Section 4 presents the results of a case study. Section 5 offers a brief comparison. Section 6 concludes the paper.

2. Overview of the LDM

The LDM, the details of which can be found in [15, 18], includes three decision layers. Security policies are defined at Layer Zero, defense strategies at Layer One, and defense tactics at Layer Two. Security policies, defense strategies, and real-time defense tactics vary with business types and times. To formally describe the LDM, we consider a particular business type b at a particular time τ, and make the following definitions:

$G = \{g_1,\ldots, g_I\}$ is a set of business goals.
$T' = \{t_j, \ldots, t_M\}$ is a set of threats
$T = \langle t_1,\ldots, t_M\rangle$ is an ordered set of threats after ranking based on expected cost.
$P = \{p_1,\ldots, p_L\}$ is a set of security policies.
$S = \{S_1,\ldots, S_k\}$ is a set of defense strategies.
$R = \{r_1,\ldots, r_Z\}$ is a set of defense tactics.

Figure 1 is the LDM flow chart. Compared with the related work, the LDM has the following advantages: (1) It offers a consistent framework to reasonably organize the decision process. (2) It explicitly establishes the connectivity among security policies, defense strategies and defense tactics. This allows us to show how decisions made in one layer impact decisions in another layer. (3) It optimizes security decisions within the context of business goals through cost-benefit analysis.

2.1 Decision process

A decision making process in the LDM can be described as a decision graph with start points and end points. The use of the LDM can help select the best defense for a given threat from a set of defenses with different cost-effectiveness. This decision process should be consistent and rational.

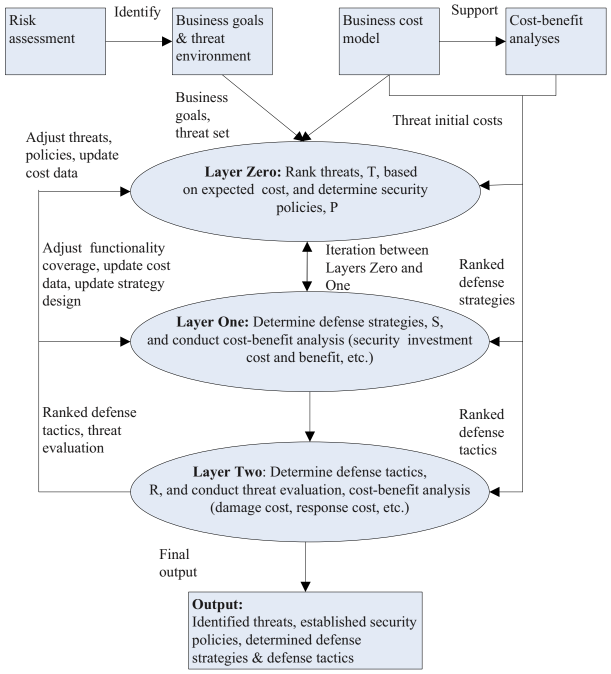

Risk assessment
Identify
Business goals & threat environment
Business cost model
Support
Cost-benefit analyses
Business goals, threat set
Threat initial costs
Adjust threats, policies, update cost data
Layer Zero: Rank threats, T, based on expected cost, and determine security policies, P
Adjust functionality coverage, update cost data, update strategy design
Iteration between Layers Zero and One
Ranked defense strategies
Layer One: Determine defense strategies, S, and conduct cost-benefit analysis (security investment cost and benefit, etc.)
Ranked defense tactics, threat evaluation
Ranked defense tactics
Layer Two: Determine defense tactics, R, and conduct threat evaluation, cost-benefit analysis (damage cost, response cost, etc.)
Final output
Output: Identified threats, established security policies, determined defense strategies & defense tactics

Figure 1. Layered decision model.

Definition 1: A decision process (*DP*) corresponds to a directed graph (*DG*) which is composed of a set of sub-directed graphs:

$$DG = (SDG_i \mid i = 1 \ldots c) \qquad (2.1)$$

Each sub-directed graph SDG_i represents an execution path of a layered decision making process that goes through the defense strategy $S_i \in S$, and includes the vertices and arcs in the dotted area as shown in Figure 2 below.

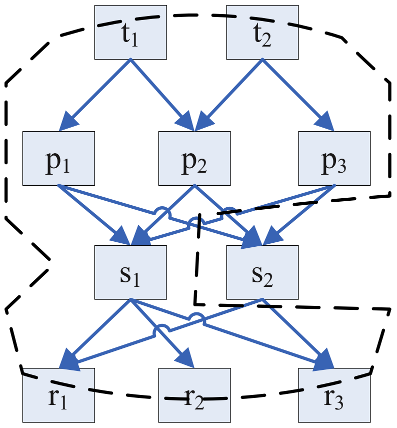


Figure 2. An SDG_i example.

Let $T_i = T \cap SDG_i$, $P_i = P \cap SDG_i$, $R_i = R \cap SDG_i$. Let $<v_i, v_j>$ denote an arc from v_i to v_j. Then we have

$$E_i^{T,P} = \{<t_j, p_k> \mid t_j \in T_i \wedge p_k \in P_i\}$$

$$E_i^{P,S} = \{<p_j, S_i> \mid p_j \in P_i \wedge S_i \in S\}$$

$$E_i^{S,R} = \{<S_i, r_j> \mid S_i \in S \wedge r_j \in R_i\}$$

Thus, an SDG_i is defined as follows:

$$SDG_i = (S_i, V_i, E_i, ROI_i) \qquad (2.2)$$

where

$$S_i \in S.$$
$$V_i = T_i \cup P_i \cup \{S_i\} \cup R_i$$
$$E_i = E_i^{T,P} \cup E_i^{P,S} \cup E_i^{S,R}$$
$$ROI_i = \text{Max}\{\text{ROI}(r_j) \mid r_j \in R_i\}.$$

2.2 Properties

We are interested in two properties here for the model: consistency and rationality. We say that a policy covers a threat, denoted by $p \trianglerighteq t$, if t will be thwarted when p is in place. A strategy S is said to be consistent with a policy p, denoted as $S \trianglerighteq p$, if S is a plan of actions to fulfill the goals in p. A tactic r is consistent with a strategy S, denoted as $r \trianglerighteq S$, if r sanctions a technique that implements S.

Definition 2: Consistency. A decision process in the LDM is said to be *consistent* if we have the following:

$$\forall t \in T\, \exists p \in P\, \exists S_i \in S\, \exists r \in R\, (r \trianglerighteq S_i \trianglerighteq p \trianglerighteq t) \qquad (2.3)$$

We use $r \trianglerighteq^{+} t$ to indicate that r is a consistent tactic to the threat t.

Let $\Re(DP)$ indicate a set of recommendations produced by a decision process *DP* in the LDM. The model's rationality can be defined as follows.

Definition 3: Rationality. A decision process *DP* in the LDM is *rational* if and only if *DP* is consistent and its recommendations yield the best cost-effective actions in terms of ROI:

$$\forall t \in T\, \exists p \in P\, \exists S_i \in S\, \exists r \in R\, [(r \trianglerighteq S_i \trianglerighteq p \trianglerighteq t) \wedge (r \in \Re(DP)) \wedge (\forall \text{ROI}(r_j) \in \{\text{ROI}(r_j) \mid r_j \in R \wedge r_j \trianglerighteq^{+} t\}\ (\text{ROI}(r) \geq \text{ROI}(r_j)))] \qquad (2.4)$$

3. Simulation of LDM

The purpose of developing a simulation software and conducting simulations is two-fold: (1) to demonstrate how the LDM can be applied in decision making process for cost-effective network defense; and (2) to validate the rationality and consistency properties for the model.

We developed a Prolog based simulation software to capture the essence in the LDM and to validate its properties. The simulation software ranks the threats, and determines security policies, cost-effective defense strategies, and defense tactics based on input threat profiles and available defense mechanisms. The input of the simulation software includes a set of facts to represent threat profiles, security policies, available defense mechanisms and estimates of their costs and effectiveness. We also define a set of Prolog rules to search for the defense strategies, defense tactics, and calculate their cost-effectiveness (ROI). The following are some defined Prolog facts and rules:

- *Annual frequency of threats***.** The estimated annual frequency for each threat is defined as `frequency(Threat, F)`.
- *Single loss expectancy*. The cost of damage or the single loss expectancy (SLE) for each threat is defined as `sle(Threat, SLE)`. This is the post-occurrence cost that includes labor cost, material cost, data loss, idle pay and business disruption. Annual loss expectancy (ALE) for each threat can be obtained by multiplying SLE of the threat with its annual frequency.
- *Cost of defense mechanisms***.** The investment cost for each defense strategy is defined as `cost_s(Strategy, Cost)`. This is a pre-occurrence cost that includes: material cost, labor cost, maintenance cost, and training cost. The investment cost for each defense tactic is defined as `cost_t(Tactic, Threat, Cost)`. This is also a pre-occurrence cost that includes: operation cost and response cost (The unit of cost is up to the users. We assume US dollars in this work).
- *Effectiveness of defense mechanism***.** The estimated effectiveness of each defense strategy when countering each threat is defined as `effect_s(Threat, Strategy, E)`. Similarly, the estimated effectiveness of each defense tactic when countering each threat is defined as `effect_t(Threat, Tactic, E)`.
- *Mappings between layers*. Several mappings are defined here:
  - `threat_policy(Threat,Policy)` for the relationships from threats to security policies (which `Threat` is covered by what security `Policy`.) Each threat is covered by at least one security policy, and all threats must be covered.
  - `policy_strategy(Policy,Strategy)` for mappings from a security policy to a defense strategy (which `Strategy` fulfills what `Policy`.
  - `strategy_tactic(Strategy,Tactic)` for mappings from defense strategies to defense tactics (which defense `Strategy` determines what defense `Tactic`.
  - `enforce_PST(Pol, Stra, Tac)` is used to enforce the consistency among security policies, defense strategies and defense tactics (`Tac` $\trianglerighteq$ `Stra` $\trianglerighteq$ `Pol`).
- *Cost-benefit analysis***.** We use ROI as the driving force behind the cost-benefit analysis to enforce the model rationality. The canonical definition of ROI is defined as (benefits – costs)/costs [15, 18]. Thus,

$$\mathrm{ROI}(S) = [(\textstyle\sum \mathrm{ALE}(t)\times \mathrm{effect}(t,S)) - \mathrm{cost}(S)]/\mathrm{cost}(S) \quad (3.1)$$

  where the summation is ranging over $t \in T$ and S is a defense strategy. The ROI can be defined similarly for defense tactics. Details on the cost-effective analysis can be found in [15-18].

4. A simulation case study

A simulation case study is included for cost-effective network defense for an e-commerce company. This company manages Web-based services for real estate brokers and mortgage companies. The network system handles customers' on-line requests on membership applications, property searching, and loan applications. Therefore, the security in the network system is very critical in protecting customers' personal and confidential data, such as credit card numbers, social security numbers, and financial information.

4.1 Facts of the simulation case

Based on interviews with security managers at the e-commerce company, we defined the following facts as inputs to the simulation system.

(1). A set of threats (T), annual frequencies, and SLEs are shown in Table 1. The ranked threat set is as follows: $T = \langle t_1, t_6, t_2, t_3, t_5, t_4\rangle$.

(2). A set of security policies (P) and their coverage are given in Table 2. Therefore, the established security policy set is: $P = \{p_1, p_2, \ldots, p_{13}\}$.

(3). A set of defense strategies $\{S_1, S_2\}$ and their purposes. Defense strategy S_1 is shown in Table 3, and S_2 in Table 4.

(4). The estimated effectiveness and cost of each defense strategy when countering specific threats are given in Table 5.

(5). A set of defense tactics (R) is shown below:

r_1: Block access (*BA*)
r_2: Terminate session/connection and disable account (*TSCD*)
r_3: Record/log and notify administrator (*RLA*)
r_4: Switch to redundant network (*SRN*)
r_5: Back up and restore (*BR*)
r_6: Turn off the host and reboot server (*TOHRS*)
r_7: Automatically scan and clean (viruses and worms) (*ASC*)
r_8: Cooperate with other ISPs (Internet service providers) for rate limiting (*RL*)
r_9: No response (*NR*)

Therefore, the defense tactic set is

$R = \{r_1, r_2, \ldots, r_9\}$.

Table 6 lists the effectiveness and investment cost of each defense tactic when countering specific threats.

4.2 Results of the simulation

Based on the above established facts, we run the simulation software with the cost-effectiveness information for each defense strategy and defense tactic. We could also query the hierarchical relationships between security policies, defense strategies, and defense tactics. Table 5 is the simulation result of defense strategies, which indicates that S_1 has a better cost-effectiveness (ROI) than that of S_2. Table 6 presents the simulation result of defense tactics and illustrates that "terminate session/connection and disable account (*TSCD*)" is the best defense tactic for the "internal user misuse", and "block access (*BA*)" is the best defense tactic for the rest of the attacks.

Table 1. Threat (*T*) rankings.

Threats	Estimated annual frequency	Single loss expectancy (SLE)	Ranking
Unauthorized access (UA) (t_1)	20	\$10k	1
Internal user misuse (IUM)(t_6)	50	\$3k	2
Application level attack (ALA) (t_2)	10	\$10k	3
Denial of service attack (DoS) (t_3)	5	\$10k	4
Virus and worm attack (VWA) (t_5)	10	\$4k	5
IP spoofing attack (IPSA) (t_4)	2	\$10k	6

Table 2. Security policies (*P*) and coverage.

Security policies	Coverage
p_1: Ingress and egress filtering must always be conducted.	$t_1 - t_6$
p_2: The system must be virus free.	t_5
p_3: If network traffic exceeds its normal threshold by 25%, traffic rate limitation must be activated.	$t_1 - t_3$
p_4: If the Web server is substantially slower than normal, security manager may need to restart the Web server or switch the service to a backup server.	$t_2 - t_5$
p_5: Remote access must be authenticated with passwords, and passwords must be no less than 8 characters long and must be changed every 60 days.	t_1, t_6
p_6: Improper communication between servers must be recognized and blocked.	t_1, t_3, t_4, t_6
p_7: Unauthorized access must be blocked.	t_1, t_2, t_4, t_6
p_8: Communication with servers must be encrypted.	$t_1 - t_6$
p_9: No unapproved software may be installed on any workstation without authorization from the security managers.	t_2, t_5
p_{10}: No email or Internet access is allowed on critical corporate financial servers and database servers.	$t_1 - t_6$
p_{11}: All account security events must be logged.	$t_1 - t_6$
p_{12}: All server data will be backed up daily using incremental backup.	$t_1 - t_6$
p_{13}: If users' behaviors are not authorized, their accounts will be closed.	t_1, t_6

Table 3. Features and coverage areas of S_1.

Defense techniques	Location	Policy goals	Threats covered
Three firewalls	Between gateway and Internet; between Web server and application server; between application server and database server	p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{13}	Control access (major threats: $t_1 - t_6$)
Three host-based network intrusion detection systems(HIDS)	One on each server	p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_9, p_{10}, p_{11}, p_{12}, p_{13}	Monitor entire network (major threats: $t_1 - t_6$)
Virtual private network (VPN)	At remote access points	p_5, p_7, p_{11}, p_{13}	Secure remote control (major threats: t_1, t_2, t_3)
RFC 2827 and 1918 protocols	Whole network	p_1, p_2, p_3, p_4, p_6, p_7	Enforce ingress and egress filtering (major threat: t_3)
Content filtering server	Whole network	p_1, p_2, p_6, p_7, p_9, p_{10}, p_{13}	Scan URL request (major threat: t_4)
Virus/worm scanner	Whole network	p_1, p_2, p_5, p_6, p_7, p_9, p_{10}	Detect and disinfect viruses (major threat: t_5)
Level 2 switch	Between the servers	p_3, p_4, p_6, p_7, p_{10}	Maintain proper communication (major threats: $t_1 - t_4$)
Level 3 switch with IDS	Between the servers	p_1, p_2, p_3, p_4, p_6, p_7, p_{10}	Maintain proper communication (major threats: $t_1 - t_6$)
Level 4-7 application switch	Whole network	p_1, p_3, p_4, p_5, p_6, p_7, p_9, p_{10}, p_{13}	Regulate network traffic (major threats: t_1, t_3, t_5)
Secure Sockets Layer (SSL)	Whole network	p_1, p_5, p_6, p_7, p_8, p_{10}	Web-based transactions (major threats: t_1, t_4, t_6)
IPSec encryption	Whole network	p_5, p_6, p_7, p_8, p_9, p_{10}	Enforce authentication (major threats: $t_1 - t_6$)

Table 4. Features and coverage areas of S_2.

Defense techniques	Location	Policy goals	Threats covered
One firewall	Outside Web server	p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{13}	Control access (major threats: $t_1 - t_6$)
Intrusion detection system (IDS)	Whole network	p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_9, p_{10}, p_{11}, p_{12}, p_{13}	Monitor entire local network (major threats: $t_1 - t_6$)
Virtual private network (VPN)	At remote access points	p_5, p_7, p_{11}, p_{13}	Secure remote control (major threats: t_1, t_2, t_3)
RFC 2827 & 1918 protocols	Whole network	p_1, p_2, p_3, p_4, p_6, p_7	Ingress and egress filtering (major threat: t_3)
Network address translator (NAT)	Whole network	p_5, p_7, p_{11}, p_{13}	Conceal real IP addresses (major threats: t_1, t_2, t_3)
Virus/worm scanner	Whole network	p_1, p_2, p_5, p_6, p_7, p_9, p_{10}	Detect and disinfect viruses (major threat: t_5)
Secure Sockets Layer (SSL)	Whole network	p_1, p_5, p_6, p_7, p_8, p_{10}	Secure Web-based transactions (major threats: t_1, t_4, t_6)

Table 5. Defense strategies (S) and cost-effectiveness.

Cost / Threat	S_1		S_2	
	Effectiveness	**Benefit**	**Effectiveness**	**Benefit**
Unauthorized access (UA)	0.85	Benefit(UA, 1)= $170k	0.6	Benefit(UA, 2)= $120k
Internal user misuse (IUM)	0.9	Benefit(IUM, 1)= $135k	0.5	Benefit(IUM, 2)= $75k
Application level attack (ALA)	0.85	Benefit(ALA, 1) = $85k	0.6	Benefit(ALA, 2) = $60k
Denial of service attack (DOS	0.75	Benefit(DOS, 1) = $38k	0.5	Benefit(DOS, 2) = $25k
Virus and worm attack (VWA)	0.75	Benefit(VWA, 1) =$30k	0.7	Benefit(VWA, 2) =$28k
IP spoofing attack (IPSA)	0.7	Benefit(IPSA, 1) =$14k	0.6	Benefit(IPSA, 2) =$12k
Investment cost of each defense strategy		$200k		$160k
Expected total net benefit (net cost saving)		$272k		$160k
ROI		1.36		1.0

Table 6. Defense tactics and cost-effectiveness.

Cost / Threat	Defence tactics	Damage cost (SLE)	Response cost	Operational cost	Effectiveness	ROI
Unauthorized access (UA)	BA	$10k	$2k	$2k	0.9	1.25
	TSCD		$3k	$2k	0.8	0.6
	RLA		$2k	$2k	0.5	0.25
	SRN		$5k	$4k	0.95	0.05
	BR		$3k	$3k	0.7	0.16
	ASC		$2k	$1.5k	0.2	-0.4
	TOHRS		$5k	$2k	0.9	0.8
	RL		$10k	$4k	0.2	-0.85
	NR		$0	$0	0.0	(-$10k)
Internal users misuse (IUM)	BA	$3K	$1k	$2k	0.6	-0.4
	TSCD		$1k	$1k	0.9	0.35
	RLA		$5k	$4k	0.8	-0.1
	SRN		$5k	$4k	0.2	-0.93
	BR		$3k	$3k	0.4	-0.8
	ASC		$2k	$1.5k	0.2	-0.8
	TOHRS		$5k	$2k	0.7	-0.7
	RL		$10k	$4k	0.1	-0.98
	NR		$0	$0	0.0	(-$3k)
Application level attack (ALA)	BA	$10k	$2k	$2k	0.85	1.25
	TSCD		$3k	$2k	0.8	0.6
	RLA		$2k	$2k	0.5	0.25
	SRN		$5k	$4k	0.9	0.0
	BR		$3k	$3k	0.7	0.2
	ASC		$2k	$1.5k	0.2	-0.95
	TOHRS		$5k	$2k	0.9	0.3
	RL		$10k	$4k	0.2	-0.99
	NR		$0	$0	0.0	(-$10k)
Denial of service attack(DOS)	BA	$10k	$2k	$2k	0.8	1.0
	TSCD		$3k	$2k	0.7	0.4
	RLA		$2k	$2k	0.55	0.4
	SRN		$5k	$4k	0.95	0.0
	BR		$3k	$3k	0.8	0.3
	ASC		$2k	$1.5k	0.2	-0.95
	TOHRS		$5k	$2k	0.85	0.2
	RL		$10k	$4k	0.9	-0.3
	NR		$0	$0	0.0	(-$10k)
Virus and worm attack (VWA)	BA	$4k	$2k	$1k	0.9	0.2
	TSCD		$3k	$2k	0.6	-0.2
	RLA		$2k	$2k	0.6	-0.2
	SRN		$5k	$4k	0.5	-0.7
	BR		$3k	$3k	0.7	-0.5
	ASC		$2k	$1.5k	0.9	0.1
	TOHRS		$5k	$2k	0.65	-0.7
	RL		$10k	$4k	0.2	-0.96
	NR		$0	$0	0.0	(-$4k)
IP spoofing attack (IPSA)	BA	$10k	$2k	$2k	0.8	1.0
	TSCD		$3k	$2k	0.9	0.8
	RLA		$2k	$2k	0.6	0.5
	SRN		$5k	$4k	0.9	0.0
	BR		$3k	$3k	0.85	0.4
	ASC		$2k	$1.5k	0.2	-0.95
	TOHRS		$5k	$2k	0.9	0.3
	RL		$10k	$4k	0.2	-0.98
	NR		$0	$0	0.0	(-$10k)

5. Discussion

While there is a proliferation of separate modeling and simulation methodologies [15] for decision support for network defense, there is no comprehensive technology that integrates interrelated decisions about security policies, cost-effective defense strategies, and real-time defense tactics into a single efficient decision framework. The LDM overcomes these weaknesses and connects these three essential decision types together through the combination of risk management, cost modeling, and cost-benefit analysis. Comparing to other modeling and simulation approaches [1-9, 11-14], our simulation system is simple, preserving the major features of the LDM framework, which include searching hierarchical relationships between threat profiles, security policies, defense strategies, and real-time defense tactics, and conducting cost-benefit analysis for selecting the best defense mechanisms. Executing the simulation software can help analyze diverse business cases, and allow security managers to gain insight into the hierarchical relationships among the inter-connected entities and decision types before actually implementing a defense plan. The simulation system can be a useful tool to test the LDM features and performance, which include model rationality and sensitivity.

6. Conclusion and future work

This paper presents a study of validating the rationality and consistency for the proposed layered decision framework for cost-effective network defense. The LDM combines cost modeling and cost-benefit analysis for determining cost-effective defense mechanisms for network defense. We wrote a Prolog program to simulate the LDM and to obtain some empirical results through some real-world case studies. The preliminary results indicate that the LDM is rational and consistent in cost-effective network defense.

Future work includes: conducting more simulation cases and combining the probabilistic approach with the simulation system to further study how the uncertainty and variability of input variables affect the decision results and to test the model's sensitivity.

Reference

1. S. Butler, "Security attribute evaluation method: A cost-benefit approach", *Proceedings of the 24th International Conference on Software Engineering*, Orlando, FL, USA, May 19-25, 2002, pp. 232-241.
2. R. Campbell and G. Sands, "A modular approach to computer security risk management", *The American Federation of Information Processing Societies (AFIPS)Conference Proceedings*, AFIPS Press, 1979, V. 48, pp.293-303.
3. H. Cavusoglu, S. Raghunathan and B. Mishra, "A model for evaluating IT security investments", *Communications of the ACM*, July 2004, 47(7): pp.87-92.
4. F. Cohen, "Simulating cyber attacks, defense and consequences",http://www.all.net/journal/ntb/simulate/simulate.html, March 1999.
5. F. Cohen, "Managing network security: attack and defense strategies",http://www.windowsecurity.com/whitepapers/Managing_Network_Security_Attack_and_Defense_Strategies.html, 2002.
6. J. Conrad, "Analyzing the risks of information security investment with Monte-Carlo simulations", 4th Workshop on the Economics of Information Security, WEIS 2005, Cambridge, MA, USA.
7. P. Fites and M. Kratz, *Information systems security: a practioner's reference*, Van Nostrand Reinhold, New York, NY, 1993.
8. C. Locher, "Methodologies for evaluating information security investments-what BASEL II can change in the financial industry", http://csrc.lse.ac.uk/asp/aspecis/20050136.pdf, 2005.
9. K. Lye and J. Wing, "Game strategies in network security", *Proceedings of the Foundations of Computer Security Workshop 2002*, Copenhagen, Denmark, July 26, 2002.
10. K. Ning, Y. Chen and D. O'Sullivan, "Rationality Validation of Business Process Model by Simulation Method", 4th International Conference on E-Business, Beijing, China, Dec. 5-9, 2004.
11. J. Saunders, "A Dynamic Risk Model for Information Technology Security in a Critical Infrastructure Environment", *Proceedings of Risk-Based Decision Making in Water Resources X, 10th United Engineering Foundation Conference*, Santa Barbara, CA, USA, Nov. 3-8, 2002, pp. 23-39.
12. J. Saunders, "The Case for Modeling and Simulation of Information Security", http://www.johnsaunders.com/papers/securitysimulation.htm, 2006.
13. G. Schow, W. Lunceford, L. Massey, J. Filsinger and L. Adelman, "Utilizing Modeling and Simulation (M&S) to Evaluate Information Assurance (IA) Policies, Guidance, and Systems Designs", *Proceedings of the 2000 IEEE Workshop on Information Assurance and Security*, United States Military Academy, West Point, NY, USA, June 6-7, 2000, pp.26-32.
14. H. Seo and T. Cho, "Simulation Model Design of a Security System Based on a Policy-Based Framework", *SIMULATION*, 2003, 79(9): pp. 515-527.
15. H. Wei, "A layered decision model for cost-effective network safeguarding", Ph.D. Dissertation, the University of Idaho, December, 2006.
16. H. Wei, D. Frincke, O. Carter and C. Ritter, "Cost-benefit analysis for network intrusion detection systems", *CSI 28th Annual Computer Security Conference*, Washington, DC, USA, Oct. 29-31, 2001.
17. H. Wei and D. Frincke, "Risk assessment and cost-effective business modeling for network security", *The 7th World Multi-Conference on Systemics, Cybernetics and Informatics SCI 2003*, Orlando, FL, USA, July 29-Aug.1, 2003.
18. H. Wei, H., D. Frincke, J. Alves-Foss, T. Soule and H. Pforsich, "A layered decision model for cost-effective network defense", *Proceedings of the 2005 IEEE International Conference on Information Reuse and Integration*, Las Vegas, NV, USA, August 15-17, 2005, pp. 506-511.

Toward Modeling and Analysis for Software Installation Testing

Jerry Gao, Sujana Tirumalasetti, Chien-Pin Hsu, Yip Cheong
San Jose State University, San Jose, California 95192, USA

Anne Colendich and Todd Fitch
Intuit, Mountain View, USA

Abstract: Software testing is the last critical phase in software quality control. Software installation testing is one of the most important and complex tasks in system testing. However, in the past years, researchers have not paid much attention to the related issues and challenges in software installation testing. Today test and QA engineers have lacked systematic processes, test models, methods and tools to help them in software installation testing and patch testing. This paper addresses the test modeling and analysis issues in software installation testing. The paper proposes a new model, known as *a semantic tree*, to assist engineers to model and present diverse system environments and configurations, various running conditions, and system functional features. It is very useful to assist engineers to automatically identify, generate software installation test items and test cases.

KEYWORDS: software test modeling and analysis, software installation testing, software installation criteria and standards.

1. Introduction

Software testing is the last critical phase in software quality assurance. Software installation testing is one of the important types of system testing. It is also one of the most complicated types of testing to plan and execute. However, in the past years, researchers have not paid much attention to the related issues and challenges in software installation testing. Hence, until now, testing and QA engineers have lacked well-defined installation testing processes, practical and cost-effective test models, well-defined test criteria, systematic validation methods and tools. According to QA and test engineers in the real world, they need to cope with the following challenges in software installation validation.

- Validating software on diverse system environments with many different configurations.
- Validating software under various system running conditions.
- Validating software under a very tight schedule without well-defined installation test models, test criteria and tools.

Since software installation provides the first encounter for users and sets the tone for them on software quality, the quality of software installation becomes very critical for software vendors – i.e. first impressions matter tremendously. Many software makers experience a high abandonment rate due to the quality issue of software installation. Many users give up before using the software product. According to our received feedbacks from industry, the current practice in the real world can be summarized as follows.

- Software installation testing is conducted in an ad-hoc manner without test models and coverage criteria.
- Installation test case design is done manually without well-defined methods, test criteria and standards.
- Most test executions are done manually although some engineers use the existing software test tools to create GUI-based executable test scripts.
- It is common practice to use a matrix [3] to carry out software installation planning.

Based on our recent literature survey, there are a very few published papers discussing software installation topic and issues. Hence, there are four urgent needs in software installation validation. They are given below:

- Well-defined software installation test processes with cost-effective test criteria and standards.
- Well-defined test models to assist engineers to model and present diverse system environments and configurations, running conditions, and installation functions.
- Well-defined metrics measuring and predicting test complexity, costs, and coverage.
- Cost-effective systematic test generation methods and tools to support automatic software installation testing.

Today, model-driven software construction becomes a popular research topic in software engineering [4]. Recently, there have been a number of published papers using a model-based approach to conducting test selection [8], test specification and data generation [6], as well as test automation [5]. The typical used models are: a) finite-state machine [9][7], b) UML models [6], and statistical models. These models are not useful to address the needs of software installation testing. This paper addresses the first two issues and needs in software installation testing by proposing new models to support installation testing and analysis. In this paper, we introduce a new model, known as a semantic tree, to assist engineers to perform modeling and analysis for software installation testing. This model can be used to present diverse system environments and configurations, various running conditions, and functional features and capabilities. It is very useful to assist engineers in automatically identifying and generating software installation environments and configurations, and conditions, as well as related installation function test items. Based on the given model, a set of software installation test criteria, test complexity, and coverage metrics are defined.

The paper is structured as follows. The next section discusses the basic concepts, background, issues and challenges in software installation testing. Section 3 proposes a test model, known as a semantic tree, for software installation testing and analysis. Section 4 provides a set of metrics to define software installation test criteria and compute test complexity and coverage. Section 5 provides the conclusion remarks and future work.

2. Software Installation Testing

What is software installation testing? Its ***major purpose*** is to validate software products to see if they can be correctly installed in a specified system environment with proper system configurations and running conditions. Moreover, it is able to demonstrate the installation functions and behaviors correctly. The major focus of software installation validation is to find the answers to the following questions:

- Can a given software product be properly installed on all specified system configurations?
- On each configuration system environment, can the software be successfully installed under all of validated running conditions?
- Does the under test software product demonstrate its installation functions and behaviors correctly?

Similar to other types of software testing activities, conducting software installation testing needs a well-defined test process. Based on our understanding and experience, a ***software installation test process*** must include the following steps:

1. Understand, identify and document all of the system configurations and possible environment settings.
2. Understand, identify, and document all of the system installation conditions for each configured system environment.
3. Plan software installation testing following the steps below:
   - o Identifying and select the right test standards and test criteria
   - o Defining tasks and a working schedule
   - o Selecting a cost-effective test strategy by focusing on the important and popular system configurations and running conditions
   - o Selecting (and adopting) a proper installation test tool
4. Identify and design installation test cases, and develop automatic test scripts.
5. Execute the specified installation tests and report results.

Although there has been a great deal of research work on software testing in the past decades, only a few of them address the issues in software installation testing. Typical issues are given as follows:

- Is there an effective way to identify diverse system configurations and settings during software installation?
- Is there an effective way to identify various system running conditions for software installation?
- What are the proper quality control standards and test criteria for software installation?
- What are the systematic test design methods for software installation?
- Where are the cost-effective automatic solutions and tools for software installation?

Based on our recent literature survey, there is a few papers addressing software installation testing. For example, Edward Kit in his book [1] discussed the current status and existing problems in installation testing. In [3], Mark Pawson introduced the Install Shield Test Matrix, which can be useful for testers to identify, document, and select the major focuses in software installation validation. Using this matrix, a tester can select test items and design test cases. However, there are two issues with this matrix. First, there is a lack of detailed engineering guidelines and systematic solutions to help engineers identify and create this matrix. Second, when software with complex configurations are executed under diverse running conditions, generating this matrix manually becomes very complex and too tedious. In addition, testers also need a rational approach and strategy to make cost-effective testing trade-off, measure test complexity, and analyze test cost and coverage for software installation.

As shown in Figure 1, the scope of *software installation testing* can be presented as a 3-dimonional software installation test space, in which the X-axis presents all specified system configurations, the Y-axis presents all of system running conditions before installation, and the Z-axis presents system installation functions. Clearly, well-defined test models and systematic methods are needed to model and present the issues in this space. This paper is written to address the mentioned software installation issues using a model-based approach.

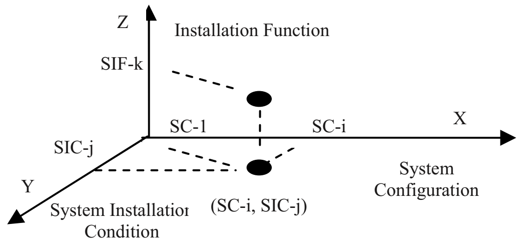


Figure 1 Software Installation Test Space

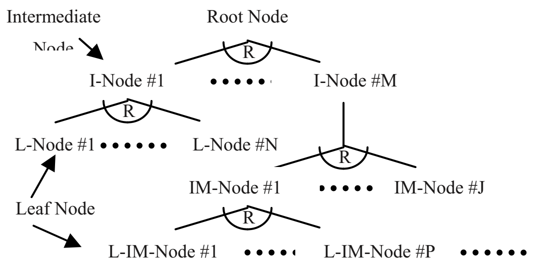


Figure 2 A Semantic Tree Model

3. Software Installation Test Models

This section proposes a new test model for software installation. It can be used by test engineers to analyze, model, and present diverse system configurations, complex running conditions, and various system installation functions during installation test planning. It is a very useful test model for test engineers in software installation and patch validation. As we know, a good test model usually is defined and developed as a base to define test criteria, to develop systematic test generation solutions, and to monitor and measure test coverage.

3.1. A General Semantic Tree Model

The proposed model is a semantic tree model, which can be formally defined as 3-tuple = (N, E, R), where

- N is a set of tree nodes. There are three types of nodes: a) a single root node, b) intermediate nodes, and c) leaf nodes.
- E is a set of tree edges. Each edge is a link, which connects a parent node and child node in a tree.
- R is a set of relations, and each item in R has a semantic label that presents one semantic relation between a parent node and its child node. There are five types of semantic labels. They are: OR, AND, NOT, NAND, and Select-1.

Figure 2 shows an example of the proposed semantic tree model, and Figure 3 shows the notation of the five different semantic labels in a model. Now let us use the rest of this section to show how this general semantic tree model can be used to model and present diverse system configurations, installation conditions, and system installation functions with the detailed semantics.

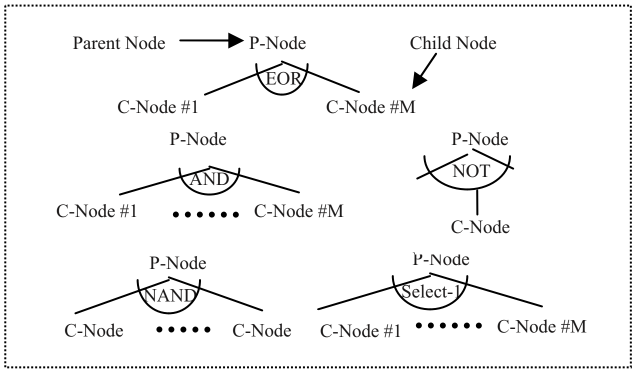


Figure 3 The Notations of Five Semantic Labels

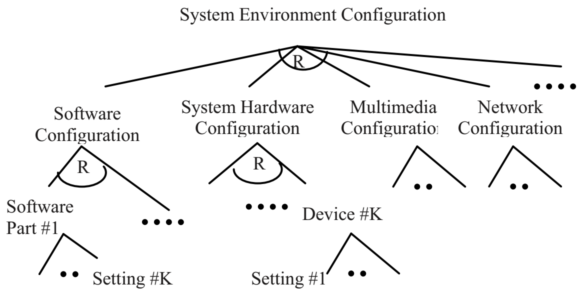


Figure 4 A System Environment Configuration (SEC) Model

3.2. Modeling System Environment and Configurations

Now let's first use the proposed semantic model to present diverse system environments and configurations. We define it as System Environment Configuration (SEC) model. As shown in Figure 4, the model presents various system configurations in a hierarchical tree model. In this model, each leaf node presents one configuration of a hardware (or software) part of the system. Since each hardware/software part may have more than one configuration, or may be equipped with more than one sub-part. Hence, each parent node may have one (or more) child node(s). Their relations can be presented using five types of semantic labels. Table 1 explains the semantics of the five different relations between a parent node and its child nodes. Figure 5 shows the operating system (OS) configurations for a product (QW) as a part of its SEC model.

Clearly, a SEC model for a product presents its all possible system configurations because each system configuration can be presented as a spanning tree of the model. Of course, the spanning tree is driven from a SEC model based on the involved semantics. The details can be found in Sections 4 and 5.

Table 1 Semantic Relations in a SIC Model

Relations	Semantics in a System Environment Configuration Model
EOR	P-Node must be provided and set up with only one of its exclusive parts, which are denoted as two child nodes. In other words, the two parts can't be set up at same time.
AND	P-Node must be provided only when all of its child nodes are set up.
NOT	P-Node must be provided without setting up its specific part, denoted as the only child node.
NAND	P-Node must be provided without the support of some parts, denoted as its child nodes.
Select-1	P-Node can be set up with any of one of its child nodes.

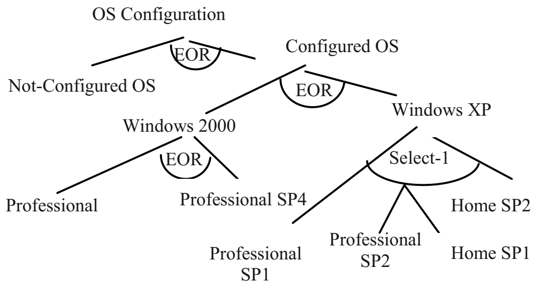


Figure 5 OS Configuration in product QW

3.3. Modeling Installation Conditions

Similarly, we can use the proposed semantic tree model to present diverse system installation conditions. We define it as System Installation Condition (SIC) model. As shown in Figure 6, the model presents various system installation conditions in a hierarchical tree model. In this model, the root node presents the overall condition under the system installation, and it depends on a number of conditions. Each condition, as a parent node, may depend on a number of sub-condition factors as its child nodes.

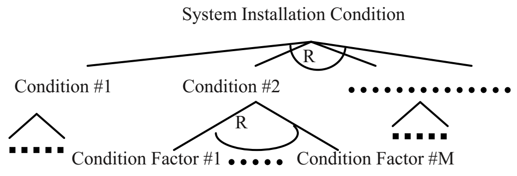


Figure 6 A System Installation Condition Model

Table 2 Semantic Relations in SEC Model

Relations	Semantics in an Installation Condition Model
EOR	P condition holds only when one of its two exclusive sub-conditions (denoted as child conditions) holds.
AND	P condition holds only when all of its child conditions hold.
NOT	P condition holds only when its child condition is not hold.

NAND	P condition holds only when all of its child conditions are not hold.
Select-1	P condition holds when any one of its child conditions holds.

Each leaf node presents a special condition of its parent node (as a condition factor). The semantic relations between a parent node and child node can be presented using the same set of five semantic labels. Table 2 explains the semantics of the five relations between a parent node and its child nodes. Figure 7 displays the SIC model for product QW. It presents related system installation conditions. Similarly, a SIC model for a product (QW) presents its all possible system installation conditions because a combinational system installation condition can be presented as a semantic spanning tree of the SIC model. Of course, the spanning tree is driven from a SIC model based on the involved semantics. The details can be found in Sections 4.

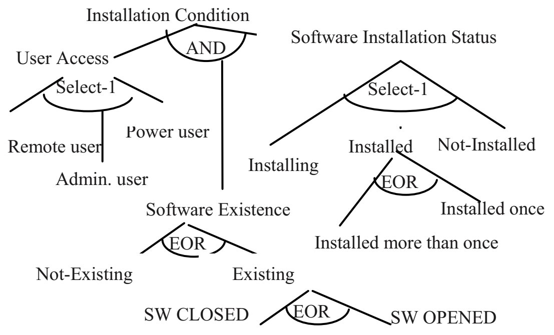


Figure 7 A Sample System Installation Condition Model

3.4. Modeling System Installation Functions

The proposed semantic tree model can also be used to present system installation functions. We define it as System Installation Function (**SIF**) model. As shown in Figure 8, the model presents various system installation functions. The model provides a complete picture about system installation functions in a hierarchical structure, and it depicts all of system installation functions from the top level to the button level. Each high level function can be presented as a parent node, and its low-level functions (or sub-functions) are presented as its child nodes. A similar set of semantic labels are used to represent their relations.

Table 3 explains the semantics of the three relations between a parent node and its child nodes in the SIF model. It should be noticed that in the most cases, a system function usually is supported by a number of its sub-functions together. In other words, only the AND relation exists between a parent node and its child nodes. However, if a system allows users to select, configure, assembly, or customize its function components, then other semantic relations may occur.

Figure 9 displays a part of the SIF model for product QW. It presents related system installation functions. A SIF model can be used to present system functions in two different views. One is a hierarchical function view, like Figure 9. And the other is a functional feature view, which presents a system functional feature in terms of its required system components (or sub-systems), which support the feature. A typical example in a telecommunication system will be caller-ID and conference-call. They are the functional features supported by a number of sub-systems and components.

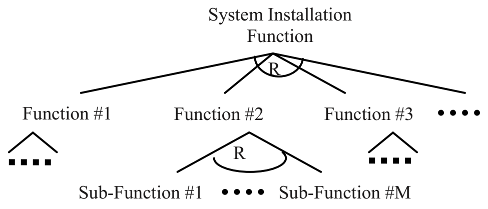


Figure 8 A System Installation Function Model

Table 3 Semantic Relations in a SIF Model

Relations	Semantics in a System Installation Function Model
EOR	The P function is supported only when any of its two exclusive sub-functions (denoted as child nodes) is provided.
AND	The P function is supported only when all of its sub-functions (as denoted child nodes) are provided.
NOT	The P function is supported without its specific sub-function, denoted as the only child node.
NAND	The P function is supported without the support of some parts, denoted as its child nodes.
Select-1	P function is provided when anyone of its sub-functions (denoted as child nodes) is provided.

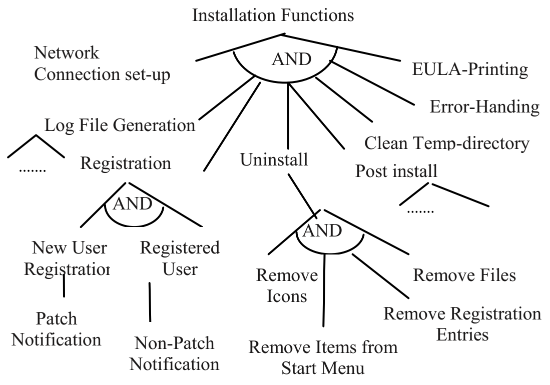


Figure 9 A Sample System Installation Function Model

Clearly, the proposed semantic tree model does provide good information to help engineers to analyze and understand system configurations, running conditions, system functions and its features in a hierarchical way. In many cases, the semantic tree model for system installation functions may only consist of AND relations in parent nodes. However, when a system support customization and configuration functions based on the various function components, the other relations (such as EOR, NOT, SELECT-1, and NAND) may occur. Using this model-based approach, engineers can discover and analyze the test models for each product during the test planning phase. Once the models are generated, they can be updated and maintained to support the system evolution.

4. Installation Test Criteria and Metrics

Based on the proposed three models in Section 3, we define software installation test criteria, complexity, and coverage metrics here. First, let's define a spanning tree for the proposed model.

Semantic Spanning Tree:

A ***semantic spanning tree*** $\mathbf{G_{SPT}}$ is a sub-tree of a given semantic tree $\mathbf{G_{ST}}$, its holds the following **properties**:

- $\mathbf{G_{SPT}}$ must include all parent nodes in $\mathbf{G_{ST}}$.
- For each parent node Npi with an AND (or NAND) relation, $\mathbf{G_{SPT}}$ must includes all of its child nodes and its links connected them.
- For each parent node Npi with an EOR relation, $\mathbf{G_{SPT}}$ must include only one of its child nodes and the corresponding link.
- For each parent node Npi with a Select-1 relation must include only one of its child nodes and the corresponding link.
- For each parent node Npi with a NOT relation, $\mathbf{G_{SPT}}$ must include the only child node and its corresponding link.

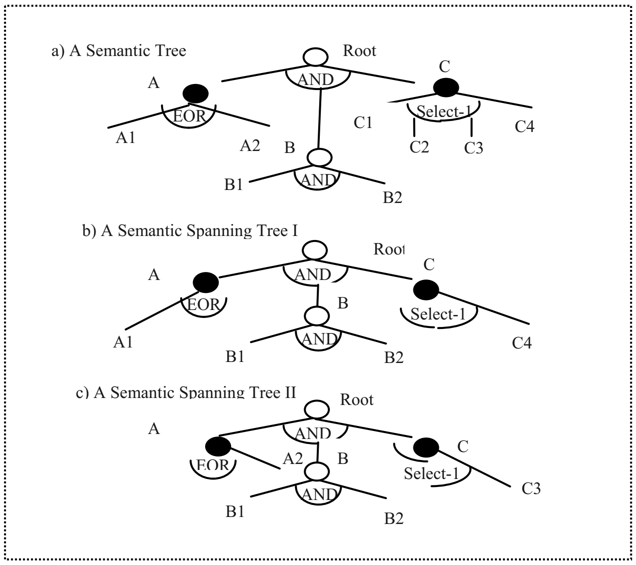


Figure 10 A Semantic Tree Model and Its Spanning Tree

4.1. System Environment Test Criteria and Test Metrics

For a given **SEC** model $\mathbf{G_{SEC} = (N_{SEC}, E_{SEC}, R_{SEC})}$, let's define the system test criteria for the system installation environment and its various configurations. Assume **SEC** is a set of all system environment configurations, and any of its elements, say **SECi**, stands for a specific system environment configuration. Based on the semantic spanning tree concept given before, **SECi** can be represented as a spanning tree of $\mathbf{G_{SEC}}$. Hence, we can define the test criteria for system environment and configurations as follows.

Single System Environment Configuration Test Criterion:

This test criterion only can be achieved when the given test case set T_{IS} has been exercised under the **SECi** setting.

All System Environment Configuration Test Criteria:

This test criterion only can be achieved when the given test case set T_{IS} have been exercised under all system environment configurations. This implies that all elements of SEC have been tested. They refer to all of the spanning trees of G_{SEC}.

System Environment and Configuration Test Complexity:

For given software product P, its system environment and configuration test complexity ($SECT_{Complecxity}$) can be computed as follows: $SECT_{Complecxity}$ = No. of elements in SEC = $|SEC|$ (1)

= No. of different semantic spanning trees in G_{SEC}

Another alternative approach is to use a bottom-up approach to compute $SECT_{complexity}$ from its leaf nodes to parent nodes, until its root node. Section 4.3 provides a detailed algorithm and examples.

System Environment Configuration Test Coverage:

For given software product P, its system environment and configuration test coverage ($SECT_{Coverage}$) can be computed as follows:

$SECT_{Coverage}$ =No. of the covered SEC's elements / $|SEC|$ (2)

4.2. Installation Condition Test Criteria and Test Metrics

For a given SIC model, $G_{SIC} = (N_{SIC}, E_{SIC}, R_{SIC})$, let's define the test criteria and test metrics to address the diverse software running conditions under a given system environment and configuration.

Under a system configuration SECi, let's assume SIC is a set of possible system running conditions before installation, and any of its elements, say SICj, stands for a specific system running condition. Based on the spanning tree concept given before, SICj can be represented as a spanning tree of G_{SIC}.

Single-System Installation Condition Test Criterion:

This test criterion only can be achieved when the given test case set T_{IS} has been exercised under a system installation condition (say SICj) when P is configured as SECi.

All- System Installation Condition Test Criteria:

For a product P configured as SECi, this test criterion only can be achieved when the given test case set T_{IS} have been exercised under all system installation conditions in SIC. This implies that all elements of SIC have been tested. They actually are the spanning trees of G_{SIC}.

System Installation Condition Test Complexity:

For given software product P under a given configured system environment SECi, its system installation condition test complexity ($SICT_{Complecxity}$) can be computed as follows:

$SICT_{Complecxity}$=No. of items in SIC=|SEC| (4)

= No. of different semantic spanning trees in G_{SIC}

Another alternative approach is to use a bottom-up approach to compute $SICT_{complexity}$ from its leaf nodes to parent nodes, until its root node. Section 4.3 provides some detailed algorithm and examples.

System Installation Condition Test Coverage:

For given software product P under a given configured system environment SECi, its system installation condition test coverage ($SICT_{Coverage}$) can be computed as follows:

$SICT_{Coverage}$ =No. of the covered SIC's elements / |SIC| (5)

4.3. Software Installation Function Test Criteria and Metrics

For a given product P, and its software installation function model (SIF), $G_{SIF} = (N_{SIF}, E_{SIF}, R_{SIF})$, let's define the function test criteria and test metrics to cover system installation functions under a given system installation condition SICj for a configured system environment SECi as follows.

Leaf Node Function Test Criterion:

For any leaf node *Ni* in G_{SIF}, this criterion is achieved when the given T_{IS} includes at least one test case, which exercise the corresponding function of *Ni.*

Adequate Leaf Node Function Test Criterion:

For any leaf node *Ni* in G_{SIF}, this criterion is achieved when the given T_{IS} includes an adequate test set, which exercise the corresponding function of *Ni.*

Adequate Parent Node Function Test Criterion:

For any parent node *Npi* in G_{SIF}, including the root node and intermediate nodes, this criterion is achieved only when the given T_{IS} includes an adequate test set for each child node. In other words, all of its child nodes have achieved its adequate test criterion.

```
Semantic-Spanning-Tree($G_{ST}$-Node, $N_{SPT}$)  {
if $G_{ST}$ -Node is a leaf node, then
        $G_{ST}$ –Node → $N_{SPT;}$ // add this leaf node into $N_{SPT}$
    return
else  add $G_{ST}$ -Node into $N_{SPT}$
    switch ($G_{ST}$ -Node's relation)  {
        case 'EOR': pick a $G_{ST}$-Node's child node (say Ci);
                Ci → $N_{SPT}$ // add into $N_{SPT}$
                Add $G_{ST}$-Node's link to Ci → $E_{SPT}$
                Semantic-Spanning-Tree(Ci, $N_{SPT}$); **break;**
        case 'Select-1': pick a $G_{ST}$-Node's child node (say Ci)
                Ci →  $N_{SPT}$ // add into $N_{SPT}$
                Add $G_{ST}$-Node's link to Ci → $E_{SPT}$
                Semantic-Spanning-Tree(Ci, $N_{SPT}$); **break;**
        case 'AND' or 'NADN': $G_{ST}$-Node's child nodes → $N_{SPT}$
                Add all its links to its child nodes → $E_{SPT}$
                Loop for each child node (say Ci) and do:
                Semantic-Spanning-Tree(Ci, $N_{SPT}$); **break;**
        default 'NOT': pick a $G_{ST}$-Node's child node (say Ci)
                Ci →  $N_{SPT}$ // add into $N_{SPT}$
                Add $G_{ST}$-Node's link to Ci → $E_{SPT}$
                Semantic-Spanning-Tree(Ci, $N_{SPT}$); **break;**
        }
}
```

Figure 11 A Procedure to Find a Semantic Spanning Tree

Installation Function Test Complexity:

For a given software product P under a system installation condition SICi based on a configured system environment SECi, its system installation function test complexity ($SIFT_{Complecxity}$) can be computed using a bottom-up approach based on the given G_{SIF}. For any leaf node *Ni* of $G_{SIF,}$ its function test complexity is the number of test cases in its adequate test set. For any parent node Npi of $G_{SIF,}$ its function test complexity can be computed based on its relation with child nodes and the test complexity of its child nodes. Let Cj stands for a child node of Npi.

- If its semantic relation with its child nodes is SELECT-1, then its complexity can be computed below.
$$Npi\text{'s } FT_{Complecxity} = \sum (Cj\text{'s } FT_{Complecxity}) \qquad (6)$$
Where j = 1, …n, Cj is a child node of Npi, n is the number of its child nodes.
- If its semantic relation with its child nodes is OR, then its complexity can be computed as follow.
$$Npi\text{'s } FT_{Complecxity} = \sum (Cj\text{'s } FT_{Complecxity}) \qquad (7)$$
Where j = 1,2, Cj is a child node of Npi.
- If its semantic relation with its child nodes is AND, then its complexity can be computed below.
$$Npi\text{'s } FT_{Complecxity} = \prod (Cj\text{'s } FT_{Complecxity}) \qquad (8)$$
Where j = 1,…,m, m is the number of its child nodes, and Cj is a child node of Npi.

Using a similar approach, engineers can easily identify the $SIFT_{complexity}$ from the leaf nodes of a given semantic tree model to their parent nodes, as well as the root node.

5. Conclusions and Future Work

This paper addresses the needs in modeling and analysis of software installation validation. A model-based approach is used to support systematic modeling and analysis for software installation testing. A semantic model, known as a semantic tree is proposed here to assist engineers to define a sound test model for software installation testing. Comparing with the existing approach, this model provides a rational and systematic means to assist engineers to identify and analyze diverse system environments and configurations, various installation conditions, and installation functional features. Based on this model, installation test items, test cases, test criteria, and test strategies can be easily defined and developed. The details about them will be reported in future publications. The proposed approach has been used in a project at a local software company, and received very positive feedback in test model discovery and establishment.

The future extension of this research includes two parts. The first is to develop more detailed product-oriented test cost metrics to support more effective strategic test planning and measurement. The other is on software installation test automation. Currently, we are developing a software installation automation tool based on the proposed models to support test modeling and analysis, test generations, and test coverage analysis.

References

[1] Kit, Edward (1999) Software Testing in the Real World: Improving the process (5th). England: ACM Press.
[2] Chirs Agruss, "Software Installation Testing: How to automate tests for smooth system installation", Testing & Quality Magazine, Vol. 2, Issue 4, July/August 2000.
[3] Mark Pawson, "The Test Matrix: How one company kept a complex test on track?" (2001). Retrieved at URL: http://www.stickyminds.com/ March 3, 2006.
[4]Ibrahim, K. et al., Model-based Software Testing, Encyclopedia on Software Engineering (edited by J.J. Marciniak), Wiley, 2001.
[5] A. Pretschner, et al., "One Evaluation of Model-Based Testing and Its Automation", Proceedings of 27th International Conferences on Software Engineering (ICSE2005), St. Louis, MO, USA, 2005.
[6] A. J. Offutt and A. Abdurazik, "Generating Tests from UML Specifications". Second International Conference on the Unified Modeling Language (UML99). Fort Collins, CO, October 1999.
[7] S. Fujiwara, et al., "Test Selection Based on Finite State Models". IEEE Transactions on Software Engineering. V. 17, No. 6, pp. 591-603, June 1991.
[8] E. Farchi, A. Hartman, and S. S. Pinter, "Using a Model-Based Test Generator to Test for Standard Conformance", IBM Systems Journal. V. 41, No. 1, pp. 89-110, 2002.
[9] T. S. Chow, "Testing Software Design Modeled by Finite-State Machines", IEEE Transactions on Software Engineering, Vol. 4, No. 3, pp. 178-187, May 1978.

Automatic Test Generation for Database-Driven Applications

Zhenyu Dai
Amazon.com
705 5th Ave. S,
Seattle, WA 98104
206-266-7865
zhenyud@yahoo.com

Mei-Hwa Chen
SUNY at Albany
Computer Science Dept.
Albany, NY12222
518-4424283
mhc@cs.albany.edu

Abstract

Database-driven software has been widely adopted in many areas of software applications. In this type of software, the database is an integrated part of the system. Traditional testing techniques have focused either on the software or on the databases, but ignored the interactions between the two core components of the system. Recently, the importance of testing database-driven applications has gradually been recognized. Testing cannot be considered complete until the software and its interactions with databases are adequately exercised. In this paper, we present an automatic test case generation for testing the interactions between the applications and the databases. The aim is to generate a test suite that can fulfill the requirement of a set of test adequacy criteria that cover def-use associations of database interactions and the boundary points. We present a tool that supports automatic generation of test cases and a case study conducted to demonstrate the validity of the technique and the tool.

1. Introduction

A ***database-driven application*** is a program strongly coupled with external databases. A database is a persistent component of a database-driven application; the behavior of the application is highly dependent on the correctness of the databases and the interactions between the application and its databases. To ensure the reliability of database-driven applications, testing is crucial and must cope with the following complex situations: (1) Modern software is often deployed in a shared resource execution environment in which more than one application may access the same database. Under such circumstances, when one application changes a shared database, other applications may be affected. Testing must take into account the shared database as well as all the applications using the database. (2) Many database-driven applications are designed to be deployed in various execution environments and each of them may use different database management systems (DBMSs). Although there are standard interfaces such as SQL, JDBC, and ODBC, the interoperability between the applications and the DBMSs cannot be fully guaranteed. For example, ORACLE will enforce the "CHECK" statement in database schema while MySQL will not. Thus, testing database-driven applications in every deployment context is necessary. (3) Databases are often frequently updated, and each update may affect the applications that make use of the data. Thus after each update, all the applications using the database must be re-tested. To effectively and efficiently test database-driven applications under these complex situations, there is a need for a testing tool that can automatically generate test cases to fully exercise not only the applications and the databases, but more importantly, the interactions between them.

The importance of testing database-driven applications has recently been recognized. Several unit level testing techniques have been proposed [3] [6] [13]. These techniques focus on testing the database at the unit level to investigate the states of the databases after the execution of a single or a group of SQL statements. These techniques were not designed to detect database faults affecting the application programs, or program faults propagating to databases and to the other applications. To test the interactions between the applications and the databases in database-driven applications, Kapfhammer and Soffa proposed a family of test adequacy criteria based on the data-flow analysis of database entities [9]. These coverage criteria can be utilized to design test cases to exercise the dependences between the databases and the applications. Nevertheless, as in traditional dataflow testing, a significant amount of effort will be required to generate test cases in order to fulfill the coverage requirement if a tool/technique for automatic test case generation is lacking. Existing techniques for automatic test case generation include symbolic execution [1] [4] [10], execution-based techniques [7] [8] [11], and model-based approaches [2] [5] [12]; however, none of these techniques account for database-driven applications.

In this paper, we present an automatic test case generation technique designed to fully exercise the interactions between the applications and their databases. Our test model is to cover a set of database-related dataflow coverage criteria by extending the concept of symbolic execution [10] and to select inputs from the boundary values of critical points. We developed an automatic test case generation tool implementing the technique to demonstrate its feasibility and conducted a case study by using this tool to show the potential strengths of the technique.

The remainder of this paper is organized as follows: Section 2 describes our test model. The technique for test case generation is presented in Section 3 and an automatic test case generation tool is shown in Section 4. A case study is demonstrated in section 5 and we give our conclusions in section 6.

2. The test model

A database-driven application normally contains many database interaction operations. A database interaction operation can be viewed as a computation of a (partial) function from the input space I to the output space O with the database state (i.e., contents of the database) being a part of I or O or both [3]. Our test model is to tackle **database-related faults** that cause the output of database interaction operations to depart from their specifications. Three test coverage criteria are proposed to guide the selection of test cases to expose database related faults, including all-DIPs, all-DUs, and all-constraints. We use a database entity flow analysis to identify all-DIPS and all-DUs, and a boundary analysis to cover all-constraints. Figure 1 shows a high level view of the test model. It takes the database schema in SQL database definition language (DDL) and program source code as inputs, and conducts DE-dataflow analysis and boundary value analysis. The test case generator then generates test cases, and each test case includes values for both input parameters and database state. All these steps in the model are automated in a tool.

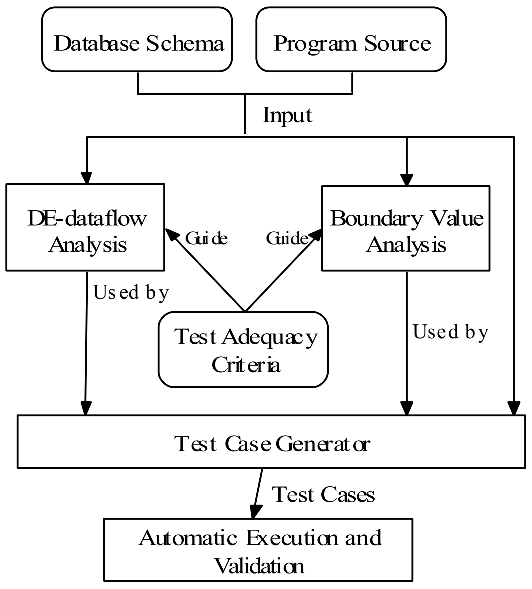


Figure 1: Testing Model

In the following, we first describe the type of database related faults, and then we describe the database entity flow analysis and the boundary value analysis.

2.1 Database-related faults

Database-related faults, in general, can be classified into two categories by the two types of failures they can cause: (1) ***database integrity failure***; (2) ***output inconsistency failure***.

Database integrity failure:

In [9] Kapfhammer and Soffa defined four types of failures relevant to the database integrity. The first two types are related to the database validity. A program can violate the validity of one of its databases if (1-v) it inserts a record into a database that does not reflect the real world; or (2-v) it fails to insert a record into the database when the status of the real world changes. The last two types are related to the database completeness. A program can violate the completeness of one of its databases if (1-c) it deletes a record from a database while the record still reflects the real world or (2-c) the status of the real world changes and the program fails to include this information as a record in the database.

We adopt their definitions of database integrity failures with minor modifications. Under the assumption that when real world changes, the user will conduct a corresponding database interaction operation to commit these changes in database, then type (2-v) can be defined as an insertion operation that fails to insert a record to reflect the real world, and type (2-c) can be defined as a deletion operation that fails to delete a record to reflect the real world. Moreover, in addition to insertions and deletions, modifications of some records can also cause violation of the database integrity. Thus, we define ***database integrity failures*** as the following two types: (1) an operation changes the database to a state that is inconsistent with the real world, which covers types 1-v and 1-c. (2) An operation fails to change the database content to reflect the real world, which covers types 2-v and 2-c.

Output inconsistency failure:

A database interaction operation normally will render the user some outputs based on the contents of the database. For example, if a student conducts a "personal information retrieval" operation, the application should display the student's personal information on the screen; and if an advisor wants to check whether a student has finished a specific course, the application should respond with a "yes" or "no" according to that student's record in the database. The output inconsistency failure occurs when the outputs of a database interaction operation are inconsistent with the database content. For example, if the application responds with a "no" while the student has indeed finished a specific course, it's an output inconsistency failure.

2.2 Dataflow analysis for database entities

The purpose of database entity dataflow (DE-dataflow) analysis is to investigate where a database entity is defined and where the defined value of the database entity is referenced subsequently. A statement contains a **definition** of a database entity if it inserts, deletes, or modifies this database entity, and a statement contains a **use** of a database entity if it references this database entity. A statement containing a definition or use of database entity is called a **database interaction point (DIP)**. Database integrity failures are caused by incorrect modifications of a database, which usually take place at database interaction points. Furthermore, the database entities are the persistent components of an application; thus they can exist before and after the execution of the application program. According to this, we define the root node of the control flow graph of the application program as the **pseudo-definition** of all the database entities. A database entity or its content may be assigned to a program variable, and this variable may be used in a statement different from the statement references this database entity. Therefore, we introduce the notion of **content-use**, which is a statement where a program variable representing a database entity (or some attribute(s) of this entity) is used.

The all-DIPs criterion requires the test set to cover all the database interaction points in the program, and all-DUs requires that, in addition to covering all-DIPs, the test set should cover all the paths between any def-use, def-contentUse, pseudoDef-use and pseudoDef-contentUse pairs of the any database record in the program. Thus, all-DUs criterion subsumes all-DIPs. The coverage of all the def-use and pseudoDef-use pairs is intended to reveal the faults causing database integrity failures, since every possible change to database will be tested together with a subsequent use of the changed content through all possible paths, which have a great chance to detect any incorrect database modification. The coverage of all the def-contentUse and pseudoDef-contentUse pairs is intended to reveal the faults causing output inconsistency failures.

2.2.1 DE-dataflow based path selection

To generate test cases to fulfill the all-DIP and the all-DU criteria, we use a depth-first-search to perform dataflow analysis for database entities and then select paths in the application's system control flow graph that will satisfy the two test adequacy criteria. The CFG is annotated by DE-dataflow information during the search. In the depth-first-search, we maintain a definition/pseudo-definition set that contains all the record definitions/pseudo-definitions with their CFG node numbers in the current path. Every time a use or a content-use node is visited in the search, if it matches any in the definition/pseudo-definition set with a definition-clear path, the current control flow path will be selected. We also select paths to cover all the DIPs that are not covered by any def/pseudoDef – use/contentUse pair. A selected path starts from the root node of the system-CFG and ends in a database-record use or contentUse node. The following steps are performed to select the paths:

Step1. We maintain a definition/pseudo-definition set for the current path during the search. Specifically, when a node is added (or removed) into (or from) the current path during the search, if this node is a def/pseudo-def of database record, we insert (or delete) this node into (or from) the set with corresponding def/pseudo-def information.

Step2. If a visited node is a use/content-use of database record, then we find all the nodes in the def/pseudo-def set that have a definition clear path to the current node, and mark the corresponding paths as selected paths.

Step3. After the depth-first-search is completed, we do a second run of depth-first-search. In this run, when a DIP node that is not covered by the paths selected by the first run is visited, we mark the current path (from the root node to the current node) as the selected path. This is to ensure that every DIP will be covered, because some DIP may not be in any pair of def/pseudo-def and use/content-use.

To determine the number of iterations in the presence of loops in path selection, we use a commonly used technique in tradition dataflow analysis, which assumes a loop can only iterate up to two times. This assumption uses "two" to represent multiple times, and thus gains efficiency in testing by sacrificing some completeness.

2.3 Boundary value analysis

Considering that database integrity failures may also be caused by violation of database constraints, simply covering a path may not expose the fault. **All-constraints** criterion complements the all-DUs and all-DIPs to catch the constraints violation faults, which requires that for each "definition" type DIP the test set should exercise all the conditions to ensure that the DIP does not violate any database constraint, and should exercise all the possible database constraint violation conditions for this DIP. The database constraints can be classified into four types: (1) database attribute type and length constraints; (2) " UNIQUE" , " NOT NULL" and " DEFAULT" constraints; (3) primary key and foreign key constraints; (4) constraints in "CHECK" statement. To satisfy the all-constraints criterion, we conduct boundary value analysis to complement our DE-dataflow analysis in test case generation. Boundary values are the boundaries of the input domain which satisfies the database constraints. We first find the boundary values for both database entities and input parameters based on the database constraints, and then partition the input domain into sub-domains based on the boundary values. In test case generation, we generate test cases to exercise sub-domains and the boundaries of sub-domains.

Boundary value analysis focuses on the boundary of the input space to identify test cases. The rationale behind boundary value testing is that "errors tend to occur near the extreme values of an input variable." In boundary value analysis, a boundary value can also be used to partition the input domain into sub-domains, and each sub-domain should be tested at least once. We define that a boundary value for a variable or database attribute as a value such that the program may have different behaviors (or results) in the following three different cases: (1) the variable/attribute is assigned to this value; (2) the variable/attribute is assigned to a larger value; (3) the variable/attribute is assigned to a smaller value. In a database-driven application, we can get boundary values for database attributes by the following three ways: (1) from database constraints in the database schema; (2) from the SQL queries in the application; (3) from "general boundary values" of different data types.

We use the following three rules to get boundary values for database attributes from a database schema: (1) the value specified by " DEFAULT" constraint should be a boundary value of the corresponding attribute. (2) The values specified in " CHECK" statements should also be boundary values. (3) If an attribute is a string, the length of this attribute should also have boundary values.

In addition to the boundary values found from database schema, the database attributes should also have "*general boundary values*" dependent on their data types: (1) " NULL" should be a boundary value for all data types. (2) " 0" should be a boundary value for numeric types (integer, double, float, etc.). (3) "TRUE" and " FALSE" should be boundary values for Boolean type. (4) Empty string should be a boundary value for a string type (i.e., the length of the string is 0).

After we find the boundary values for all the database attributes, the boundary values for the input parameters can be easily found. First, we introduce the notation "***host variable.***" Host variables are the "variables in the host language that are used as parameters in SQL queries" [3]. If an input parameter is a host variable, its boundary values are the same as those of the corresponding database attribute. If not, its boundary values are just the "general boundary value*s.*"

In [3], Chays et al., proposed a similar approach to find boundary values for database attributes, but their analysis is only limited to the "check" statement in database schema and SQL queries in program.

3. Test Case Generation

In this section, we present a technique for automatic test case generation, which generates test cases not only exercising the selected paths that satisfy the first two test adequacy criteria, but also covering the all-constraints criterion. Each test case includes all the input parameter values and a database state.

For each selected path, we calculate the ***generalized path condition (GPC)***; the notion of GPC is inherited from the symbolic execution methodology proposed in [14]. In a symbolic execution, symbolic values are used instead of real values for input parameters, and the values of program variables are represented by symbolic expressions during execution and a path condition (PC) is calculated. The GPC is an extension of PC by treating database state as a special input parameter. It is the condition that both the normal and the special input parameters must satisfy for a specific path to be executed. It is calculated by combining the initial conditions and all the branch conditions in a path by using a logical "AND".

The test cases satisfying GPC and covering the all-constraints criterion are generated by using a ***sub-domain generation*** technique. In this technique, the domain of each input parameter defined by GPC is first partitioned into several sub-domains by the boundary values of that input parameter. Then one value is randomly chosen strictly within each sub-domain (i.e., not on the boundary of any sub-domain) and is put into the *candidate value set* of that input parameter. The intuition is that the values from different sub-domains may have different violation conditions for a database constraint. The boundaries of all the sub-domains within GPC should also be put into the candidate value set no matter whether the boundaries are open or close, since boundaries are usually more fault-sensitive. A test case can be generated by extracting one value from each input parameter's *candidate value set*. We provide four value-combination rules that can be used in sub-domain generation: (1) All the candidate value combinations should be exhausted. (2) Each candidate value should appear at least once. (3) For every two input parameters, every pair of candidate values should appear at least once. The last rule is called *pair-wise-coverage* in [5]. (4) The user provides a normal test case that represents the normal running of an application; in each generated test case, one and only one input parameter will have different value from the normal test case, and each candidate value should appear at least once in the test set. The first rule will generate the largest number of test cases and the second rule will generate the smallest number of test cases.

The sub-domain generation cannot be directly used when the GPC is *non-statically defined*, i.e., the domain of some input parameter defined by GPC depends on the values of other input parameters. For example, in GPC " $x>y$ and $x<3$" , the domain of x is dependent on the value of y, hence is non-statically defined. To make the GPC suitable to be partitioned, we develop a ***domain reduction*** technique to generate a statically defined subset of the original non-statically defined GPC. The basic idea of domain reduction is as follows. We first randomly choose an input parameter whose domain is non-statically defined in GPC, and assign to it a random value that does not conflict with GPC. Then we check whether the reduced GPC is statically defined. This process will repeat until we get a statically defined subset of GPC or an empty set. We usually give more than one run of domain reduction for a GPC and union the reduced GPC from each run into the final reduced GPC. For example if the GPC is " $x<y$ and $x>3$" , y may first be assigned a value " 9" , and the GPC is reduced to "$y=9$ and $x<9$ and $x>3$". It's now statically defined and domain reduction will stop. If the final reduced GPC is empty, we will ignore the corresponding GPC, since it is hard to find inputs satisfying this GPC.

The test cases generated by sub-domain generation may not cover the following types of database constraints: type, "UNIQUE" , primary key, and foreign key constraints. So, after sub-domain generation, we will check if these four constraints have been covered by the generated test cases for each "definition" type DIP. If not, additional test cases will be generated to cover them. The new test cases will be generated by modifying existing test cases from sub-domain generation. We call this technique "***incremental generation"***, since the new test cases are incrementally generated by modify existing test cases. To cover the type constraint for a DIP, we choose an existing test case covering this DIP, and modify the value of the input parameter to be a different type. For example, we give a string value "Mike" to the "age" input parameter, which should be an integer. To cover a "UNIQUE" or primary key constraint, we choose an existing test case covering this DIP, and modify its database state to contain the to-be-inserted (or to-be-modified) value for the attribute with "UNIQUE" or primary key constraint. For example, if a DIP will insert a record in STUDENT table, and an existing test case covering this DIP, studentId = 001, studentName = "Mike", STUDENT table is empty, where studentId and studentName are normal input parameters. We will generate a new test case: studenId = 001, studentName = "Mike", student record with (PK: 001) exists. Similar idea is used to generate new test case to cover foreign key constraint.

Each generated test case contains the path id, the values for input parameters and the database state, and is recorded in an XML file. A sample test case for getting patient by patient id is presented in Figure 2.

```
<testCase   id="1"   pathId="1">
      <input>
            <patientId>112938765</patientId>
      </input>
      <databaseState>
            <condition1>
                  <table>PATIENT</table>
                  <PK>112938765</PK>
                  <exist>true</exist>
                  <attribute>
                        <status>true</status>
                  </attribute>
            </condition1>
      </databaseState>
</testCase>
```

Figure 2: A Sample Test Case

4. An automatic test case generation tool

We have developed a tool that implements the test case generation algorithm to automatically generate test cases. The tool is written in Java and currently only supports Java database applications. As shown in Figure 3, the input for the tool is the control flow graph annotated by database entity def-use information (in XML format), host variables in the target application (in XML format), and the database schema (in formal database definition language). The path selector will parse the control flow graph to select the paths and calculate the GPC. And the database schema parser will analyze the database schema to obtain the database constraints. The host variables and the results from the path selector and the database schema parser will be sent to the test case generator for further processing.

The test case generator consists of three components: a boundary value generator, a candidate value generator and a test case organizer. The boundary value generator generates the boundary values for all the input parameters and the database attributes. These boundary values and the GPC are used by the candidate value generator to do domain partition and then generate candidate values for each input parameter and database attribute. Finally, the test case organizer will combine candidate values from different input parameters and database attributes to form a set of test cases, based on the value-combination rule the user chooses. The generated test cases are also in XML format.

The test case generator consists of three components: a boundary value generator, a candidate value generator and a test case organizer. The boundary value generator generates the boundary values for all the input parameters and the database attributes. These boundary values and the GPC are used by the candidate value generator to do domain partition and then generate candidate values for each input parameter and database attribute. Finally, the test case organizer will combine candidate values from different input parameters and database attributes to form a set of test cases, based on the value-combination rule the user chooses. The generated test cases are also in XML format.

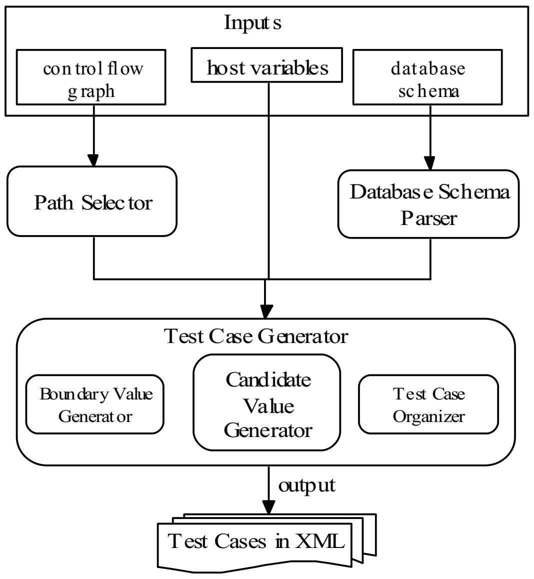


Figure 3: Test Case Generation Tool

5. A case study

We conducted a case study to demonstrate the validity of our test case generation technique. The target application used in our case study is Patient Keeper, which was implemented in Java and uses MySQL as database system. It contains 8 operations that a doctor can perform, and all of them need to access the database (through JDBC): (1) Create doctor; (2) Delete doctor; (3) Create patient; (4) Delete patient; (5) Update patient's room and bed; (6) Get patient by doctorId; (7) Create visit history; (8) Get visit history by patientId. It contains 14 DIPs (database interaction point), including 7 definitions and 7 uses, and contains 9 content-use points. And there are around 600 lines of codes and 82 nature faults.

We created the control flow graph and the host variable file and use them together with the database schema as inputs of our tool. We choose the fourth value-combination rule when running the tool, which can isolate the root cause of different faults; thus we can get a more accurate estimate of the number of faults and the root cause of the faults. The tool selected 15 execution paths and generated 199 test cases and 81 faults were detected by executing the 199 test cases. We classify the faults into the following 9 types: (1) Empty string or NULL is assigned to primary key, UNIQUE or NOT NULL attribute; (2) Violation of the Uniqueness for primary key; (3) Violation of length constraint of database attribute; (4) Violation of foreign key constraint; (5) MySQL is lack of support for part of the database definition language: CHECK constraint and CHAR length constraint is not working; (6) Truncation of decimal number by MySQL makes the application fail; (7) Incorrect handling of type mismatch between input parameter and database attribute; (8) Incorrect handling of deletion (or updating) for non-existing record; (9) Escape characters from the input causes the SQL statement fail. Type (1), (2), (3), (4), (5), (6), (7), and (9) can cause database integrity failure, since these faults can cause the database state to not reflect the real world. Types (7) and (8) can cause output inconsistency failure, since they may cause the message showing to the user (or customer) to be inconsistent with the values in database. The faults detected are shown in Table 1.

Table 1: fault analysis

Type	1	2	3	4	5	6	7	8	9	Total
Fault	25	3	12	6	9	8	8	3	7	81

The tool missed one fault, that is, if the input contains a single quote and the input parameter is used in SQL query, it may cause an SQL syntax error since SQL uses single quotes to specify a string. We can modify our tool to catch this type of fault by summing up all the escape characters for SQL and inserting them into the input parameters in the test case. The most significant result of our tool is that it detected all the faults caused by the interoperability between the applications and the DBSMs, the type (5) faults. For example, it found that all the CHECK constraints and the CHAR length constraints are not working in MySQL.

Because the tool can generate the test cases automatically, it saves a significant amount of time for the QA for analyzing the application and creating test cases.

6. Conclusions

We have presented a methodology for automatic test generation for database-driven applications. Our test model includes a set of test adequacy criteria and boundary value analysis. By using the notion of symbolic execution, our methodology can generate test cases to fulfill the test adequacy criteria and to fully exercise the domains partitioned by the boundary values.

We have developed a software tool to support the automatic generation of the test cases. With the availability of this tool, the effort for testing database-driven application can be significantly reduced, while the quality of this type of application can be greatly improved.

7. References

[1] R. S. Boyer, B. Elspas and K. N. Levitt. Select – a formal system for testing and debugging programs by symbolic execution. *ACM SIGPLAN Notices*, volume 10, issue 6 (June 1975), pages: 234 – 245.

[2] L. Briand, and Y. Labiche. A UML-based approach to system testing. *Technical Report SCE-01-01*, February 2002, Carleton University, Ottawa, Ontario, Canada.

[3] D. Chays, Y. Deng, P. G. Frankl, S Dan, F. Vokolos and E J. Weyuker. AGENDA: A test generator for relational database applications. *Technical Report TR-CIS-2002-04* (8/08/2002), Polytechnic University, New York City, NY.

[4] A. Coen-Porisini, G Denaro, C Ghezzi and M. Pezze. Using symbolic execution for verifying safety-critical systems. In *Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 142 – 151, Vienna, Austria, September 2001.

[5] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, B. M. Horowitz. Model-based testing in practice. In *Proceedings of the 21st International Conference on Software Engineering*, pages 285 – 294, Los Angeles, CA, May, 1999.

[6] Y. Deng, P. Frankl, and D. Chays. Testing database transactions with AGENDA. In Proceedings of the 27th International Conference on Software Engineering, pages 78 – 87, St. Louis, MO, May, 2005.

[7] R. Ferguson and B. Korel. The chaining approach for software test data generation. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, volume 5, issue 1 (January 1996), pages: 63 – 86.

[8] N. Gupta, A. P. Mathur, and M. L. Soffa. Automated test data generation using an Iterative Relaxation Method. In *Proceedings of the 6th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 231 – 244, Lake Buena Vista, Florida, November 1998.

[9] G. M. Kapfhammer and M. Soffa. A family of test adequacy criteria for database-driven applications. In *Proceedings of the 9th European Software Engineering Conference held jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 98 – 107, Helsinki, Finland, September 2003..

[10] J. C. King. Symbolic execution and program testing. *Communications of the ACM*, volume 19, issue 7 (July 1976), pages: 385 – 394.

[11] B. Korel. Automated software test data generation. *IEEE Transactions on Software Engineering*, volume 16, issue 8 (August 1990), pages: 870 – 879.

[12] J. Ryser, and M Glinz. Using dependency charts to improve scenario-based testing. In *Proceedings of the 17th International Conference on Testing Computer Software (TCS2000).* Washington D.C., June 2000.

[13] J. Zhang, C Xu, and S. C. Cheung. Automatic generation of database instances for white-box testing. In *Proceedings of 25th International Computer Software and Applications Conference*, 2001.

Fault-Based Testing of Data Schemas

Maria Claudia F. P. Emer*
State University of Campinas
mcemer@dca.fee.unicamp.br

Silvia Regina Vergilio
Federal University of Paraná
silvia@inf.ufpr.br

Mario Jino
State University of Campinas
jino@dca.fee.unicamp.br

Abstract

Data are manipulated in several applications and can be involved in critical operations. To have correct data in these operations is fundamental. Data schemas are used to define the structure and relationships among data. One way to ensure the quality of data defined by schemas is to test them by using schema specific testing approaches, criteria and tools. However, few works address this subject. To fulfill this demand, we present a testing approach based on typical fault classes identified for data schemas. This testing approach provides means to reveal faults related to incorrect or absent definition of constraints for the data in the schema. The approach includes the automatic generation of a test set which contains data instances and queries to these instances. Data instances and queries are created according to patterns defined for each fault class. The testing approach usefulness is shown by examples from the context of databases, XML Schema documents and XML based components, such as web services.

1. Introduction

Testing is an important activity in software development for contributing to generate reliable products and to evaluate their quality [17]. Moreover, testing involves many correctness aspects in software applications. For example, we should consider the code to be tested, the operational environment, and data.

Schemas are used and designed according to the data specification to describe how the data can be stored in an application. Reliable data is essential in software applications because incorrect data can cause failures. To ensure data integrity and accuracy in the application, testing schema is fundamental and, in this sense, specific testing approaches and tools are necessary.

Most testing approaches are applied only for testing the applications that manipulate the schema. For example, works on testing of web applications have been reported in the literature [12, 14, 24], but few papers address schemas testing. Li and Miller [13] propose mutation operators for XML schemas; however, they do not use these operators to generate test cases and do not present a testing process. Franzotte and Vergilio [10] introduce mutation operators and apply the mutation analysis criterion on XML schemas; however, the test data in the testing process are generated manually and the operators introduced can be applied only for XML Schemas.

In the database context, there are various works investigating the test of database applications. Most of these works, however, focus on the test of database applications involving data generation, application and design [2, 4, 5, 7, 11, 21, 25]. Some of them address testing application using information from database schemas [3, 18] without exploring the test of schemas.

Considering the importance of ensuring the consistency of data defined in the schemas, we have introduced a fault-based testing approach for XML schemas [8, 9]. In addition to that, there are other schema contexts, where this testing approach can be applied, such as relational databases, object-oriented databases, object relational databases, communication among web components, as well as web services. To allow schema testing in these other contexts, we now propose a generic fault-based testing approach by presenting a representation for data models based on MOF (Meta-Object Facility) Specification [15, 16]. MOF is used to identify schema components. We also define generic fault classes that represent typical faults that may be found in diverse types of data schemas.

The remainder of this paper is organized as follows. Section 2 introduces the testing approach for data schemas. Section 3 shows the application contexts our approach. Section 4 describes related work. Section 5 contains the conclusions and future work.

2. The Testing Approach

This section describes the testing approach. We first introduce a model to represent the schemas to be tested and to allow the identification of its elements. The use of such model makes the approach generic, that is, the approach can be applied to any schema that can be represented by this model.

Common faults introduced during the creation of a schema are organized into fault classes. A formal representation is used to identify fault classes. Fault classes guide the generation of data instances and queries for these instances. Data instances represent the potential faults in the schema according to the fault classes. Queries are able to

* This work is partially supported by CNPq (Brazil).

detect the faults related to fault classes in the schema under test.

The testing approach aims to discover faults according to fault classes in data schemas. These faults address incorrect or absent definition of constraints for data in the schema. The idea is to avoid incorrect data to be considered correct or correct data to be considered incorrect, causing failures in the application that manipulates them.

2.1. Data Model

A data model may be defined as a collection of data related to each other, described by means of elements, attributes, constraints and relationships.

In our testing approach the data model is represented by a metamodel C. This metamodel is defined based on MOF (Meta-Object Facility) Specification [15, 16], composed of four layers. Figure 1 shows that the metamodel C corresponds to layer M2.

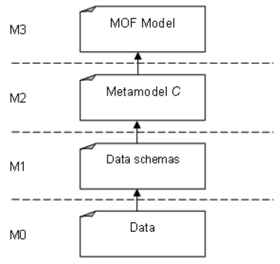


Figure 1. Metamodel C and the MOF model

Figure 2 illustrates the Metamodel C, described in UML notation [1], consisting of the following classes: Element (elements or entities); Attribute (elements' properties) and Constraint (restrictions associated to elements and attributes).

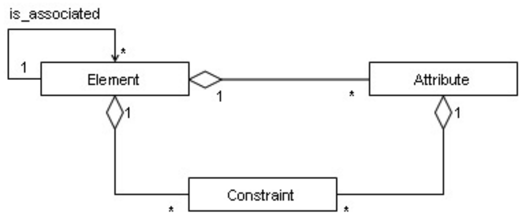


Figure 2. Metamodel C

2.2. Fault Classes

Four fault classes for data schemas were determined. They are associated to typical mistakes that may occur during the schema design. These classes are subdivided into fault types, as shown next.

Class 1 (C1) – Domain Constraints: faults related to domain definition of the element or attribute values.

- Incorrect Data Type (IDT): incorrect definition of data type;
- Incorrect Value (IV): incorrect definition of default or fixed value;
- Incorrect Enumerated Value (IEV): incorrect definition of the list of acceptable values;
- Incorrect Maximum and Minimum Values (IMMV): incorrect definition of upper and lower bounds values;
- Incorrect Length (IL): incorrect definition of number of characters allowed for values;
- Incorrect Digits (ID): incorrect definition of total amount of digits or decimal digits for numeric values;
- Incorrect Pattern (IP): incorrect definition of sequence of characters or numbers allowed for values;
- Incorrect White Space Characters (IWSC): incorrect definition of how white space characters must be treated.

Class 2 (C2) – Definition Constraints: faults related to attribute definition concerning data integrity

- Incorrect Use (IU): the attribute is defined incorrectly as optional or obligatory;
- Incorrect Uniqueness (IN): the attribute is defined incorrectly as unique;
- Incorrect Key (IK): the attribute is defined incorrectly as key.

Class 3 (C3) - Relationship Constraints: faults related to relationship definition among elements.

- Incorrect Occurrence (IO): incorrect definition of number of times a same element may occur;
- Incorrect Order (IR): incorrect definition of the order elements may appear;
- Incorrect Cardinality (IC): incorrect definition of number of occurrences of an element in relation to other element according to a relationship;
- Incorrect Generalization/Specialization (IGS): incorrect definition of a generalization/ specialization;
- Incorrect Aggregation (IA): incorrect definition of an aggregation;
- Incorrect Associative Element (IAE): incorrect definition of an associative element.

Class 4 (C4) – Semantic Constraints: faults related to constraints definition in relation to data content expressed by business rules.

- Incorrect Condition (ICO): incorrect definition of predicate expressed for a condition that must be satisfied by attributes.

2.3. Formal Representation

The formal representation of data schemas provides the identification of the elements, attributes, constraints and associations among them. In the testing approach, it is necessary to generate data instances and queries to reveal faults in data schemas.

A data schema S is denoted by $S = (E, A, R, P)$, where:

- E is a finite set of elements (or entities);
- A is a finite set of attributes;
- R is a finite set of constraints concerning domain, relationship and semantics associated to the elements and attributes;
- P is a finite set of association rules among elements, attributes and constraints represented by:
    - $e(x_1\, x_2 \ldots x_i) \mid e \in E,\ x_1\, x_2 \ldots x_i \in E \vee x_1\, x_2 \ldots x_i \in A,$ $1 \le i \le n$, where n is the number of elements or attributes;
    - $x(r_1,\, r_2, \ldots, r_i) \mid x \in E \vee x \in A, r_1\, r_2 \ldots r_i \in R, 1 \le i \le n$, where n is the number of constraints;
    - $e(r_1 : x_1\, x_2 \ldots x_i) \mid e \in E, r_1 \in R,\ x_1\, x_2 \ldots x_i \in E \vee$ $x_1\, x_2 \ldots x_i \in A,\ 1 \le i \le n$, where n is the number of elements or attributes. The notation $e(r : x)$ is used to indicate an association between e and x by a constraint r.

2.4. Testing Process

The testing process for the testing approach is shown in Figure 3.

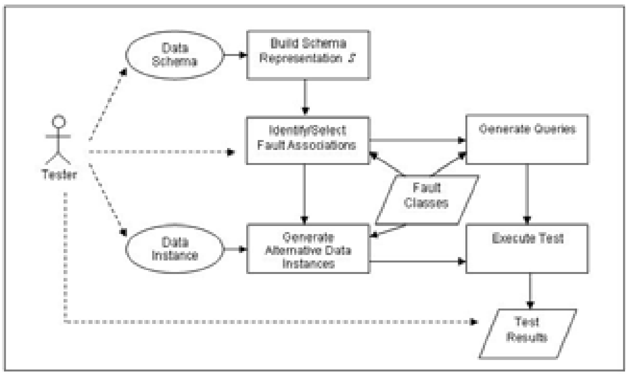

Data Schema
Build Schema Representation S
Tester
Identify/Select Fault Associations
Generate Queries
Fault Classes
Data Instance
Generate Alternative Data Instances
Execute Test
Test Results

Figure 3. Process for Testing Data Schemas

The testing process begins with the tester providing the schema under test and the corresponding data instance (original data instance). The representation for the schema, *S*, is built. Based on *S* fault associations are generated. They relate elements, attributes and elements or attributes constraints to the fault classes, presented previously. Fault associations are identified automatically or by the tester. *S* contains the association rules between elements, attributes and constraints. These association rules are used to establish the fault associations automatically. Other associations that are not present in the schema and consequently not in *S* can be identified by the tester. The tester, considering the specification, can select elements or attributes in the schema and associate them to fault classes. These associations are related to absent constraint definitions.

After the fault association selection, the alternative data instances are generated through single modifications to the original data instances. These modifications are made by insertions and changes to the original data instances according to patterns defined for the fault classes. The fault associations guide the generation of the alternative data instances by indicating the schema element to be modified in the original data instance and the fault class that defines the modification patterns that should be applied. Thus, the alternative data instances represent possible faults in the schema.

The generated alternative data instances are separated into valid or invalid data instances with respect to the schema under test; that is, an alternative instance generated by modification patterns may not be in conformity with the schema under test. Invalid alternative instances are not queried. However, they can be used by the tester during the analysis of test results.

The fault associations also guide the generation of queries. The queries are automatically generated according to the query patterns associated to each fault class that can be detected in the schema.

Test data are formed by a valid alternative data instance and a query to this alternative instance. The specification of expected result for test data (alternative instance + query) is obtained from data specification. The test data are executed and test results are compared with expected ones by the tester. If the results differ, a fault was revealed by the testing approach and the schema needs to be corrected.

3. Applying the Approach

In this section, testing contexts and examples of using the fault-based testing approach are shown.

3.1. XML [22]

The testing approach in the context of XML schemas may be applied to web applications to reveal faults in XML schemas in different situations:

- XML documents used to store data as a database;
- XML messages used to change information in web applications;
- Query results of database in XML format;
- XML documents updated due to alterations in the specification of the data stored in these documents.

To illustrate the testing approach in this context consider a schema of XML documents related to order data of a store, written in XML Schema [23]. Figure 4 shows the data schema in UML based on the metamodel *C* described previously. A fragment of this schema is illustrated in Example 1. This fragment defines the attribute *orderID* of an order. According to the data specification, *orderID* contains six digits and is required.

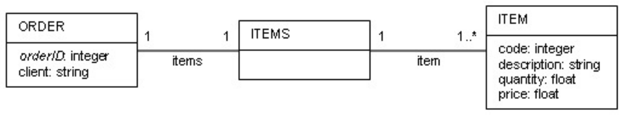


Figure 4. Data Schema for Order

Example 1. XML Schema fragment for order

```
…
<xsd:attribute name="orderID">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="0"/>
    <xsd:maxInclusive value="999999"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:attribute>
…
```

Example 2 illustrates a fragment of the formal representation S, described previously, for the data schema of Example 1.

Example 2. A fragment of $S = (E, A, R, P)$ for Example 1

E = {*order, client, items, item, code, description, quantity, price*}
A = {*orderID*}
R = {*order, type, occurrence, maximum/minimum*}
P = {*order(orderID client items, order: client items),*
orderID(type, maximum/minimum),
…}

Example 3 presents a sample of the original data instance in an XML document related to attribute *orderID*.

Example 3. A sample of content for the attribute *orderID*

```
…
<order orderID = "000134">
...
```

The fault associations among *orderID* and the fault classes Incorrect Data Type (C1-IDT) and Incorrect Maximum and Minimum values (C1-IMMV) of the Domain Constraints Class (C1) would be determined for attribute *orderID* through formal representation S of the order schema. Moreover, the tester could add the fault associations among *orderID* and the fault classes Incorrect Digits (C1-ID) of the Domain Constraints Class (C1) and Incorrect Use (C2-IU) of the Definition Constraints Class (C2). The last two fault associations would be included to reveal faults related to the absence of constraint definitions in the data schema.

The four fault associations for *orderID* are used to generate alternative data instances and queries for these alternative instances. An example of alternative data instance for order by modifying *orderID* relative to the fault class C1-ID is presented in Example 4.

Example 4. An example of alternative content for *orderID*

```
…
<order orderID = "00013">
...
```

Queries are executed on the alternative instances and the results are compared with the data specification. This leads to revealing faults described by the proposed classes. Two faults are revealed with respect to attribute *orderID*: Incorrect Use, because the attribute was defined in the schema as optional and should be required and, Incorrect Digits, because the order number can contain less than six digits and should accept exactly six digits.

3.1.1. Web Service

The approach can also be applied to the communication among web components based on XML, such as web services.

Web service is a set of functions available to remote applications in the web. The communication between the web service and the applications happens basically through SOAP, which is a protocol for exchange of information written in XML. In this way, the testing approach for schemas could be applied to schemas of these messages, used to interchange data among web applications and web services, with the goal of ensuring the integrity of the data that are being manipulated by web services. Hence, if data manipulated by web services are reliable and some failure happens in the response processing from a web service, the testing schema could help in discovering faults in the processing of data by a web service.

3.2. Databases

In the context of database schemas [20], the testing approach may be applied to relational databases and to other types of database models, such as object-oriented or object-relational databases.

Consider a data schema of a relational database from [19], based on the entity-relationship model [6], which contains data about a rural cooperative. Figure 5 shows a fragment of this data schema in UML. A fragment of this schema, written in SQL DDL, is presented in Example 5.

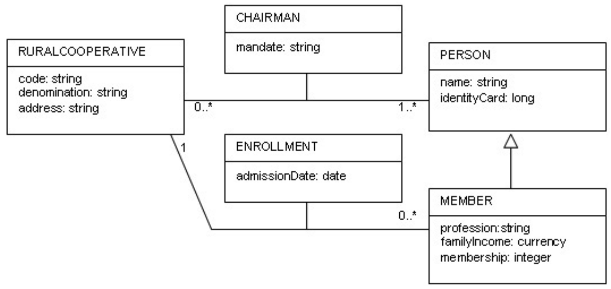


Figure 5. A Fragment of the Data Schema for a Rural Cooperative

Example 5 presents the schema fragment related to entity *member* of the cooperative database. This entity contains the attributes: *profession*, *family income* and *membership.* The attribute *membership* is obligatory. A fragment of the formal representation S for the data schema of Example 5 is shown in Example 6.

Example 5. Database schema fragment for rural cooperative

```
…
CREATE TABLE cooperative.member (
  profession varchar(100) NULL,
  familyIncome FLOAT NULL,
  membership INTEGER NULL
);
…
```

Example 6. A Fragment of $S = (E, A, R, P)$ for Example 5

E = {*ruralcooperative, person, member, chairman, …*}
A = {*code, denomination, address, name, identityCard, profession, familyIncome, membership, mandate, admissionDate, …*}
R = {*type, cardinality, use, …*}
P = {*ruralcooperative(code denomination address, cardinality: person, cardinality: member),*
 …
 member(profession familyIncome membership, cardinality: ruralcooperative),
 …
 membership(type, use),
 …}

Two fault associations can be identified for the attribute *membership* through formal representation S: the attribute *membership* with the fault class Incorrect Data Type (C1-IDT) and the attribute *membership* with the Incorrect Use (C2-IU). These fault associations guide the generation of alternative data instances with modifications in the records of the original data instance according to the patterns defined for the fault classes C1-IDT and C2-IU. An example of a modification based on C2-IU in the attribute *membership* would be to set its value to *null* in a record of the alternative data instance.

After executing the queries generated for each alternative instance according to the fault associations, the fault class C2-IU would be revealed for attribute *membership*; the attribute was defined in the schema as optional and the result was not the expected one, because the attribute *membership* is required according to data specification.

4. Related Work

There are several works reported in the literature on testing of database applications; a few of them use information on the schema to test databases. Chan and Cheung [2] propose test data generation for testing database applications by using white box testing and considering SQL semantics. Chays et al. [5] propose an approach to generate test data for database applications. AGENDA [4, 7] is the tool that supports the test of relational database applications in this approach. Kapfhammer and Soffa [11] propose test criteria based on data flow to verify the quality of the test set to test database applications; they consider the application and its interaction with the relational database through information on the persistent data flow. Suárez-Cabal and Tuya [21] propose a coverage measurement for SQL statement to be used as an adequacy criterion in database application testing. Zhang et al. [25] propose to automatically generate database instances for structural testing of database applications. These authors propose approaches to generate test data to test database applications considering the application, interactions with the database, queries and transactions executed on the database. Differently, our approach aims to test the database schema. The goal is to obtain high reliability and to ensure integrity of the data stored in the database.

Proposals which involve the database schema in testing are: Robbert and Maryanski [18] use information from the database schema to generate a test plan for the database application; Chan et al. [3] propose a fault-based approach to test SQL statements of a database application using information captured from the conceptual data model to generate SQL statement mutants. Our approach is fault-based and uses database instances and queries to test database schemas.

In the context of XML, there are studies that investigate the use of fault-based testing to validate data interactions by using XML messages. Lee and Offutt [12] propose operators to apply mutation testing in XML for data interaction validation among web-based software system components; Offutt and Xu [14] use data perturbation in XML to generate test cases for web services where communication is made through SOAP messages; Xu et al. [24] test communication among web services by modifying XML schema using mutation operators to generate incorrect XML messages. All these proposals do not address the test of schemas.

There are few proposals on schema testing in the context of XML. Li and Miller [13] propose mutation operators for XML schemas that make simple alterations in the schema, such as change of a value or an attribute; they do not show the application of their operators in a testing process for XML schemas. Franzotte and Vergilio [10] introduce a new set of mutation operators for XML schemas and present a tool to support their mutation testing process. Mutation operators produce a simple change in the schema under test. These operators are divided into elementary (to alter attributes values and elements) and structural (to modify the tree structure) mutations; their test data are generated manually.

5. Conclusions and Future Work

A generic testing approach for data schemas is being proposed. The testing approach is fault-based; therefore, fault classes based on typical faults made during the development of schemas are identified and classified into fault classes. Fault classes identified in the data schema under test guide the generation of data instances and queries, which are used to reveal the potential faults in the schema. Data instances and queries are generated according to patterns defined for each fault class. Data instances represent faults in the data schema and queries are capable of revealing these faults.

The generic approach can be used to test: XML documents used to store data as a database; XML messages used to change information in web applications; query results of database in XML format; XML documents updated due to alterations in the specification of the data stored in these documents. In the context of database schemas, the testing approach may be applied to relational databases and to other types of database models, such as object-oriented or object-relational databases.

An example of the testing approach in the context of XML was presented with XML Schema. Another example presented is on the testing of a data schema of a relational database.

In our approach the schema can be tested even if the application is not available. In addition to that, the different generated data instances may also be used, in the context of XML, to test applications that manipulate a document in format XML, interaction among web components and web services; and in the context of database, to test database applications.

This testing approach is being implemented by the tool XTool and we also intend to conduct experiments on real schemas in the contexts presented in this paper.

References

[1] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[2] CHAN, M.; CHEUNG, S.. *Testing Database Applications with SQL Semantics*. In Proc. of the 2nd Intl. Symp. on Cooperative Database Systems for Advanced Applications, pp 364-375, March 1999.

[3] CHAN, W. K.; CHEUNG, S.C.; TSE, T. H.. *Fault-Based Testing of Database Application Programs with Conceptual Data Model*. In Proc. of the 5th Intl. Conference on Quality Software, pp 187-196, 2005.

[4] CHAYS, D.; DENG, Y. *Demonstration of AGENDA Tool Set for Testing Relational Database Applications*. In Proc. of the 25th Intl. Software Engineering Conference, 2003. IEEE Computer Society, pp 802 – 803, May 2003.

[5] CHAYS, David; DAN, Saikat; FRANKL, Phyllis G.; VOKOLOS, Filippos I.; WEYUKER, Elaine J.. *A Framework for Testing Database Applications*. In Proc. of the 2000 ACM SIGSOFT Intl. Symp. on Software Testing and Analysis, Vol. 25 Issue 5, August 2000.

[6] CHEN, P. P.. *The Entity-Relationship Model – Toward a Unified View of Data*. ACM Transactions on Database Systems, Vol. 1, N^o 1, pp 9-36, 1976.

[7] DENG, Yuetang; FRANKL, Phyllis; CHAYS, David. *Testing Database Transactions with AGENDA*. In Proc. of the 27th Intl. Conference on Software engineering. ACM Press, May 2005.

[8] EMER, M.C.F.P.; VERGILIO, S.R.; JINO, M.. *A Testing Approach for XML Schemas.* In Proc. of the 29th Annual Intl. Computer Software and Applications Conference, Vol. 2, pp 57 – 62, July 2005.

[9] EMER, M.C.F.P.; NAZAR, I. F.; VERGILIO, S.R.; JINO, M. *Evaluating a Fault-Based Testing Approach for XML Schemas*. In Proc. of the 8th IEEE Latin-American Test Workshop, March 2007.

[10] FRANZOTTE, L.; VERGILIO, S. R. *Applying Mutation Testing to XML Schemas*. In Proc. of the 18th Intl. Conference on Software Engineering and Knowledge Engineering, July 2006.

[11] KAPFHAMMER, Gregory M.; SOFFA, Mary Lou. *A Family of Test Adequacy Criteria for Database-driven Applications*. In Proc. of the 9th European Software Engineering Conference, held jointly with 11th ACM SIGSOFT Intl. Symp. on Foundations of Software Engineering, Vol. 28 Issue 5, September 2003.

[12] LEE, S. C.; OFFUTT, J.; *Generating test cases for XML-based web component interactions using mutation analysis*. In Proc. of the 12th Intl. Symp. on Software Reliability Engineering, 2001. ISSRE 2001. Proceedings, pp 200 –209, November 2001.

[13] LI, J. B.; MILLER, J. *Testing the Semantics of W3C XML Schema.* In Proc. of the 29th Annual Intl. Computer Software and Applications Conference, July 2005.

[14] OFFUTT, J.; XU, W. *Generating Test Cases for Web Services Using Data Perturbation*. In Proc. of the TAV-WEB/ACM SIGSOFT SEN, vol. 29, n. 5, September 2004.

[15] OMG. *Meta-Object Facility (MOF) Specification Version 1.4*. http://www.omg.org/docs/formal/02-04-03.pdf, April 2002. (accessed in August 2005).

[16] OMG. *Meta-Object Facility Core Specification Version 2.0*. http://www.omg.org/cgi-bin/doc?formal/2006- 01-01, January 2006. (accessed in September 2006).

[17] PRESSMAN, Roger S. *Software Engineering – A Practitioner's Approach*. 5th ed., McGraw-Hill, New York, 2000.

[18] ROBBERT, M. A.; MARYANSKI, F. J.. *Automated Test Plan Generator for Database Application Systems*. In Proc. of the ACM SIGSAMLL/PC Symp. on Small Systems, pp 100-106, 1991.

[19] RUIZ, Duncan D.. *Transformação de modelos conceituais em modelos de implementação de bancos de dados*. Brazilian Symp. Database, 2005. (in Portuguese).

[20] SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Database System Concepts*. 3rd ed., McGraw-Hill, 1998.

[21] SUÁREZ-CABAL, M. J.; TUYA, J.. *Using an SQL Coverage Measurement for Testing Database Applications*. In Proc. of the 12th Intl. Symp. on the Foundations of Engineering, November 2004.

[22] W3C. Extensible Markup Language (XML) 1.0 (Third Edition)–W3C recommendation, February 2004.

[23] W3C. XML Schema, Working Draft, July. 2004.

[24] XU, W.; OFFUTT, J.; Luo, J. *Testing Web Services by XML Pertubation.* In Proc. of the 16th IEEE Intl. Symp. on Software Reliability Engineering, November 2005.

[25] ZHANG, Jian; XU, Chen; CHEUNG, S.-C.. *Automatic Generation of Database Instances for White-box Testing*. In Proc. of the 25th Annual Intl. Computer Software and Applications Conference, pp 161 – 165, October 2001.

NLForSpec: Translating Natural Language Descriptions into Formal Test Case Specifications

Daniel Leitão, Dante Torres and Flávia Barros

Centro de Informática – Universidade Federal de Pernambuco
PO Box 7851 - 50.732-970 Recife(PE), Brazil
E-mail: {dal,dgt,fab}@cin.ufpe.br

Abstract

This paper describes the NLForSpec, a Natural Language (NL) processing tool to translate software test cases descriptions in NL into a formal representation in CSP specification language. NLForSpec is part of a larger project which aims to automate the software test process for mobile phone applications. Our tool can be used in the process of update or partially generate requirements documents from test cases (one of the project's main goals). NLForSpec follows the traditional NL interpretation approach, counting on a lexicon, a case grammar and a domain ontology. The prototype was tested with a corpus of 100 test cases descriptions, obtaining a performance rate of 91%. This is an original and innovative work.

1 Introduction

The software development process involves three major artifacts: software requirements, code and test cases. Requirements and test cases are often written in some natural language (NL), since this is the stakeholders natural way of communicating. However, NL descriptions may be ambiguous and inconsistent. As a consequence, the interpretation of software requirements and test cases will depend on readers experience. Misinterpretations may introduce mistakes into the code and in the testing phase. One way to minimize these drawbacks is to derive unambiguous formal specifications from NL (requirements and test cases) descriptions. However, this is not a straightforward task, and may require domain knowledge and supporting tolls based on formal methods.

Besides imprecision of NL descriptions, we cite yet other problems specifically concerning requirements documents. Well-organized companies usually maintain the three artifacts mentioned above. However, during the development process the initial requirements documents may not follow the continuos update of code and test cases, therefore becoming out-of-date. On the other side, some companies maintain only code and test cases defined from non-documented requirements. For companies in the situation (1), it would be of great help to provide for automatic update of requirements documents from more up-to-date test cases. In turn, for companies in situation (2), it would be desirable to (even partially) automatically generate requirements documents from test cases.

We propose here the NLForSpec, a NL Processing (NLP) tool to translate software test cases descriptions in English into a formal representation. This tool addresses the two above-cited problems: (1) by providing a (precise) formal description of test cases, (2) which can be later used as a basis to generate or update requirements documents [5].

NLForSpec is based on the traditional pipeline NL interpretation architecture, counting on three processing modules and four knowledga bases. The input sentence is parsed and then mapped into case grammar structures (based on thematic roles). Finally, these structures are mapped into representation in CSP (Communicating Sequential Processes) formal language [9]. The prototype was tested with a corpus of 100 test cases descriptions within the mobile phone messaging application, obtaining a performance rate of 91%. This is an original an innovative work in both NL Processing and Software Engineering areas.

This work is part of the Motorola CIn-BTC research project, a partnership between the Informatics Center (CIn-UFPE) and the Motorola Brazil Test Center. The overall goal of this project is to automate test case generation, selection and evaluation for mobile phone applications. The CSP representations generated by the NLForSpec are used as input by other tools in CIn-BTC project responsible for generating use case models and requirements descriptions.

In what follows, Sect. 2 presents some state-of-the-art in translation from NL descriptions into formal specifications. Sect. 3 describes the architecture and basic features of the NLForSpec. Sect. 4 presents experiments and results, followed by related work (Sect. 5) and conclusions (Sect. 6).

2 From NL to Formal Specifications

This section briefly describes techniques and systems for translating NL descriptions into formal specifications. In the available literature, we could only identify three systems with similar (although, not exactly the same) goal as our tool: NL-OOPS [14], [6], [10]. This section will comment on the main features of these systems, which will be later compared to the NLForSpec (Sect. 5).These systems were developed within the Symbolic NLP approach, which centers around three major modules: syntactic analyzer, semantic analyzer and discourse/pragmatic analyzer [2].

The syntactic analysis stage aims to determine the input sentence syntactic structure according to a formal grammar. It may also include a pre-processing lexical analysis stage. This module can also be based on a Part-of-Speech (POS) tagger [7], which is commonly used in NLP systems based on the Empirical approach. In contrast to traditional parsers, POS taggers aim to determine the grammatical category of each word in the input sentence, rather than the sentence syntactic structure. Regarding the above-cited systems, unfortunately, we did not find in the available literature clear information on how they perform the syntactic analysis phase.

The semantic analysis is responsible for creating a representation of the sentence meaning, and it can deploy different formalisms (e.g., First-Order Logics, Semantic Networks, among others). Here, Knowledge Bases (KBs) may be used to explicit represent the objects, concepts, and other entities in a particular domain, as well as the relationships between them. Two of the above-cited systems follow this approach, using Semantic Networks as KBs [14] [6].

The discourse/pragmatics stage covers discourse analysis and intention recognition. Among the reviewed systems, only [10] treats discourse, since it takes as input descriptions of communications protocols, handling this input not as isolated sentences, but rather as paragraphs.

The following section describes the NLForSpec tool in detail, discussing its architecture and main features.

3 NLForSpec: from English Descriptions to Test Cases Specifications

As said before, the NLForSpec is part of a larger project, aimed to update requirements documents, among other goals. Our tool is the first step in the translation process from test case descriptions into formal use models, following the Anti-Model-Based Testing Approach [5]. As said, the formal language used in the major project is CSP [9].

Our tool was developed within the NLP symbolic approach, which decomposes the mapping process into a sequence of well-defined tasks (Sect. 2). However, the NLForSpec does not count on discourse and pragmatics analyzers, since each sentence in a test case represents an isolated action to be taken. Therefore, there is no need to process discourse fragments, but only isolated sentences.

In what follows, Sect. 3.1 presents the tools overall architecture, and a brief description of the tools knowledge bases and processing modules. Finally the systems process is illustrated by an example.

3.1 Architecture

Figure 1 illustrates the NLForSpec architecture. The rectangles represent the processing modules (Sect. 3.3): POS-Tagger, Semantic Processor and CSP Test Cases Generator. The cylinders represent knowledge bases (Sect. 3.4): Lexicon, Case Frame Base, Ontology and CSP Events.

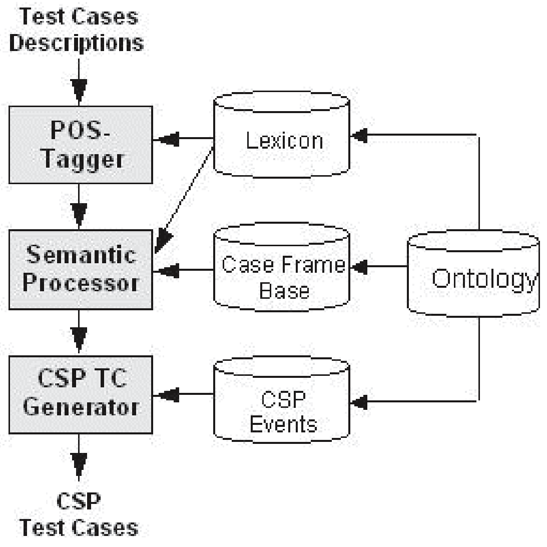


Figure 1. NLForSpec architecture

Input test cases are composed by a sequence of sentences of three different types, in the following order: Initial Conditions, Steps and Post Conditions. As said, each sentence in a test case represents an action to be taken. Here, the initial and post conditions are also interpreted in the same way as the actions. Each sentence is mapped onto one or more CSP events, which compose the CSP process that formally represents the correspondent test case.

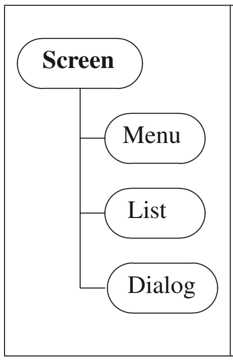
3.2 Knowledge Bases

This section presents the tool's KBs, which are all represented in XML format.

3.2.1 Ontology Base.

The domain entities are represented in the Ontology, which groups them into classes, according to their characteristics. Figure 2 shows a fragment of ontology and its representation in XML format in the domain of mobile phone applications.

The ontology represents only specialization relations between classes, in order to ease the addition of new domain entities (since it is just necessary to assign a class to the new entity).



```
<class>
    <name>Screen</name>
    <code>screen</code>
    <subclasses>
        <class>
            <name>Menu</name>
            <code>menu</code>
            <subclasses />
        </class>
        ...
    </subclasses>
</class>
```

Figure 2. Ontology excerpt: graphical and XML representations.

3.2.2 Lexicon.

The Lexicon stores the terms that may appear in the input domain. It is based on the phrasal lexicon approach [4], in which the terms are multi-word phrases. This KB contains three types of terms (Fig. 3): (1) Noun, representing a domain entity; (2) Verb, representing an action; and (3) Modifier, representing a modifier that may appear before or after a noun, changing its meaning.

```
<noun>
    <term>inbox folder</term>
    <plural/>
    <class>list</class>
    <model>INBOX_FOLDER</model>
</noun>
```

```
<modifier>
    <term>at least <int/> </term>
    <position>before</position>
    <precedence>last</precedence>
    <model>AT_LEAST.Int</model>
</modifier>
```

Figure 3. A fragment of the Lexicon.

Figure 3 presents two lexical terms in the sentence "Select at least 3 messages from inbox folder". As said, nouns represent domain entities. Then, each noun entry must contain a tag associating it to one Ontology class.

3.2.3 Case Frame Base.

This KB stores the system's grammar, which is based on the Case Grammar formalism [8]. It comprises a set of case frames with information about the input domain verbs and their thematic roles. These frames represent linguistic semantic knowledge (whereas the ontology represents domain semantic knowledge).

Each case frame corresponds to one verb meaning in the input domain, containing all verbs which share that same meaning and thematic roles.[1] For instance, the case frame SelectItem (Fig. 4) groups the verbs 'select' and 'choose'.

```
<frame>
    <name>SelectItem</name>
    <verblist>
        <verb>select</verb>
        <verb>choose</verb>
    </verblist>
    <roles>
        <role mandatory="True">agent</role>
        <role mandatory="True">theme</role>
        <role mandatory="False">from-loc</role>
    </roles>
</frame>
```

Figure 4. Case frame base entry.

The Case Frame base also contains a set of restrictions that specifies which ontology classes can be associated to each frame thematic roles. For example, the case frame SelectItem (Fig. 5) is composed by the *theme* role (associated to MenuItem ontology class), and by the *from-loc* role, which is associated to the Menu ontology class.

```
<restriction name="DTSEL_MENUITEM_MENU">
    <class role="theme">menu_item</class>
    <class role="from-loc">menu</class>
</restriction>
```

Figure 5. Frame restriction example.

[1] Our approach differs from the FrameNet Project [3], in which a case frame contains a set of verbs with the same thematic roles, not necessarily with the same meaning (e.g., 'rent' and 'buy' verbs in the Commerce_buy frame).

3.2.4 CSP Events Base.

This KB defines the CSP events (CSP channels and datatypes) used to compose the output CSP representation (Fig. 6). Each channel in this base corresponds to one frame in the Case Frame Base. The datatypes are defined based on the ontology classes, existing one datatype for each class.

```
<channel>
    <name>select</name>
    <casegrammar>SelectItem</casegrammar>
    <datatype>DTSelect</datatype>
</channel>
```

```
<datatype class=”list”>
    <label>LIST</label>
    <name>List</name>
</datatype>
```

Figure 6. CSP Base Example.

3.3 Processing Modules

The NLForSpec three processing modules are linked together in a pipeline. These modules are detailed in the following sections.

3.3.1 POS-Tagger.

The NLForSpec uses the Stanford POS-tagger [15] to replace the syntactic parsers used in traditional NLP systems. A comparative study among three Web available POS-taggers was performed: Montylingua [11], Stanford POS-tagger and OpenNLP POS-tagger [1]. These taggers were evaluated with 450 test case steps and the Stanford POS-tagger reached the best result.

As said, tagging aims to determine the grammatical category of each word in the input sentence, instead of its syntactic structure. However, because of lexical ambiguity, the correct determination of each category depends on the category of neighboring words. Therefore, the analysis of each word cannot occur in isolation. The POS-tagger receives a sentence in English, breaks it into tokens, and tags each token with its corresponding grammatical class.

3.3.2 Semantic Processor.

This module maps the tagged sentence into one (or more) case grammar frame in the Case Frame base. Each verb phrase in the sentence is mapped into one case frame. This module analyses the tagged sentence, and applies a set of rules to identify the most suitable case frame to each verb phrase. It then relates the nouns associated to the verb phrase to the corresponding frame's thematic roles. Finally, the Semantic Processor verifies the restrictions that determine whether the nouns related to thematic roles are allowed for the case frame under analysis.

3.3.3 CSP Test Cases Generator.

This module is responsible for mapping each case frame delivered by the previous module into the corresponding CSP event. As said, each event is composed by a channel and its associated datatypes (Fig. 6). This way, the case frame verb is mapped into the event's CSP channel, and the thematic roles are mapped into the corresponding CSP datatypes (that will constitute the channel messages).

3.4 System Execution Flow

Here we present an input-output flow example of all NLForSpec modules (Fig.7).

Consider the input sentence “Select at least 3 messages from Inbox”. The POS-Tagger receives this sentence as input, and provides a string with its parts-of-speech. The tag *VB* stands for Verb, *DT* for determiner, *NN* for Noun, *IN* for preposition, *JJS* for Adjective, and *CD* for Cardinal.

The Semantic Processor maps the tagged sentence into the corresponding case frame. Initially, this module selects the case frame that contains the verb identified by the POS-Tagger (i.e., ‘Select’), and retrieves the ontology classes associated to each noun in the sentence. In this example, the ontology class associated to the noun “message” is *SendableItem*[2], and the “inbox” class is *list*. After that, this module maps the sentence nouns into their respective thematic roles.

1. Input sentence: *Select at least 3 messages from Inbox*
2. Parts-of-speech: *Select/VB at/IN least/JJS 3/CD messages/NN*
3. Case Frame: *Verb: select; Theme: Message; From-loc: Inbox*
4. CSP Event: *select.DTSEL_SENDABLEITEM_LIST. (MESSAGE,{AT_LEAST.3}).(INBOX_FOLDER,{})*

Figure 7. NLForSpec Example Flow.

[2]i.e., an item that can be sent.

The Semantic Processor also verifies whether there is a restriction associated to the case frame 'Select' containing: (1) the thematic role *theme* associated to the ontology class *SendableItem*, and (2) the role *from-loc* associated to the class *list*. Next, the Semantic Processor identifies the nouns modifiers. In our example, the modifier "at least 3" is associated to the noun "message".

Finally, the CSP Test Case Generator receives the case frame instantiated by the previous module and the modifiers associated to the frame's nouns. This module maps the verb to the corresponding CSP channel, and the nouns and modifiers to the datatypes, delivering the CSP representation of the input sentence.

4 Case Study

This section presents the implemented prototype and experiments results. We worked within the domain of mobile phone applications testing. The experiments were performed over a corpus of 100 test cases descriptions from the messaging domain (Sect. 4.2). The obtained results were very encouraging, achieving a performance rate of 91%.

4.1 The Prototype

The tool prototype was implemented in Java, for portability and modularity. The implementation followed the persistent data collections pattern [13]. This way, the Knowledge Bases, represented as XML files, can be easily migrated to another storage form, such as a database.

In order to populate the Knowledge Bases, we randomly selected 450 test cases sentences from the mobile phone messaging application, which were manually analyzed. The Lexicon was filled-in with the nouns, verbs and modifiers identified in these sentences. After that, the Case Frame base was created considering the verbs in the lexicon, and the restrictions for each case frame were defined. It was also necessary to classify the identified nouns according to the Ontology. After that, the CSP Events Base was populated with the channels corresponding to the existing verbs, and the datatypes corresponding to nouns and modifiers.

4.2 Experiments and Results

In order to validate the prototype, we selected another feature from mobile phone application, one not used to populate the KBs. This feature contained 100 test cases.

The experiments results achieved a performance rate of 91%. Regarding the sentences that were not correctly translated into CSP specifications, the following situations were identified in test case pre-conditions:

1. Sentences without a verb, e.g., the sentence "At least 2 messages in Inbox" in test case pre-conditions, instead of "There are at least 2 messages in Inbox".

2. Ambiguous sentences (which can be interpreted in more than one way), e.g., the sentence "There is one read, unread, and protected message in Message Inbox". Which messages are in the Inbox: one of each type exclusively or inclusively?

5 Related Work

This section presents the three systems mentioned in Sect. 2, which aim to translate NL descriptions into specification models: [14], [6], and [10].

The NL-OOPS (Natural Language Object Oriented Production System) is a CASE tool that generates OO models from requirements documents in natural language. Its nucleus is the NLP system LOLITA [12], which receives requirements description and generates a semantic network (SemNET) that represents these requirements. A specific algorithm obtains from this network the classes, attributes, associations and operations that will appear on the generated OO model.

Two main drawbacks are identified in the NL-OOPS system: (1) it requires a manual pre-processing of the requirements documents in order to identify ambiguities, inconsistencies and omissions; and (2) for large documents, the corresponding semantic networks are very complex; consequently, the OO model generation task becomes difficult and slow, resulting in inadequate models.

Cyre et al. [6] present a system that generates VHDL models from NL descriptions. This system is composed by four processing modules. The syntactic analyzer parses input sentences based on a grammar of 120 rules. The second module performs the semantic analysis of each sentence and generates its respective conceptual graph (a semantic network). The third module integrates the generated graphs, and the final module generates the VHDL models.

The authors state that all generated models were correct. However, the experiments were based on descriptions containing at most 10 sentences. In our opinion, a description with 10 sentences does not seem to be complex enough to allow a more accurate evaluation of the system.

Finally, we highlight the [10] system, which translates natural language specifications of communication protocols into algebraic specifications. The natural language specifications define action sequences performed by the protocol machine (program). These sentences are translated into an algebraic specification language, called ASL. The published results of this system are good. However, the focus of this work is on communication protocols, which is different from ours, that is focused on test cases descriptions.

6 Conclusion and Future Work

We presented here the NLForSpec, an innovative NLP tool to translate test cases descriptions into formal specifications. This tool integrates a larger system that has as one of its goals the update of requirement documents from more up-to-date test cases. NLForSpec was developed based on the traditional symbolic NLP architecture, consisting of three processing modules (POS-Tagger, Semantic Processor and CSP Test Cases Generator) and four knowledge bases (Lexicon, Case Grammar, Ontology and CSP Events).

NLForSpec achieved a performance rate of 91% in experiments within the domain of Motorola's mobile phone testing. This result indicates that the proposed tool can be successfully integrated into a real industry software development and testing environment. Besides that, the flexibility of our KBs design allows the use of NLForSpec in different domains. For that, it is only necessary to change the KBs content. NLForSpec can also be used to translate NL descriptions into specifications in others formal languages, different than CSP. In this case, the CSP Events Base has to be replaced by other KB that contains information about the new output specification.

As future work, the NLForSpec will be adapted to translate Use Cases (extracted from requirement documents) into CSP specifications. This adaptation represents the first step towards the automatic test cases generation from requirement documents (another goal of our major project). Besides that, we will develop an intelligent user interface to facilitate the update of the NLForSpec KBs. This interface will favor the integration of our tool into the mobile phone testing activities, also allowing experiments with new mobile phone features.

References

[1] The opennlp homepage, 2006. Available at http://opennlp.sourceforge.net/. Accessed on March 07, 2007.

[2] J. Allen. *Natural Language Understanding*. The Benjamin Cummings Publishing Company, Inc, New York, NY, 1995.

[3] C. Baker, C. Fillmore, and J. Lowe. The berkeley framenet project. In *Proceedings of COLING/ACL*, pages 86–90, Montreal, Canada, 1998.

[4] J. Becker. The phrasal lexicon. In *Proceedings of the Conference on Theoretical Issues in Natural Language Processing*, pages 70–77, Cambridge, MA, 1975.

[5] A. Bertolino, A. Polini, P. Inverardi, and H. Muccini. Towards anti-model-based testing. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 124–125, Florence, Italy, 2004.

[6] W. R. Cyre, J. R. Armstrong, M. Manek-Honcharik, and M. Honcharik. Generating vhdl models from natural language descriptions. In *Proceedings of the Euro-VHDL*, pages 474–479, Grenoble, France, 1994.

[7] R. Dale, H. L. Somers, and H. Moisl. *Handbook of Natural Language Processing*. Marcel Dekker, Inc, 2000.

[8] C. J. Fillmore. Frame semantics and the nature of language. In *Annals of the New York Academy of Sciences: Conference on the Origin and Development of Language and Speech*, pages 20–32, 1976.

[9] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, 1978.

[10] Y. Ishihara, H. Seki, and T. A. Kasami. Translation method from natural language specifications into formal specifications using contextual dependencies. In *Proceedings of IEEE International Symposium on Requirements Engineering*, pages 232–239, 1992.

[11] H. Liu. Montylingua: An end-to-end natural language processor with common sense, 2004. Available at: web.media.mit.edu/ hugo/montylingua. Accessed on March 07, 2007.

[12] D. Long and R. Garigliano. *Reasoning by Analogy and Causality: Model and Applications*. Ellis Horwood, Chichester, UK, 1994.

[13] T. Massoni, V. Alves, S. Soares, and P. Borba. Pdc: The persistent data collections pattern. In *First Latin American Conference on Pattern Languages of Programming*, pages 232–239, Rio de Janeiro, Brazil, 2001.

[14] L. Mich. Nl-oops: From natural language to object oriented requirements using the natural language processing system lolita. *Journal of Natural Language engineering*, 2(2):161–187, 1996.

[15] K. Toutanova, D. Klein, C. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of HLT-NAACL*, pages 252–259., 2003.

Enhanced Random Testing for Programs with High Dimensional Input Domains

F.-C. Kuo[1], K.-Y. Sim[2*], Chang-ai Sun[3], S.-F. Tang[4] and Zhi Quan Zhou[5]

[1,4]Faculty of Information and Communication Technologies, Swinburne University of Technology, Australia

[2]School of Engineering, Swinburne University of Technology, Sarawak Campus, Malaysia

[3]Department of Mathematics and Computing Science, University of Groningen, The Netherlands

[5]School of Computer Science and Software Engineering, University of Wollongong, Australia

[1,4]{dkuo,satang}@ict.swin.edu.au, [2]ksim@swinburne.edu.my, [3]csun@cs.rug.nl, [5]zhiquan@uow.edu.au

Abstract

Random Testing (RT) is a fundamental technique of software testing. Adaptive Random Testing (ART) has recently been developed as an enhancement of RT that has better fault detection effectiveness. Several methods (algorithms) have been developed to implement ART. In most ART algorithms, however, the above enhancement diminishes when the dimensionality of the input domain increases. In this paper, we investigate the nature of failure regions in high dimensional input domains and propose enhanced random testing algorithms that improve the fault detection effectiveness of RT in high dimensional input domains.

1. Introduction

Effective test case selection strategies are essential to increase the chance of detecting failures and reduce the cost of software testing process. *Random Testing* (RT) is a simple test case selection strategy that treats the *Software Under Test* (SUT) as a black box [11]. In RT, test cases are selected randomly from the *input domain* (that is, the set of all possible inputs) of the SUT. Despite the criticism that RT may be ineffective as it does not make use of the information about the SUT [15] or previously executed test cases, RT has been a popular and successful testing method in many applications [8][9][10][12][13][14][16] as it is simple in concept and easy to implement.

Previous studies have shown that failure-causing inputs tend to be clustered in certain ways [1][2][9]. Regions formed by the failure-causing inputs are referred to as *failure regions* [1]. *Chan et al.* [3] classified these failure regions into three typical patterns: point, strip and block failure patterns.

Chen et al. [4] made use of the general information about the typical patterns of failure regions to improve the fault-detection effectiveness of RT. They found that when the failure-causing inputs cluster in a block or a reasonably thick strip area, the chance of detecting the first failure can be magnified by spreading the test cases widely and evenly across the input domain. This test case selection approach is known as *Adaptive Random Testing* (ART). ART can be implemented using the Fixed-Size-Candidate-Set (*FSCS*) algorithm [4].

In a study of the fundamental factors that may affect the fault-detection effectiveness of FSCS-ART [7], it was found that the performance of FSCS-ART deteriorates when the dimensionality of the input domain (that is, the number of input parameters) increases. Since real-life applications may have many input parameters, it is essential to create new algorithms that can improve the fault-detection effectiveness of RT in high dimensional input domains.

In this paper, we propose two enhanced RT algorithms for high dimensional input domains. These algorithms are designed based on our analysis on the locations of FSCS-ART test cases and the failure regions in high dimensional input domains. In addition, the fault-detection effectiveness of the proposed enhanced RT algorithms is evaluated through simulation experiments.

The rest of this paper is organized as follows: the next section presents the notations and evaluates the fault-detection effectiveness of FSCS-ART in high dimensional input domains. Section 3 analyzes the locations of FSCS-ART test cases and the failure regions

[*] Contact Author

in high dimensional input domains. Then Section 4 proposes two enhanced RT algorithms and evaluates their fault-detection effectiveness against RT through simulation experiments. Section 5 concludes the paper.

2. Background

In this section, we present the preliminaries of RT and ART performance evaluation and the problems associated with FSCS-ART algorithm in high dimensional input domains.

2.1 Preliminaries

Assume that a SUT has a set of n input parameters $\{x_1, x_2, \ldots, x_n\}$, where x_i (i=1, 2, …, n) has a bounded range $0 < x_i \le d_i$ (the situation where $u < x_i \le v$ and $u \neq 0$ can be mapped to $0 < x_i \le v - u$ as this mapping does not change the shape of the input domain). For an n-dimensional bounded input domain **D**, the size of the input domain is defined as $|\mathbf{D}| = \prod_{i=1}^{n} d_i$. With a failure region of size m, the failure rate Θ of the SUT is then defined as $\Theta = m/|\mathbf{D}|$.

F-measure [6] has been used to evaluate the performance of ART (in this paper, the term "fault-detection effectiveness" and "performance" are used interchangeably). F-measure is defined as the number of test cases required to detect the first failure. Let F_{ART} and F_{RT} denote the F-measure of ART and RT, respectively. Since ART is an enhanced version of RT, the *ART F-Ratio* (= F_{ART} / F_{RT}) was introduced to serve as the comparison metric to show how much improvement ART has over RT. Obviously, the smaller the ART F-Ratio is, the better the performance of ART will be.

For RT with uniform usage profile and replacement [6], the theoretical F_{RT} mean is $1/\Theta$. The F_{ART} can be obtained via empirical study. To obtain a statistically significant F_{ART} mean, all simulations were repeated until the F_{ART} mean has an accuracy range of 5% and a confidence level of 95%. For further details, please refer to [4].

ART is known to have the best performance in the block failure pattern [7]. In this paper, we will confine both empirical and analytical studies to a single block failure region with equal size for each of its dimensions.

2.2 Performance of FSCS-ART in high dimensional input domains

Figure 1 shows the performance of FCSC-ART when it is applied in 1(1D), 3(3D), 6(6D) and 9(9D) dimensional input domains for Θ ranging from 0.75 to 0.0005. For ease of presentation, Θ is plotted on a $\log_{0.5}$ scale.

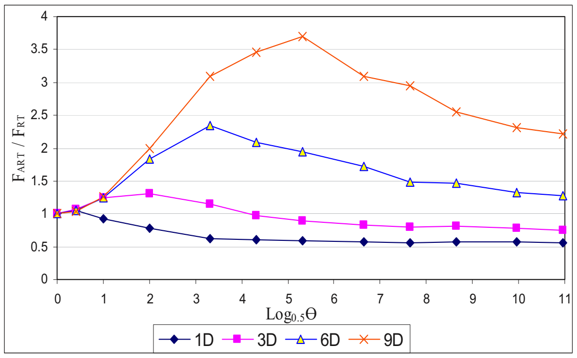


Figure 1: The ART F-ratios of FSCS-ART for input domains of different dimensionalities.

The experiment results in Figure 1 show that the existing FSCS-ART does not perform well when the failure rate is large or the number of dimensions of SUT is large. This observation was explained as a consequence of the "edge bias" of FSCS-ART in [7]. As the number of dimensions of the input domain increases, the performance of FSCS-ART deteriorates more significantly. Therefore, it is necessary to further investigate the "edge bias" problem and its impact on the performance of FSCS-ART.

3. Why the Performance of FSCS-ART Deteriorates in High Dimensional Input Domains?

To investigate the above problem, we examine the test case distribution of FSCS-ART and analyze the location of the failure region in the input domains.

3.1 Test case distribution of FSCS-ART

Based on the definitions in Section 2.1, we additionally define a *centre region* and an *edge region* within an n-dimensional input domain as follow.

The centre region, $\mathbf{D_c}$, is a sub-region of input domain **D** which shares the same centre with the input domain (denoted by O). The size of the centre region $\mathbf{D_c}$ is 50% of the size of input domain **D** (that is, $|\mathbf{D_c}| = 0.5 \times |\mathbf{D}|$). The width for each dimension of the centre region, cd_i, is given by $cd_i = \sqrt[n]{0.5}\, d_i$. The edge region, $\mathbf{D_e}$, is defined as the non-centre region in the input domain. The size of the edge region is equal to the centre region (that is, $|\mathbf{D_e}| = |\mathbf{D_c}| = 0.5 \times |\mathbf{D}|$). The "width" for each dimension of the edge region, ed_i, is defined as $ed_i = \left(1 - \sqrt[n]{0.5}\right) d_i / 2$.

Having introduced the concepts of the centre and edge regions, we would like to compare the numbers of test cases distributed over these two regions. Let *edgeCount* be the number of test cases located in the edge region, and *centreCount* be the number of test cases located in the centre region. Let $R_{E,C}$ be the ratio of *edgeCount* to *centreCount*. In our experiment, whenever FSCS-ART generates a test case, we will check its location and update the corresponding *centreCount* or an *edgeCount* accordingly. As the two regions are equal in size, an $R_{E,C}$ greater than 1 implies more test cases are being selected from the edge region. We conducted experiments to observe the values of $R_{E,C}$ produced by FSCS-ART for input domains with different dimensionalities when the number of test cases changed from 1 to 10,000. The results are shown in Figure 2.

From Figure 2, we can observe that, firstly, $R_{E,C}$ is almost always greater than 1, which means that FSCS-ART selects more test cases from the edge region than from the centre region. Secondly, $R_{E,C}$ becomes larger as the number of dimensions of the input domain increases. Intuitively, this is because the higher the dimensionality, the more corners/edges the input domain has, which will be filled up first by the test cases generated by the FSCS-ART algorithm. Thirdly, $R_{E,C}$ increases to a peak and then fluctuates prior to decreasing gradually towards limit 1.

Before we further analyze the impact of such "edge biased" test case distribution on the performance of FSCS-ART, we would like to examine the location of the failure region in the input domain.

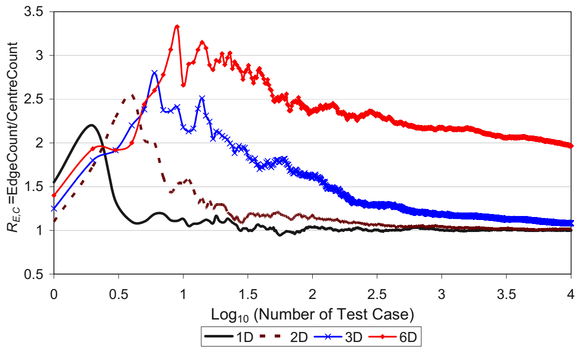


Figure 2: The ratio $R_{E,C}$ produced by FSCS-ART when the number of test cases ranged from 1 to 10,000

3.2 The location of the failure region in the input domain

When the number of dimensions of the input domain increases, the width of the centre region will increase, and in turn the width of the edge region will decrease. This observation brings up an important question: for a specific failure rate, as the edge width becomes narrower in high dimensional input domains, would the failure-causing inputs (points in a randomly located block failure region) have an equal chance of falling into the centre region and the edge region?

Let the *n*-dimensional input domain be homogeneous and have a unit size, that is, |D|=1 and d_i=1 (i = 1, 2, …, n). The size of a failure region F is given by |F|= Θ×|D|. Assuming that the failure region has the same orientation as the input domain and let the failure region be a block with equal width for each of its dimensions. Each dimension of the failure region is denoted by f_i (i = 1, 2, …, n), with the width $|f_i| = \sqrt[n]{\theta|D|}$. Figure 4 shows how $|f_i|$ varies against the number of dimensions for the failure rates Θ=0.005 and Θ=0.0005.

Figure 3 shows that, for the same failure rate Θ, when the number of dimensions of the input domain increases, $|f_i|$ will also increase but the width of the edge region, ed_i, will decrease. When Θ=0.0005, $|f_i|$ will become greater than ed_i for input domains of 4 dimensions and above; for a higher failure rate, say Θ=0.005, this will happen more quickly. When $|f_i| > ed_i$, it means that even in the situation where the failure region is attached to a border of the input domain, part of the failure region will still fall into the centre region. Therefore, the probability distribution for the location of failure region within the input domain warrants further analysis.

Referring to Figure 4, consider how f_i of different sizes can be located fully or partially within the centre region and edge region of dimension i. Let $P_{centre,i}$ denote the probability that some or all the elements of f_i fall into the centre region of dimension i. Similarly, the probability that some or all elements of f_i fall into the edge region of dimension i is denoted by $P_{edge,i}$. Equations (1) and (2) define $P_{centre,i}$ and $P_{edge,i}$ respectively.

$$P_{\text{centre},\,i} = \begin{cases} \dfrac{b-a}{d_i - |f_i|} & 0 < |f_i| \le a \\[2ex] 1 - \dfrac{\dfrac{a^2}{|f_i|}}{d_i - |f_i|} & a < |f_i| \le b \\[2ex] \dfrac{(b-a)\left(\dfrac{d_i}{|f_i|}+1\right)}{d_i - |f_i|} & b < |f_i| \le d_i \end{cases} \tag{1}$$

$$P_{\text{edge},i} = \begin{cases} 1 - \dfrac{(b-a)}{d_i - |f_i|} & 0 < |f_i| \le a \\[2ex] \dfrac{\dfrac{a^2}{|f_i|}}{d_i - |f_i|} & a < |f_i| \le b \\[2ex] 1 - \dfrac{(b-a)\left(\dfrac{d_i}{|f_i|}+1\right)}{d_i - |f_i|} & b < |f_i| \le d_i \end{cases} \tag{2}$$

For an n-dimensional input domain, the probability that some or all elements of the failure region are located in the centre region is denoted as P_{centre}. On the other hand, the probability that some or all elements of the failure region are located in the edge region is denoted as P_{edge}. Assuming that the input domain is homogeneous (that is, $d_1=d_2=...d_n$), P_{centre} and P_{edge} can be simplified into Equations (3) and (4), respectively.

$$P_{centre} = \prod_{i=1}^{n} P_{centre,i} \tag{3}$$

$$P_{edge} = \prod_{i=1}^{n} P_{edge,i} + \sum_{i=1}^{n-1} {}^{n}C_i \left(P_{edge,i}\right)^{i} \left(P_{centre,i}\right)^{n-i} \tag{4}$$

Figure 5 plots the ratio P_{centre}/P_{edge} against the failure rates, ϴ, from 0.75 to 0.0005. When $P_{centre}/P_{edge} > 1$, it indicates that elements of failure region have a higher probability to occupy the centre region than the edge region. As mentioned earlier, when the dimensionality of the input domain increases, $R_{E,C}$ becomes higher, which indicates that FSCS-ART will select more and more test cases from the edge region than from the centre region. However, at the same time, the ratio P_{centre}/P_{edge} also becomes higher, which means that the failure region has higher probability to occupy the centre region than the edge region when the dimensionality of the input domain increases. As a result, FSCS-ART becomes less effective in high dimensional input domain.

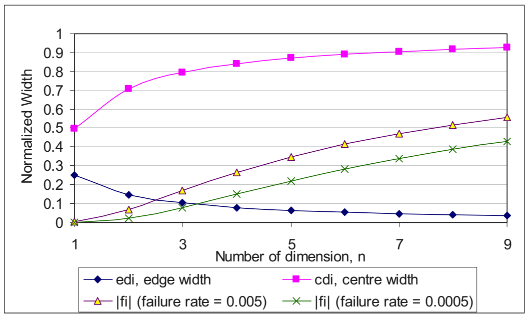


Figure 3: Widths of the edge region, centre region and failure region in different dimensionalities

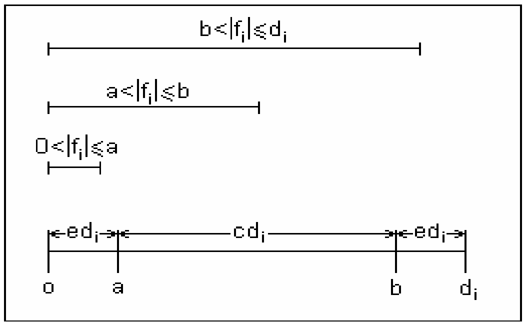


Figure 4: f_i of different sizes compared to the centre and edge widths.

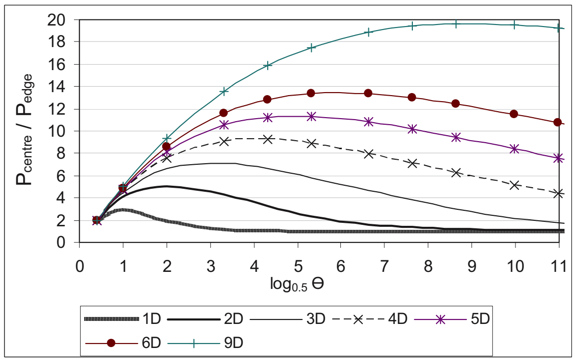


Figure 5: P_{centre} / P_{edge} for input domain of different dimensionalities

4. Enhanced RT Algorithms for High Dimensional Input Domains

The above analysis gives us an inspiration that if we select more test cases from the region where failure-causing inputs are more likely to fall into, then we will achieve better fault-detection effectiveness. In this section, we propose two methods (algorithms) to improve the fault detection effectiveness of RT in high dimensional input domains and report their performance through simulation experiments.

4.1 Inverted FSCS-ART

Since FSCS-ART selects more test cases from the edge region than from the centre region in high dimensional input domains, a simple approach to improve the fault detection effectiveness is to invert the edge/centre distribution of FSCS-ART test cases. This can be done by mapping the FSCS-ART test cases from the edge to the centre region and vice versa before executing them. We name this method *Inverted FSCS-ART*. This algorithm is outlined in Figure 6. Note that the core of the FSCS-ART test case selection algorithm remains unchanged. Equation (5) is one of the linear functions that can map the FSCS-ART test cases from the edge to the centre region and vice versa. Note that x_i is one of the input parameters (that is, one of the dimensions) in the input domain and $0 < x_i \le d_i$.

$$f(x_i) = \begin{cases} x_i + d_i/2 & 0 < x_i \le d_i/2 \\ x_i - d_i/2 & d_i/2 < x_i \le d_i \end{cases} \tag{5}$$

To evaluate the performance of Inverted FSCS-ART, simulations were conducted for failure rates, ϴ, ranging from 0.75 to 0.0005 for n-dimensional input domains where n = 1, 3, 6 and 9. The simulation results in Figure 7 show that Inverted FSCS-ART outperforms RT for all failure rates and dimensionalities of input domains under the study. The performance of Inverted FSCS-ART in 1D input domain is very similar to that of FSCS-ART.

However, for 3D input domain, it can be observed that the F_{IART}/F_{RT} ratio falls to a minimum before settling at 0.7. Similar trend can be observed for 6D and 9D input domains where the F_{IART}/F_{RT} minimums occur at smaller failure rates, Θ (that is, higher values on $\log_{0.5}\Theta$ scale). This observation concurs with the higher P_{centre}/P_{edge} and $R_{E,C}$ ratios in higher dimensional input domains. As the failure region has increasing chances of occupying the centre region in high dimensional input domains, inverting the edge-biased FSCS-ART test case distribution will increase the chance of detecting the failures region.

```
1. Initialize E as an empty set.
2. Randomly choose a test case t. Add t to E.
3. Test the SUT using test case t.
4. If a failure is detected, testing is stopped and debugging
   may start. Otherwise go to step 5.
5. Randomly generate k candidates from the input domain to
   form a candidate set C, where k is a constant integer
   greater than 0. The value of k was set to 10 in our
   experiments.
6. Find $c \in C$ such that, among all the elements in C, c has
   the longest distance to its nearest neighbor in E.
7. Add c to E.
8. Map c to c' using Equation (5).
9. Test the SUT using test case c'.
10. If testing resources are not exhausted, go to step 4.
```

Figure 6: Inverted FSCS-ART algorithm

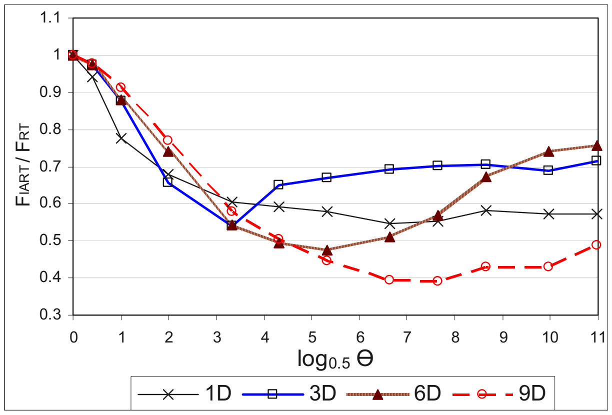


Figure 7: The ratio of the F-measure mean of Inverted FSCS-ART to the F-measure mean of RT.

4.2 Proportional Random Testing

The P_{centre}/P_{edge} ratio provides a useful guideline for the number of test cases that should be selected in the centre and in the edge region. Ideally, $1/(R_{E,C})$ should be in proportion to, and, as close as possible to the P_{centre}/P_{edge} ratio. Unfortunately, failure rate is unknown during testing. Therefore, it is impossible to determine the P_{centre}/P_{edge} ratio. However, the failure rate can be projected dynamically based on the number of test cases that have been executed. By taking the theoretical F-measure mean of random testing, after j test cases have been executed, the failure rate can be projected as $\Theta_{projected}=1/(j+1)$, assuming that next test case will detect the first failure. The ratio P_{centre}/P_{edge} can then be estimated based on $\Theta_{projected}$.

We propose the following algorithm in Figure 8 to select test cases randomly in the centre and in the edge region in proportion to the ratio P_{centre}/P_{edge}. We name this algorithm as "*Proportional Random Testing*" (PRT).

```
1. Initialize j to 0, edgeCount to 1, centreCount to 1, and
   selectTestCaseInCentre to true, where j is the number of
   test cases executed.
2. If selectTestCaseInCentre is true, randomly select and
   execute a test case in the centre region. Otherwise,
   randomly select and execute a test case in the edge
   region.
3. Increment j by 1. If selectTestCaseInCentre is true,
   increment centreCount by 1. Otherwise, increment
   edgeCount by 1. Update $R_{E,C}$.
4. If a failure is detected, testing is stopped and debugging
   may start. Otherwise go to step 5.
5. Update the projected failure rate as $\Theta_{projected}=1/(j+1)$.
6. Calculate the ratio $P_{centre}/P_{edge}$ based on $\Theta_{projected}$.
7. If $1/(R_{E,C}) < P_{centre}/P_{edge}$, set
   selectTestCaseInCentre=true. Otherwise, set
   selectTestCaseInCentre=false.
8. If testing resources are not exhausted, go to step 2.
```

Figure 8: Proportional Random Testing algorithm

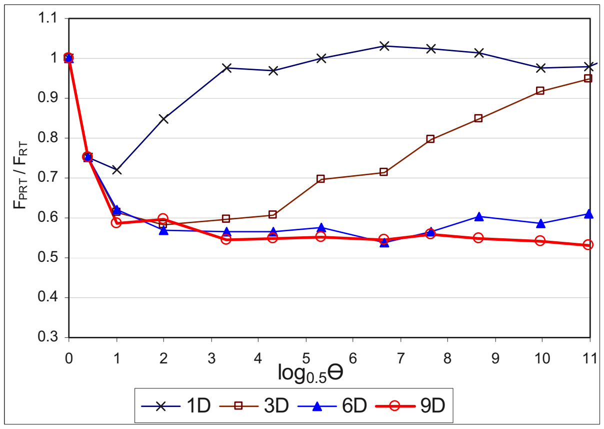


Figure 9: The ratio of the F-measure mean of Proportional Random Testing (PRT) to the F-measure mean of RT.

To evaluate the performance of the Proportional Random Testing, simulations were conducted for failure rates, Θ, ranging from 0.75 to 0.0005 for n-dimensional input domains where n = 1, 3, 6 and 9. The simulation results in Figure 9 show that Proportional Random Testing outperforms RT significantly when the number of dimensions is high (that is, when the P_{centre}/P_{edge} ratio is sufficiently large). As the dimensionality of the input domain increases, it can be observed that F_{PRT}/F_{RT} approaches 1 at smaller failure rates, Θ (that is, higher values on $\log_{0.5}\Theta$ scale). However, this algorithm is only as good as RT when the number of dimensions is low (that is, when $P_{centre}/P_{edge} \approx 1$).

5. Conclusion

In this paper, we proposed two enhanced RT algorithms for high dimensional input domains based on our analysis on two fundamental reasons that cause the performance of FSCS-ART to deteriorate in high dimensional input domains.

Proportional Random Testing is superior to Inverted FSCS-ART in computational cost for test case generation because Inverted FSCS-ART inherits the high computational cost in test case selection from FSCS-ART [5]. However, Proportional Random Testing does not give significant improvement over RT when the number of dimensions is low. Therefore, Proportional Random Testing should be used to generate test cases only when the number of dimension is high (that is, more than 3). On the other hand, Inverted FSCS-ART does not suffer from this setback. It can be used as a generic algorithm to generate test cases for programs with an input domain of any dimensionality.

For future work, we intend to evaluate the performance of the proposed algorithms for other patterns of failure regions in high dimensional input domains.

6. Acknowledgements

This research is partially supported by an Australia Research Council Discovery grant (Project No. DP0557246) and a Small Grant of the University of Wollongong.

7. References

[1] P.E. Ammann and J.C. Knight. Data diversity: an approach to software fault tolerance, *IEEE Transactions on Computers*, 37(4): pp.418-425, 1988.

[2] P.G. Bishop. The variation of Software Survival Times for different operation input profiles, *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing*, IEEE Computer Society Press, pp.98-107, 1993

[3] F.T. Chan, T.Y. Chen, I.K. Mak, and Y.T. Yu. Proportional Sampling Strategy: Guidelines for Software Testing Practitioners, *Information and Software Technology*, 38, (12), pp.775-782, 1996.

[4] T.Y. Chen, H. Leung, and I.K. Mak. Adaptive Random Testing. *Proceedings of the 9th Asian Computing Science Conference, LNCS 3321*, Springer-Verlag, pp.320-329. 2004.

[5] T.Y. Chen, F.-C. Kuo, R.G. Merkel and S.P. Ng, Mirror Adaptive Random Testing, *Information and Software Technology*. 46(15):pp.1001-1010. 2004.

[6] T.Y. Chen, F.-C. Kuo, R. Merkel. On the Statistical Properties of the F-measure, *Proceedings of the Fourth International Conference on Quality Software*. pp. 146 – 153. 2004,

[7] T.Y. Chen, F.-C. Kuo, and Z.Q. Zhou. On the Relationships between the Distribution of Failure-causing Inputs and Effectiveness of Adaptive Random Testing, *Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering*, pp.306-311, 2005.

[8] R. Cobb and H. D. Mills. Engineering software under statistical quality control. *IEEE Software*, 7(6):45-54, 1990.

[9] G.B. Finelli. NASA Software Failure Characterization Experiments, *Reliability Engineering and System Safety*, IEEE Computer Society Press, 32(1-2):pp.155-169, 1993.

[10] J. E. Forrester and B. P. Miller. An empirical study of the robustness of Windows NT applications using random testing. *Proceedings of the 4th USENIX Windows Systems Symposium*, pp. 59−68, Seattle, 2000.

[11] R. Hamlet. Random Testing, *Encyclopedia of Software Engineering*, Wiley, New York, pp. 970-978. 1994.

[12] N. Nyman. In defense of monkey testing: Random testing can find bugs, even in well engineered software. http://www.softtest.org/sigs/material/nnyman2.htm, *Microsoft Corporation*.

[13] B. P. Miller, D. Koski, C. P. Lee, V. Maganty, R. Murthy, A. Natarajan, and J. Steidl. Fuzz revisited: A reexamination of the reliability of UNIX utilities and services. Technical Report CS-TR-1995-1268, University of Wisconsin, 1995.

[14] H.D. Mills, M. Dyer, and R.C. Linger, "Cleanroom Software Engineering", *IEEE Software*, 3, pp.19-24, 1986.

[15] G.J. Myers. *The Art of Software Testing*. John Wiley and Sons, Inc, ISBN0-471-04328-1. 1979.

[16] T. Yoshikawa, K. Shimura, and T. Ozawa. Random program generator for Java JIT compiler test system. *Proceedings of the 3rd International Conference on Quality Software*, pp.20−24. IEEE Computer Society Press, 2003.

On Test Case Distributions of Adaptive Random Testing[*]

Tsong Yueh Chen   Fei-Ching Kuo[†]   Huai Liu
Faculty of ICT, Swinburne University of Technology, Australia
E-mail: {tchen, dkuo, hliu}@ict.swin.edu.au

Abstract

Adaptive Random Resting (ART) has recently been proposed as an approach to enhancing the fault-detection effectiveness of Random Testing (RT). The basic principle of ART is to enforce randomly selected test cases as evenly spread over the input domain as possible. Many ART methods have been proposed to evenly spread test cases in different ways, but no comparison has been made among these methods in terms of their test case distributions. In this paper, we conduct a comprehensive investigation on test case distributions of various ART methods. Our work shows many interesting aspects related to ART's performance and its test case distribution. Furthermore, it points out a new research direction on enhancing ART.

1. Introduction

Random Testing (RT), a fundamental software testing technique, simply generates test cases in a random manner from the set of all possible inputs, namely the *input domain* [10, 14]. RT has been successfully applied in industry to detect software failures [15, 16, 17].

It has been observed that for most programs, the *failure-causing inputs* (program inputs that can reveal failures) are clustered together [1, 2, 9]. Chen *et al.* [8] studied how to improve the fault-detection effectiveness of RT under such a situation. They proposed a novel approach, namely *Adaptive Random Testing* (ART), where test cases are not only randomly selected from the input domain, but also enforced as evenly spread over the input domain as possible. Since then, many ART methods have been proposed, such as *Fixed-Sized-Candidate-Set ART* (FSCS-ART) [8], *Ristricted RT* (RRT) [4], *ART through dynamic partitioning* [5] and *Lattice-based ART* [12]. Different ART methods distribute their test cases in different ways. All previous studies on ART methods [4, 5, 8, 12] were focused on the performance improvement that ART has over pure RT. No work has been conducted to compare ART methods with respect to their test case distributions, not to say the study on the relationship between the test case distributions and the performance of these methods.

In this paper, we study four ART methods, FSCS-ART [8], RRT [4], and two versions of *ART through dynamic partitioning*, namely, "*by bisection*" (BPRT) and "*by random partitioning*" (RPRT) [5]. We measure their fault-detection effectiveness, and examine their test case distributions using various metrics.

2. The effectiveness of various ART methods

For ease of discussion, we introduce notations and concepts commonly used in this paper as follows.

- D denotes the input domain of N dimension.
- dD denotes d-dimension, where $d = 1, 2, \cdots, N$.
- E denotes the set of already executed test cases.
- $|D|$ and $|E|$ denote the size of D and E, respectively.
- θ denotes failure rate, ratio of the number of failure-causing inputs to the number of all possible inputs.

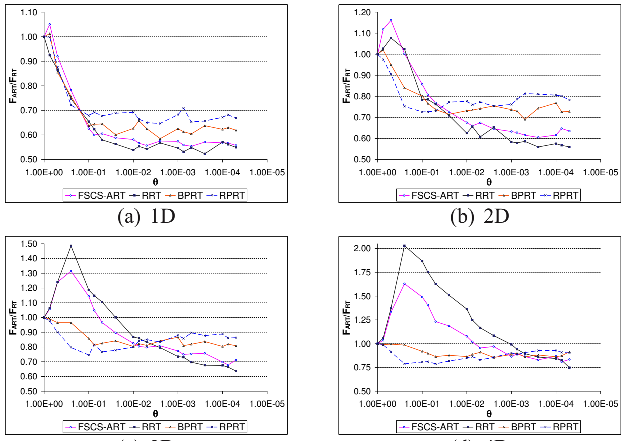
There are different notions of implementing ART, and different notions give rise to various ART methods. For the detailed algorithms of the ART methods, refer to [11]. The performance of ART methods is usually evaluated by F-measure, the expected number of test cases to detect the first failure. A testing method is considered more effective if it has a smaller F-measure.

As shown in [7], ART performs best when failure-causing inputs are well clustered into one single compact block (resuls are given in Experiment 1 of [7]). We followed the same experimental setting to study how various ART methods perform. It was expected that the data collected in this section could help us better understand the relationship between ART performance and even spreading of test cases. The experimental results are summarized in Figure 1, where x-axis denotes θ, and y-axis denotes *ART F-ratio*, which is defined as the ratio of F-measure of ART (denoted by $\mathrm{F}_{\mathrm{ART}}$) to that of RT (denoted by F_{RT}). F-ratio measures the improvement of ART over RT. From these results, the following observations can be made.

- Almost all the experimental data show that these ART methods have larger F-measures in higher dimensional spaces for the same θ.

[*]This research project is supported by an Australian Research Council Discovery Grant (DP0557246).

[†]Corresponding author



(a) 1D (b) 2D (c) 3D (d) 4D

Figure 1. The effectiveness of various ART methods in different dimensions

- FSCS-ART and RRT have very similar performance trends. When θ is large, both methods perform worse than RT and their F-ratios increase as θ decreases. When θ drops to a certain value v, their F-ratios start to decrease as θ decreases.
- BPRT and RPRT have opposite performance trends compared with FSCS-ART and RRT. For large θ, F-ratios of BPRT and RPRT are smaller than 1, and decrease as θ decreases. After θ drops to a specific value v, F-ratios of BPRT and RPRT stop decreasing.
- There is a specific failure rate v such that RRT has larger F_{ART} than FSCS-ART when $\theta > v$; and RRT has smaller F_{ART} than FSCS-ART when $\theta \leq v$.
- There is a specific failure rate v such that BPRT has larger F_{ART} than RPRT when $\theta > v$; and BPRT has smaller F_{ART} than RPRT when $\theta \leq v$.

Although all above methods aim at evenly spreading test cases, this study shows that their fault-detection effectiveness vary. The differences are mainly due to their ways of distributing test cases. In this paper, the distribution of test cases generated by these ART methods will be measured.

3. Measurement of test case distribution

For ease of discussion, the following notations are introduced. Suppose p$'$ and p$''$ are two elements of E. $dist$(p$'$, p$''$) denotes the distance between p$'$ and p$''$; and η(p, $E\backslash\{$p$\}$) denotes the nearest neighbour of p in E. Without loss of generality, the range of values for each dimension of D is set to [0, 1), or simply D = [0, 1)^N.

In this study, three metrics were used to measure the test case distributions (the distribution of E in D). The following outlines the definitions of these three metrics.

- **Discrepancy**
$$M_{Discrepancy} = \max_{i=1...m} \left| \frac{|E_i|}{|E|} - \frac{|D_i|}{|D|} \right| \quad (1)$$
where D_1, D_2, ..., D_m denote m randomly defined subsets of D, with their corresponding sets of test cases being denoted by E_1, E_2, ..., E_m, which are subsets of E. Note that m is set to 1000 in this paper.

  Intuitively, $M_{Discrepancy}$ indicates whether regions have an equal density of points. E is considered reasonably equidistributed if $M_{Discrepancy}$ is close to 0.
- **Dispersion**
$$M_{Dispersion} = \max_{i=1...|E|} dist(e_i, \eta(e_i, E\backslash\{e_i\})) \quad (2)$$
where $e_i \in E$.

  Intuitively, $M_{Dispersion}$ indicates whether any point in E is surrounded by a very large empty spherical region. A small $M_{Dispersion}$ indicates that E is reasonably equidistributed.
- **The ratio of the number of test cases in the edge of the input domain (E_{edge}) to the number of test cases in the central region of the input domain (E_{centre})**
$$M_{Edge:Centre} = \frac{|E_{edge}|}{|E_{centre}|} \quad (3)$$
where E_{edge} and E_{centre} denote two disjoint subsets of E locating in D_{edge} and D_{centre}, respectively; $E = E_{edge} \cup E_{centre}$. Note that $|D_{centre}| = \frac{|D|}{2}$ and $D_{edge} = D\backslash D_{centre}$. Therefore, $D_{centre} = \left[\frac{1}{2} - \sqrt[N]{\frac{|D|}{2^{N+1}}}, \frac{1}{2} + \sqrt[N]{\frac{|D|}{2^{N+1}}}\right)^N$.

  Clearly, in order for $M_{discrepancy}$ to be small, the $M_{Edge:Centre}$ should be close to 1; otherwise, different parts of D have different densities of points.

Discrepancy and dispersion are two commonly used metrics for measuring sample point equidistribution. More details of these two metrics can be found in [3].

The above three metrics were used to measure the test case distribution of a testing method from various perspectives. The space where a method generated points (test cases) was set to either 1D, 2D, 3D, or 4D. $|E|$ was set as from 100 to 10000. A sufficient amount of data were collected in order to get a reliable mean value of a metric within 95% confidence level and $\pm 5\%$ accuracy range.

It is interesting to find out how test cases of pure RT are distributed with respect to these metrics. Like all previous studies of ART, it was assumed that RT has a uniform distribution of test cases, which means that all test cases have an equal chance of being selected. Note that "uniform distribution" does not imply even spreading of test cases.

4. Analysis of test case distributions

The ranges of $M_{Edge:Centre}$ for all testing methods are summarized in Figure 2, with the following observations.

- When $N = 1$, $M_{Edge:Centre}$ for all ART methods under study is close to 1.
- When $N > 1$, FSCS-ART and RRT tend to generate more test cases in D_{edge} than in D_{centre} (or simply, FSCS-ART and RRT have *edge bias*). Moreover, the edge bias is stronger with RRT than with FSCS-ART.

RT	1D	2D	3D	4D
Max	1.0315	1.0341	1.0208	1.0202
Min	1.0000	0.9998	1.0005	0.9998
Max-Min	0.0315	0.0343	0.0203	0.0204
FSCS-ART	1D	2D	3D	4D
Max	1.0103	1.1442	1.6105	2.0733
Min	0.9997	1.0132	1.0863	1.2409
Max-Min	0.0106	0.1310	0.5241	0.8323
RRT	1D	2D	3D	4D
Max	1.0114	1.2103	2.1943	4.7346
Min	0.9995	1.0201	1.1358	1.3444
Max-Min	0.0119	0.1901	1.0585	3.3902
BPRT	1D	2D	3D	4D
Max	1.0085	1.0456	1.0482	1.0250
Min	0.9991	0.9975	0.9963	0.9979
Max-Min	0.0094	0.0480	0.0519	0.0271
RPRT	1D	2D	3D	4D
Max	1.0001	0.9995	0.9923	0.9780
Min	0.9812	0.9522	0.9236	0.9012
Max-Min	0.0190	0.0473	0.0688	0.0768

Figure 2. Range of $M_{Edge:Centre}$ for each testing method and dimension where $|E| \leq 10000$

- When $N > 1$, RPRT allocates more test cases in D_{centre} than in D_{edge} (or simply, RPRT has *centre bias*). But the centre bias of RPRT is much less significant than the edge bias of FSCS-ART or RRT.
- The edge bias (for FSCS-ART and RRT) and centre bias (for RPRT) increase as N increases.
- RT and BPRT have neither edge bias nor centre bias.

The ranges of $M_{Discrepancy}$ for all testing methods are summarized in Figure 3, with the following observations.

RT	1D	2D	3D	4D
Max	0.1056	0.1093	0.0925	0.0785
Min	0.0105	0.0106	0.0092	0.0077
Max-Min	0.0951	0.0987	0.0833	0.0709
FSCS-ART	1D	2D	3D	4D
Max	0.0437	0.0719	0.0876	0.0793
Min	0.0040	0.0062	0.0145	0.0188
Max-Min	0.0396	0.0657	0.0731	0.0605
RRT	1D	2D	3D	4D
Max	0.0401	0.0810	0.1279	0.1172
Min	0.0035	0.0074	0.0227	0.0302
Max-Min	0.0365	0.0736	0.1052	0.0871
BPRT	1D	2D	3D	4D
Max	0.0470	0.0674	0.0670	0.0609
Min	0.0018	0.0019	0.0016	0.0013
Max-Min	0.0452	0.0655	0.0654	0.0595
RPRT	1D	2D	3D	4D
Max	0.0610	0.0701	0.0696	0.0651
Min	0.0056	0.0052	0.0054	0.0054
Max-Min	0.0554	0.0649	0.0642	0.0597

Figure 3. Range of $M_{Discrepancy}$ for each testing method and dimension where $|E| \leq 10000$

- The impact of N on $M_{Discrepancy}$ of RT, FSCS-ART and RRT is different. As N increases, the $M_{Discrepancy}$ for RT decreases, but for FSCS-ART and RRT, it increases.
- $M_{Discrepancy}$ for RPRT and BPRT are independent of the dimensions under study.
- In general, BPRT has the smallest $M_{Discrepancy}$ for all N.
- When $N = 1$, FSCS-ART, RRT and BPRT have almost identical values for $M_{Discrepancy}$.

Clearly, measuring the density of points in two partitions (D_{Edge} and D_{Centre}) is only part of the measuring by $M_{Discrepancy}$ (which measures the density of points in 1000 randomly defined partitions of D). Hence, smaller $|M_{Edge:Centre} - 1|$ does not necessarily imply a smaller $M_{Discrepancy}$, but a smaller $M_{Discrepancy}$ does imply a smaller $|M_{Edge:Centre} - 1|$. This explains why the value of $M_{Edge:Centre}$ for RT is close to 1 for all N, but its $M_{Discrepancy}$ is not the smallest.

The ranges of $M_{Dispersion}$ for all testing methods are summarized in Figure 4, with the following observations.

RT	1D	2D	3D	4D
Max	0.0272	0.1488	0.2851	0.4138
Min	0.0005	0.0189	0.0714	0.1461
Max-Min	0.0267	0.1298	0.2137	0.2676
FSCS-ART	1D	2D	3D	4D
Max	0.0160	0.1281	0.2731	0.4139
Min	0.0002	0.0141	0.0601	0.1299
Max-Min	0.0158	0.1141	0.2129	0.2839
RRT	1D	2D	3D	4D
Max	0.0156	0.1253	0.2780	0.4350
Min	0.0002	0.0132	0.0585	0.1274
Max-Min	0.0154	0.1121	0.2195	0.3077
BPRT	1D	2D	3D	4D
Max	0.0182	0.1365	0.2797	0.4121
Min	0.0002	0.0150	0.0649	0.1388
Max-Min	0.0180	0.1215	0.2148	0.2733
RPRT	1D	2D	3D	4D
Max	0.0179	0.1335	0.2743	0.4020
Min	0.0002	0.0161	0.0671	0.1414
Max-Min	0.0177	0.1174	0.2071	0.2606

Figure 4. Range of $M_{Dispersion}$ for each testing method and dimension where $|E| \leq 10000$

- For $N = 1$, all ART methods have smaller $M_{Dispersion}$ values than RT.
- For $N > 1$, RRT normally has the smallest $M_{Dispersion}$ values, followed in ascending order by FSCS-ART, BPRT, RPRT and RT.

In Table 1, the testing methods are ranked according to their test case distribution metrics. For the same metric, the method which most satisfies the definition is ranked 1, and the one which least satisfies the definition is ranked 5. When two methods satisfy the definition to more or less the same degree, they are given the same ranking.

	$M_{Edge:Centre}$				$M_{Discrepancy}$				$M_{Dispersion}$			
Definition	The closer to 1 is better				The closer to 0 is better				The smaller is better			
Dimension	1D	2D	3D	4D	1D	2D	3D	4D	1D	2D	3D	4D
RRT	1	5§	5§	5§	2	4	5	5	1	1	1	1
FSCS-ART	1	4§	4§	4§	2	3	4	4	1	2	2	2
BPRT	1	1	1	1	1	1	1	1	1	3	3	3
RPRT	1	3†	3†	3†	4	2	2	2	1	4	4	4
RT	1	1	1	1	5	5	3	3	5	5	5	5

† The $M_{Edge:Centre}$ is smaller than 1, so the testing method has a centre bias.
§ The $M_{Edge:Centre}$ is larger than 1, so the testing method has an edge bias.

Table 1. Testing methods ranked according to test case distribution metrics

For those testing methods studied, in the 1D case, it has been observed that RRT has the best performance, followed by FSCS-ART, BPRT, RPRT and then RT. However, when looking at the 2D, 3D and 4D cases, the same performance ordering is observed for small θ, but almost the reverse ordering for large θ. It has been shown in [11] that there exist some hidden factors (unrelated to how evenly a method spreads test cases) which have an strong impact on the performance of ART. Only when θ is small enough, will the performance of ART strongly depend on how evenly spread its test cases are. In order to fairly analyze the relationship between the test case distribution and the performance of ART methods, without being influenced by external factors, the rest of the discussion will be carried out on small failure rates.

Table 1 shows that in terms of $M_{Dispersion}$ metric, RRT has the most even spreading of test cases, followed by FSCS-ART, BPRT and RPRT. In other words, the ranking according to the $M_{Dispersion}$ metric is consistent with the ranking according to F-measures (data shown in Figure 1). It should be pointed out that even though $M_{Discrepancy}$ and $M_{Dispersion}$ are two commonly used metrics for measuring sample point equidistribution, in this study, $M_{Dispersion}$ appears to be more appropriate than $M_{Discrepancy}$.

Interestingly, among all ART methods under study, the one with the largest $M_{Edge:Centre}$ (that is, RRT) has the smallest $M_{Dispersion}$, while the one with the smallest $M_{Edge:Centre}$ (that is, RPRT) has the largest $M_{Dispersion}$. As discussed before, $M_{Dispersion}$ could best reflect the ordering of testing methods with respect to their performance. We notice that pushing test cases away (so that $M_{Edge:Centre} > 1$) is not a bad approach to evenly spreading test cases, even though it may not be the best approach to achieving a real even spreading of test cases.

5. Conclusion

Previous studies [4, 5, 8] showed that even spreading of test cases makes ART outperform RT. The concept of even spreading of test cases is simple but vague. In this paper, several metrics were used to measure the test case distribution of ART as well as RT. The relevance and appropriateness of these metrics were also investigated. To our best knowledge, this is the first work on analyzing the relationship between test case distributions and performance of an ART method. Recently, there were some works on alleviating the edge bias of FSCS-ART [6, 13]. We shall continue the line of this research with additional knowledge gained from this study to enhance the exsiting ART methods.

References

[1] P. E. Ammann and J. C. Knight. Data diversity: an approach to software fault tolerance. *IEEE Transactions on Computers*, 37(4):418–425, 1988.

[2] P. G. Bishop. The variation of software survival times for different operational input profiles. In *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing*, pages 98–107, 1993.

[3] M. S. Branicky, S. M. LaValle, K. Olson, and L. Yang. Quasi-randomized path planning. In *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, pages 1481–1487, 2001.

[4] K. P. Chan, T. Y. Chen, and D. Towey. Restricted random testing: Adaptive random testing by exclusion. *Accepted to appear in International Journal of Software Engineering and Knowledge Engineering*, 2006.

[5] T. Y. Chen, G. Eddy, R. G. Merkel, and P. K. Wong. Adaptive random testing through dynamic partitioning. In *Proceedings of the 4th International Conference on Quality Software*, pages 79–86, 2004.

[6] T. Y. Chen, F.-C. Kuo, and H. Liu. Enhancing adaptive random testing through partitioning by edge and centre. In *Proceedings of the 18th Australian Software Engineering Conference*, pages 265–273, 2007.

[7] T. Y. Chen, F.-C. Kuo, and Z. Q. Zhou. On favourable conditions for adaptive random testing. *Accepted to appear in International Journal of Software Engineering and Knowledge Engineering*.

[8] T. Y. Chen, H. Leung, and I. K. Mak. Adaptive random testing. In *Proceedings of the 9th Asian Computing Science Conference*, pages 320–329, 2004.

[9] G. B. Finelli. NASA software failure characterization experiments. *Reliability Engineering and System Safety*, 32(1–2):155–169, 1991.

[10] R. Hamlet. Random testing. In J. Marciniak, editor, *Encyclopedia of Software Engineering*. John Wiley & Sons, second edition, 2002.

[11] F.-C. Kuo. *On adaptive random testing*. PhD thesis, Faculty of Information and Communications Technologies, Swinburne University of Technology, 2006.

[12] J. Mayer. Lattice-based adaptive random testing. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, pages 333–336, 2005.

[13] J. Mayer and C. Schneckenburger. Adaptive random testing with enlarged input domain. In *Proceedings of the 6th International Conference on Quality Software*, pages 251–258, 2006.

[14] G. J. Myers. *The Art of Software Testing*. Wiley, New York, second edition, 1979.

[15] K. Sen, D. Marinov, and G. Agha. CUTE: a concolic unit testing engine for C. In *Proceedings of 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 263–272, 2005.

[16] D. Slutz. Massive stochastic testing of SQL. In *Proceedings of the 24th International Conference on Very Large Databases*, pages 618–622, 1998.

[17] T. Yoshikawa, K. Shimura, and T. Ozawa. Random program generator for Java JIT compiler test system. In *Proceedings of the 3rd International Conference on Quality Software*, pages 20–24, 2003.

Reducing the Number of Test Cases for Performance Evaluation of Components

João W. Cangussu   Kendra Cooper   Eric Wong
Department of Computer Science
University of Texas at Dallas
Richardson-TX 75083-0688, USA
{cangussu,kcooper,ewong}@utdallas.edu

Abstract

Component-based software development techniques are being adopted to rapidly deploy complex, high quality systems. One of its aspects is the selection of components that realize the specified requirements. In addition to the functional requirements, the selection must be done taking into account some non-functional requirements such as performance, reliability, and usability, among others. Hence, data that characterize the non-functional behavior of the components are needed; a test set is needed to collect this data for each component under consideration. This set may be large, which results in a considerable increase in the cost of the development process. Here, a process is proposed to considerably reduce the number of test cases used in the performance evaluation of components. The process is based on sequential curve fittings from an incremental number of test cases until a minimal pre-specified residual error is achieved. The results from experiments with image compression components are a clear indication that a reduction in number of test cases can be achieved while maintaining reasonable accuracy when using the proposed approach.

1 Introduction

Component-based software engineering (CBSE) techniques hold the promise to support the timely, cost effective development of large-scale, complex systems; they are of keen interest to researchers and practitioners. However, there are numerous issues to address in CBSE including how to specify the functional and non-functional behavior of the components, how to evaluate, rank, and select components, how to predict the interoperability of components, etc.

A key issue in the specification of components is the problem of how to effectively test, or evaluate, the components in order to obtain quantitative data about their behavior. In particular, the non-functional behaviors such as response time performance, memory usage, etc. need to be collected. Recognizing that complete sets of test cases are not possible and large, comphrensive sets of test cases can be prohibitively expensive, techniques to reduce the number of test cases while still providing meaningful information about the components are needed.

Here, we present an approach to select a reduced set of test cases that can be applied to a wide variety of components and non-functional properties. The approach is based on the use of polynomial curve fitting techniques, which are general approaches for representing a curve. The selection of an additional test case, which can be done either adaptively or randomly, is performed iteratively until an error tolerance is reached.

The approach is validated experimentally by selecting a reduced set of test cases for evaluating image compression components. These components implement well known algorithms including Arithmetic Encoding, Huffman, and the Burrows-Wheeler Transform. The non-functional behaviors under test are the compression time (how long does it take to compress a file) and compression ratio (how much smaller is the compressed file compared to the original file). Using our approach, as few as eight test cases are needed and selected in this experiment to capture the dominant behavior of the two non-functional behaviors. The selected test cases have a root mean square error of 0.0254 in comparison to the actual behavior of the component evaluated with a comprehensive set of 190 test cases. These results indicate the accuracy of the approach is excellent and offers a significant reduction in the number of test cases. The performance of the new approach is also experimentally evaluated, comparing the random selection of an additional test case with the adpative approach. Our results indicate that non-linear behavior, e.g., compression ratio, has better performance with the adaptive approach; linear behavor, e.g., compression time, has better performance with a random approach.

The remainder of this paper is organized as follows. The new test case reduction approach is presented in Section 2;

the evaluation of the new approach is in Section 3. Related work is discussed in Section 4. Conclusions and Future work are presented in Section 5.

2 Proposed Approach

The goal of the proposed approach is to reduce the number of test cases needed to conduct a performance evaluation of non-functional behaviors of components. In general, a large number of test cases is needed to obtain a precise evaluation. The conjecture here is that a reasonably accurate evaluation can be achieved with a considerable reduction in the number of test cases. Also, the approach needs to be general; it should not be restricted for use on a specific subset of non-functional attributes

Any non-functional attribute of interest will always be a function of some input parameters, otherwise there is no need for testing. Therefore, the performance evaluation consists of finding the relationship between the input parameter(s) and the performance on the non-functional attribute. Both the input parameter (or some feature of the input parameter) and the non-functional attribute can be quantified and the relationship can be captured by some mathematical function. If the function is known in advance, then it can be directly used for the performance evaluation with no testing needed. However, this is rarely the case. In most scenarios, only the average performance is known which does not provide a comprehensive understanding of the behavior of non-functional attribute. In summary, the goal is to find the relationship using as few test cases as possible. Hereafter, the relationship between a input parameter x and a non-functional attribute y is referred to as $y = f(x)$.

Linear or non-linear regression models [10, 9] could be used to find the parameters of $f(x)$ if the general format of the function is known. For example, if y is known to have an exponential behavior such as ae^{-bx} regression models could be applied to find the values of a and b based on test cases (values of x) and the observed performance (values of y). However, in most cases, this relationship is not known and a more general approach needs to be used. Based on that, polynomial fit [5] has been chosen as, in general, any curve can be represented by a polynomial of a certain degree n, $p(x, n) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$. The problem is now the identification of the coefficients a_i, $i = 0, \ldots n$ of the polynomial. The *polyfit(x,y,n)* function available in MatLab [4] has been used to find the coefficients of $p(x)$ that fits the data points $(x_i, y_i), i = 1, \ldots, m$ in a least square sense. In the case of performance evaluation, m is the number of test cases and the pair (x_i, y_i) represents the input value and the corresponding observed performance. The method of least squares is based on the minimization of errors (least square errors); the distance between the actual point and the point in the fitting curve. The best-fit curve of a given type is the curve that has the minimal sum of errors from a given data set. Suppose a sequence $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$ of m data points are given. The fitting function $f(x)$ has an error e associated with each data point, i.e., $e_1 = y_1 - f(x_1)$, $e_2 = y_2 - f(x_2)$, ..., $e_n = y_n - f(x_m)$. Therefore, using the method of least squares, the best fitting curve has the property of minimizing Π in Eqn. 1.

$$\begin{aligned} \Pi &= d_1^2 + d_2^2 + \cdots + d_n^2 = \sum_{i=1}^{n} d_i^2 \\ &= \sum_{i=1}^{n} [y_i - f(x_i)]^2 \end{aligned} \quad (1)$$

The pseudo-code in Figure 1 presents the steps of the proposed approach. Let us assume that a performance evaluation needs to be done within a pre-specified range $a, \ldots, b$. For example, if image compression components are being considered, one may be interested in images with size ranging from 1K to 10G bytes of memory. The first step is then to select an initial small set of test cases to start with. In our experiments (refer to Section 3) the starting testing suite T is composed of three test cases $T = \{x_1 = a,\ x_2 = \frac{a+b}{2},\ and, x_3 = b\}$. The stopping criterion of the approach is based on the comparison of two consecutive fits. That is, the curve Fit_1 is achieved using the results of test suite $T_k = \{x_1, x_2, \ldots, x_k\}$, then a new curve Fit_2 is computed using the test suite $T_{k+1} = T_k \cup \{x_{k+1}\}$, where x_{k+1} is a new selected test case. Now, the root mean square error (RMSE), as given by Eqn. 2, between Fit_1 and Fit_2 is computed and the cycle stops when this error is less than a pre-specified threshold ϵ. In this case, the last computed test suite is the one that can capture the main behavior of the performance of the non-functional attribute under consideration.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=0}^{N}(x_1 - x_2)^2} \quad (2)$$

The behavior of the proposed solution is depicted in Figure 2. Let us assume $\epsilon = 10^{-2}$. Figure 2(a) shows the second iteration of the algorithm in Figure 1. In this case, the first test suite T_5 has four test cases a new point (test case) marked with a square is selected to compose T_6. The value of the RMSE of the two fits is 0.9457 which is still larger than ϵ. After one more iteration, as seen in Figure 2(b), the RMSE between the fits using T_6 and T_7 results in an error of 0.0674 which means that more test cases are needed. The stopping criterion is reached in the next iteration (see Figure 2(c)) where the error between T_7 and T_8 goes to 0.0036$<\epsilon$. In this case a total of eight test cases is

```
01  begin
02    S = initial set of points and
03        associated outputs
04    error = infinity
05    while error > e
06      Fit1 = polyfit(S)
07      S = S + new selected point
08      Fit2 = polyfit(S)
09      error = RMSE(Fit1,Fit2)
10    end
11    TestSet = S
12  end
```

Figure 1. Test composition algorithm

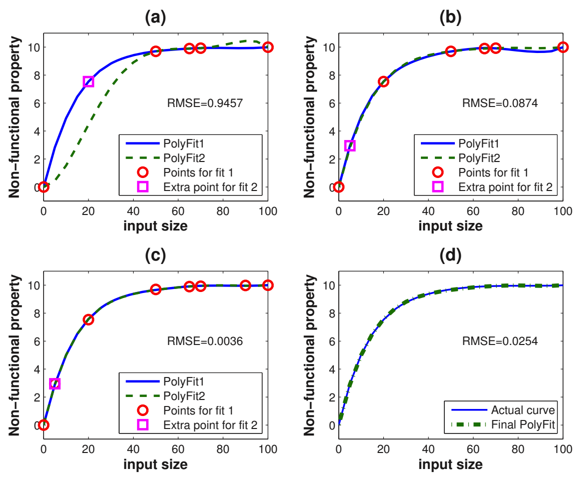


Figure 2. Results from applying the algorithm from Figure 1 to an exponentially shaped curve where (a) represents the results from iteration 2; (b) iteration 3; (c) iteration 4; and (d) is the comparison between the final curve and the actual one.

needed to conduct the performance evaluation. As can be seen from Figure 2(d), the curve computed with the results of only eight test cases is very similar to the actual curve as indicated by RMSE=0.0254. That is, using only eight test cases the dominant behavior of the non-functional attribute has been properly captured.

In the example above, one aspect of the Algorithm in Figure 1 has not been considered: how to select the new test case in Line 06? Two approaches are considered in this paper. The first simply randomly selects the new test case from the available test cases within the specified range. The second approach does the selection in an adaptive way. The largest the gap between two consecutive inputs (assuming the test cases have been sorted), the less information the fitting method has to cover that area. Therefore, the selection approach is to fill this gap and increase the information used by the fitting method. After computing the gaps between each test case, the approach finds the largest gap and then try to find an available test case within this range. This step is needed because not all inputs in the original range from a to b may be available. For example, when testing the image compression with $a = 1K$ and $b = 10G$, images from all these sizes may not be available. Very large images may be hard to find and only 3 images may be available in the range from 5G to 10G. If no input is available, then the algorithm searches for the next largest interval. Notice that this selection approach will degrade to a full binary selection if all input values in the range are available; however, this is rarely the case.

3 Evaluation of the Proposed Approach

Three components for image compression are used here to evaluate the performance of the test case reduction technique described in Section 2. Although a large number of compression techniques exist, the decision to use Arithmetic Encoding (ARI), Huffman coding (HUF), and Burrows-Wheeler Transform (BWT) is based on their generality and availability. Also, two non-functional attributes are used in the examples: compression time and compression ratio. These two have been selected to represent a linear and a non-linear non-functional attribute. A large set of 190 images is available for the performance evaluation. The images range in size from 10 to 10M bytes. The images are not uniformly distributed. The non-uniform distribution is more realistic as smaller images (less than 1M byte) are more common and easier to find than larger images (more than 5M bytes). One image is used for each test case; the goal is to use the least number of test cases as possible while still capturing the dominant behavior of the non-functional attribute.

Figure 3 shows the results of applying both the adaptive approach and the random selection approach to the evaluation of compression time for the three components. The adaptive results are shown in Figures 3(a), (b), and (c) while the random selection results are shown in Figures 3(d), (e), and (f). A polynomial degree of 3 and an error of $\epsilon = 10^1$ have been used. The choice for a third degree polynomial is because it can represent a large variety of curves. Also, although the error may appear to be large at first, it is indeed small when compared to the values of the y axis (10^5 milli-seconds). As we can see in Figures 3(a), (b), and (c), the adaptive approach requires a total of 10, 23, and 15 test cases to capture the behavior of compression time for Huffman, BWT, and Arithmetic encoding, respectively. The observed behavior is linear and in the best case

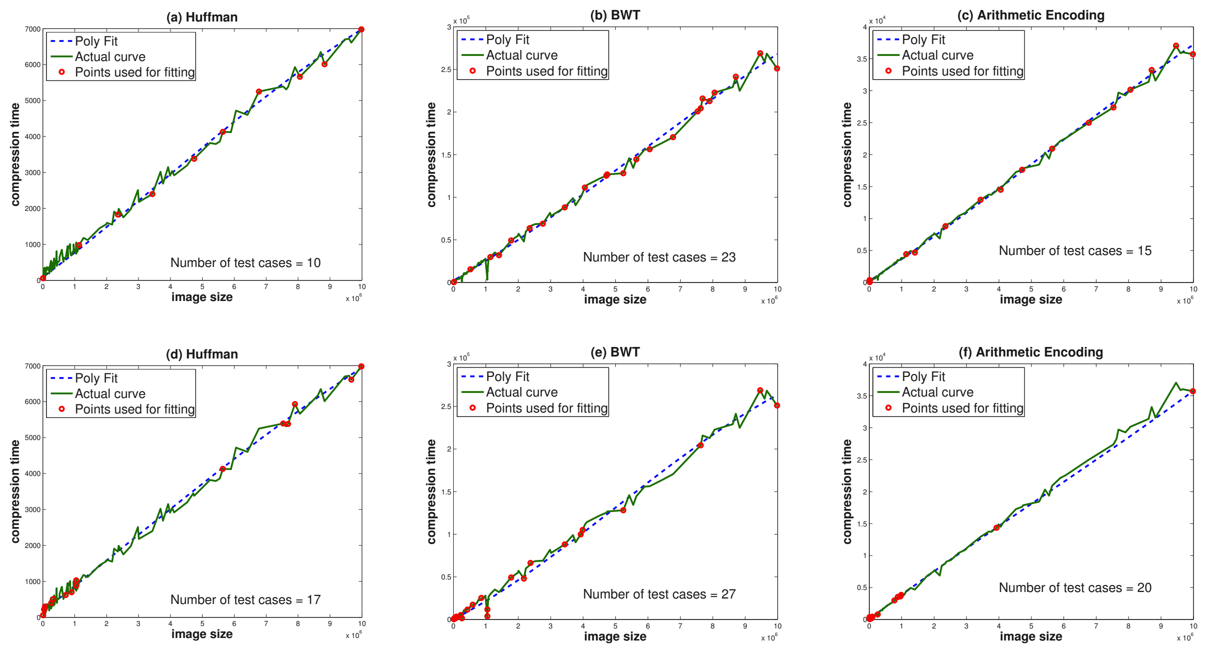


Figure 3. Results of the adaptive selection of test cases for compression time of: (a) Huffman, (b) BWT, and (c) Arithmetic Encoding. Results of the random selection of test cases for compression time of: (d) Huffman, (e) BWT, and (f) Arithmetic Encoding.

scenario only three points (test cases) would be needed to capture the behavior. However, the actual results are not a straight line and the use of only three points could lead to a different slope and possibly to a wrong characterization of compression time. In any case, the number of test cases needed seems reasonable small while appropriate to capture the behavior. The application of the random selection (Figures 3(d), (e), and (f)) resulted in, respectively, 17, 27, and 20 test cases needed for each of the components Huffman, BWT, and Arithmetic encoding. The results in this case are only slightly better for the adaptive selection when compared to the random selection. Only four extra test cases are required for BWF, seven are required for Huffman while the Arithmetic encoding requires five extra test cases. The accuracy of both approaches are almost the same with an average mean square error of 100 between the computed curve and the actual curve from all the available data points. Note again that a 100 units difference is small in the 10^5 scale.

As stated before, compression time for the evaluated components presents a linear behavior with respect to image size. To further evaluate the proposed approach the compression ratio, a non-functional attribute with a non-linear behavior, is analyzed next. In this case a polynomial of degree 3 and an error $epsilon = 10^{-2}$ have been used. The error is smaller because the scale for the y-axis (compression ratio) is comparatively smaller. Figures 4(a), (b), and (c) have present the results from the adaptive selection while Figures 4(d), (e), and (f) are the counterparts for the random selection of test cases. The adaptive selection required 12, 24, and 11 test cases. The results for the random selection present a considerable decline in performance. For the execution runs in Figures 4(d), (e), and (f), a total of 32, 83, and 31 test cases were required. Unlike compression time, the results in this case are much more favorable to the adaptive selection than to the random selection of test cases. Adaptive selection performs 2.6 times better than random selection for the Huffman component. The improvements for BWT and Arithmetic encoding are, respectively, 3.4 and 2.8.

When considering compression ratio, the adaptive approach has performed considerably better than the random approach. This is due to the non-uniform distribution of the image sizes and the non-linearity of the requirement under consideration. In general, when test cases are uniformly distributed, the adaptive and the random selection approaches will have a similar behavior. The adaptive approach tries to fill the largest interval between two input values, which tends to make the selected inputs uniformly distributed. Therefore, if the inputs are already uniformly distributed, it is expected that random selection will behave in a similar manner. However, this is not the case for the compression ratio and since images are clustered, the selection of a new image may lead to a large difference between

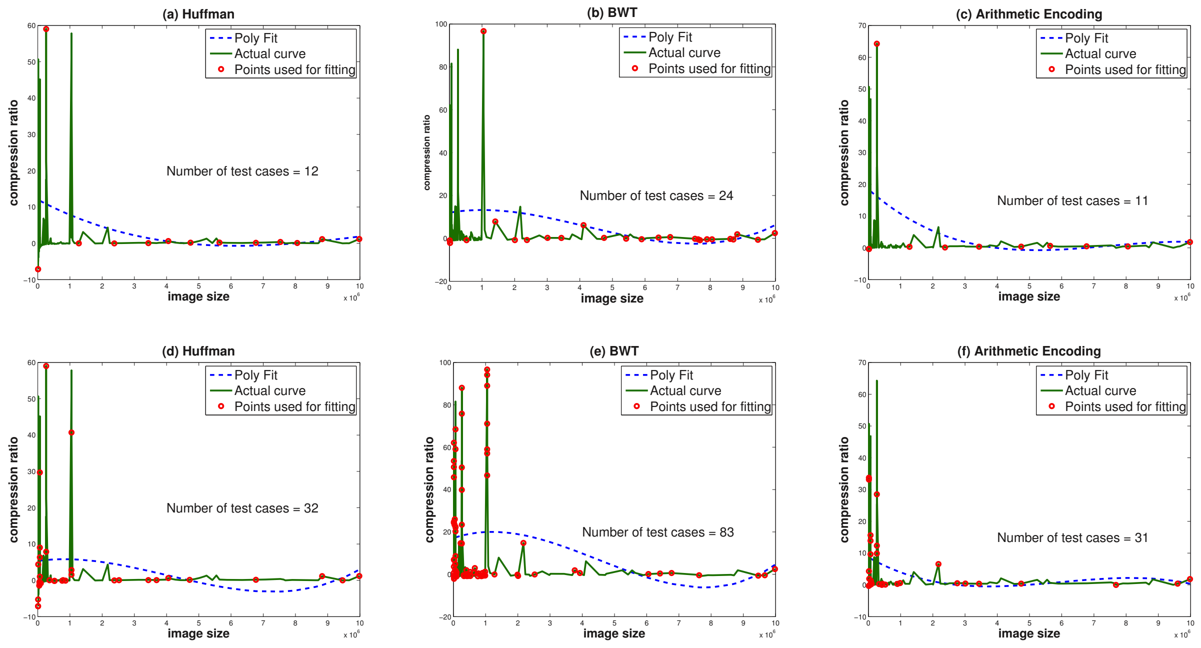


Figure 4. Results of the adaptive selection of test cases for compression ratio of: (a) Huffman, (b) BWT, and (c) Arithmetic Encoding. Results of the random selection of test cases for compression ratio of: (d) Huffman, (e) BWT, and (f) Arithmetic Encoding.

the two polynomial fits and consequently a larger number of test cases will be required. The expectation is that the more complex the behavior of the requirement (meaning, the more complex the curve to fit), the larger the number of test cases (points) needed to capture its behavior. The non-linearity of compression ratio is an indication of this complexity leading to a larger number of test cases required to capture its behavior.

The results presented in this Section provide a good indication that the new test reduction approach can select a small number of test cases to conduct an accurate performance evaluation of components. However, more extensive evaluation of the proposed approach needs to be conducted to further verify this conjecture. This has been deferred to future work. Another aspect that still needs additional evaluation is the impact of the values of the degree of the polynomial and the value of the error ϵ on the performance of the approach. The conjecture is that the higher the degree of the polynomial and the smaller the value of ϵ the larger the number of test cases required to capture the behavior. Once this conjecture is confirmed (with future case studies and simulation runs) the next step would be the optimization of these values. That is, given a specific non-functional attribute, a certain degree for the polynomial and a certain value for ϵ should be found to minimize the number of test cases required while maximizing the accuracy of the fit. This problem is also deferred to future work.

4 Related Work

To the best of our knowledge, we are not aware of any studies with a similar objective as what is reported here. The rest of this section focuses on work in three categories: test cost reduction, adaptive testing, and component-based testing.

Test Cost Reduction: Marre et al. [8] used a spanning set of entities to generate test suites in order to estimate and to reduce the cost of testing based on the observation that one test case generally covers more than one entity. As a result, if we can identify "a subset of entities with the property that any set of tests covering this subset covers every entity in the program," then we can reduce the cost of testing. They presented a method for finding a minimum set of entities of full coverage and an automated method for finding the corresponding spanning set.

Ling et al. [7] proposed a decision-tree learning algorithm to build more effective decision trees in order to minimize the sum of the misclassification cost and the test cost. Their algorithm is based on cost-sensitive learning methods such as a Markov Decision Process. They also explained the problems of other approaches and claimed that their algorithm is superior to the others.

Adaptive Testing: Adaptive Random Testing is an active research topic because of its effectiveness under some well distributed input domains [1] The actual results depend on the "distances" between different test values. However, such distances have only been defined for integers and other elementary values. Ciupa et al. [2] extended this idea by introducing an "object distance" to test object-oriented programs. A Distance-Based Adaptive Random Testing (D-ART) method was also proposed based on object distance.

Component-based Testing: Damm et al. [3] proposed a framework for automated component testing. This approach is based on Test-Driven Development (TDD) which creates the test cases before developing software components. The proposed framework extends the traditional TDD (which is for each class and method) to the component level, which needs to test for each component interface. As a result, defects can be detected earlier in the development cycle to reduce the overall cost of testing and debugging.

One difficulty of testing software components among others is testing them under numerous hardware, operating systems, and third-party COTS components. Grundy et al. [6] proposed an approach using a "validation agent" and a "component aspect" to resolve this problem. They are used at the deployment time to validate the components. The validation agent tests functional and non-functional aspects of software components in an actual deployment situation, whereas the component aspect cross-cuts the aspects of the components to increase its usability.

5 Conclusions and Future Work

A new approach that selects a reduced set of test cases for the accurate performance evaluation of components has been presented in this work. The approach is based on the use of polynomial curve fitting techniques. The approach begins by using a small set of initial test cases. Using an iterative approach, a new test case is found and added to the test case set. The new test case may be selected either randomly or using an adaptive technique. The test cases are executed; the performance evaluation results are used to compute a new curve. When the RMSE between the previous and the current curves are below a threshold, then the reduced test case set has been found.

The approach has been experimentally validated using a set of components that implement well-known image compression algorithms. Using our approach, as few as eight test cases are needed and selected in this experiment to capture the dominant behavior of the two non-functional behaviors. These results indicate the accuracy of the approach is excellent and offer a significant reduction in the number of test cases.

The performance of the new approach has also been experimentally evaluated, comparing the random selection of an additional test case with the adpative approach. Our results indicate that non-linear behavior, e.g., compression ratio, has better performance with the adaptive approach; linear behavor, e.g., compression time, has better performance with a random approach.

There are several interesting directions for future work. The first is to apply the approach to additional sets of components and evaluate different non-functional attributes, to improve the validation of the approach. The second is to investigate the impact of the values of 1) the degree of the polynomial used in the curve fitting calculation and 2) the error threshold. When the impact of these values are thoroughly quantified, then the values for non-functional attributes could be optimized.

References

[1] T. Y. Chen, H. Leung, and I. K. Mak. Adaptive random testing. In *Lecture Notes in Computer Science, 3321:320-329*, 2004.

[2] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer. Object distance and its application to adaptive random testing of object-oriented programs. In *Proceedings of the 1st International Workshop on Random Testing*, 55-63 July 2006.

[3] L. Damm and L. Lundberg. Results from introducing component-level test automation and test-driven development. *Journal of Systems and Software*, 2006.

[4] Walter Gander, J. H. Masaryk, and J Hrebicek. *Solving Problems in Scientific Computing Using Maple and MATLAB*. Springer-Verlag, 1997.

[5] Walter Gautschi. *Numerical Analysis: an introduction*. Birkhauser Boston, Cambridge, MA, USA, 1997.

[6] J. Grundy, G. Ding, and J. Hosking. Deployed software component testing using dynamic validation agents. *Journal of Systems and Software*, 2005.

[7] C. X. Ling, Q. Yang, J. Wang, and S. Zhang. Decision trees with minimal costs. In *Proceedings of the 21st International Conference on Machine Learning*, 69-76 July 2004.

[8] M. Marre and A. Bertolino. Reducing and estimating the cost of test coverage criteria. In *Proceedings of the 18th International Conference on Software Engineering*, 486-494 May 1996.

[9] G. A.F Seber and C. J. Wild. *Nonlinear Regression*. John Wiley & Sons, Inc., 2006.

[10] George A. F. Seber and Alan J. Lee. *Linear Regression Analysis*. Wiley-Interscience, second edition edition, 2003.

Combining Decorated Classification Trees with RCPS Stochastic Models to Gain New Valuable Insights into Software Project Management

Antonio Juarez Alencar, Gelson Guedes Rodrigues, Eber Assis Schmitz and Armando Leite Ferreira
Institute of Mathematics, Electronic Computer Center and The COPPEAD School of Business
Federal University of Rio de Janeiro
P.O. Box 68530 - 21941-590 - Rio de Janeiro - RJ, Brazil
juarezalencar@br.inter.net, gelsongr@gmail.com, eber@nce.ufrj.br, armando@coppead.ufrj.br

Abstract

This work presents a process that combines decorated classification trees with RCPS (Resource Constrained Project Scheduling) stochastic modeling and simulation to provide project managers with better insights into the project they run. Such insights make it easier for managers to anticipate changes in planning that favor projects to be delivered on time and in compliance with the requirements they were set to satisfy.

1. Introduction

Over the last few decades the environment in which organizations do business has changed considerably with enormous consequences for project management. The business paradigms that prevailed during the industrial revolution are giving way to new ones dictated by the information age, knowledge age and technology revolution which we are currently experiencing. For example, the existence of rigid production lines of impersonalized tangible products that characterized the industrial revolution are being successfully challenged by increasing demand for highly customized products and services, decentralized work force and intangible products, opening new vistas for industrial and business development [5].

To remain competitive under these circumstances organizations have to innovate constantly. However, the demand for new and more complex products calls for the execution of more complex projects; requiring the coordination of efforts from multidisciplinary teams with advanced technical and business skills, the establishment of strategic alliances with external partners, the outsourcing of project activities, and the use of recently developed technology.

Moreover such constrains tend to put projets managers under enormous pressure to produce results. When the pressure to deliver becomes too great, common sense often goes out the window, crucial steps in the project development are ignored, the end result is shoddy, and the rework required to repair the damage is far more expensive than if it were caught in the planning stages [9].

Effective project management in a lean resource environment requires good planning and timely information, allowing problems to be anticipated and dealt with before the worse happens. To manage projects more effectively, over the years, IT professionals have resorted to a variety of planning methods such as Gantt Charts, CPM (Critical Path Method), PERT (Performance Evaluation and Review Technique) and, more recently, to RCPS (Resource Constrained Project Scheduling) stochastic modeling and simulation [6].

This work shows how classification trees, a family of non-parametric statistical methods, can be decorated with financial information and used together with RCPS stochastic modeling and simulation to provide valuable insight into project planning in constrained environments, making it easier for managers to foresee changes in planning that help projects to be delivered within the allowed timeframe and in accordance with the requirements they were set to satisfy.

2 The RCPS Problem

One of the main goals of project scheduling is to produce a detailed plan of project related activities with the view of allowing managers to deal with a large variety of problems before the worse happens. The most frequent problem managers use project scheduling to solve is the minimization of makespan[1], other common possibilities include minimization of cost, maximization of financial results, maximization of quality measures, etc. [13].

Projects that are subjected to precedence and resource constraints are called "Resource-Constrained Project

[1]Total project duration.

Scheduling" in the literature, or RCPS for short. Not surprisingly, the vast majority of software projects in the real world face RCPS problems. Over the years numerous methods have been proposed to solve RCPS problems, such as: implicit enumeration, branch-and-bound, schedule generation schemes (SGS), X-pass, etc. A comprehensive review of these methods is found in [7]. Despite the differences these methods may have, they can all be classified in just two categories: exact methods and heuristic methods [3].

Exact methods are used to find the precise maximum or minimum value of a variable under consideration. For example, the minimum project duration or its maximum financial return. However, these methods have considerable limitations that become evident when projects have a great number of activities or complex resource constraints. Usually, in these circumstances a solution cannot be found within reasonable computing time.

Heuristic methods, on the other hand, use a schedule priority criteria to provide an approximate solution to RCPS problems. They are particularly useful when the use of exact methods are not feasible. In the heuristic methods every activity is initially a potential candidate to be scheduled. However, activities can only be scheduled if all its precedent activities have been completed and the resources required for its execution are fully available. Unfortunately, whenever activities share resources, it may imply that they cannot be executed concurrently.

The use of a heuristic is then necessary to decide when each activity should receive its required resources and be executed as a result. When no real candidate activities are available, the clock advances until one of the activities in progress is completed. At that point in time resources are freed and the process is repeated, i.e. candidate activities are identified, resources are checked for availability and activities are scheduled. The total duration of a project is the time required for the completion of all its activities. As a mathematical problem RCPS belongs to a category where, in general, an optimal solution cannot be found in a linear or polynomial time frame, i.e. the NP-hard category [1].

3 Classification Trees

The techniques described in this article to analyze RCPS stochastic models make extensive use of classifications trees, a family of statistical inferential methods conceived by Morgan and Sonquist in the 1960's and later perfected by others [8].

As an inferential method, classification trees seek to explain the behavior of a target variable by combining different values of a given set of predictive variables. To achieve this goal, the values of the predictive variables are successively combined in such a way that an n-dimensional space is portioned into increasingly more homogenous sets of values with respect to the target variable.

The way the portioning is done allows the final result to be presented as a flow-chart whose format resembles a tree, so the name of the family of methods. Furthermore, the information presented in the flow-chart may be easily translated into a set of rules that indicate the likelihood of occurrence of values in the domain of the target variable for different circumstances.

Classification trees are non-parametric methods, i.e. no restrictions are imposed on the distribution of values of predictive and target variables. In addition, these variables are allowed to hold all sort of relationships among themselves. All of this makes it easier to use classification trees to solve problems in the real word, where the distribution of values of variables and the relationships that they hold among themselves are frequently unknown. Moreover, classification trees have been widely reported as presenting satisfactory results even in the presence of noise and when little data is available; making it a very attractive class of methods to be used in all sorts of different situations [12].

4 Mining RCPS Stochastic Models

According to Seneca (4 BC-AD 65), the Roman philosopher: "rules make the learner's path long, while examples make it short and successful". As a result, the mining process presented in this article is introduced with the help of a real-world inspired example. Consider a chain of furniture stores that uses catalog marketing to increase its sales. On a regular basis this company edit a catalog with a variety of selected products that are sent to a large group of potential buyers, who are selected from the company's database. The proper undertaking of this task requires that eight activities are efficiently executed within a tight time-frame, i.e.

1. *Product Selection* - that chooses the products that will be advertised in the catalog;
2. *Prospect Selection* - that identifies the prospects to whom the catalogs are going to be mailed;
3. *Pricing* - that establishes the promotional price of every product to be advertised in the catalog;
4. *Catalog Design* - where the graphic and textual aspect of the catalog, and accompanying advertising material are conceived and put together;
5. *Label Printing* - where labels with prospects' names and addresses are printed and organized;
6. *Stock Control* - that makes sure that the products advertised in the catalog will be available for shipping when they are ordered ;
7. *Catalog Printing* - where the actual print of the catalog is done;
8. *Catalog Labeling & Mailing* - which labels the catalogs with prospects' names and addresses and sends them to their intended destinations over the mail.

Tough competition in the furniture business has brought down the company's profit margins over the years. As a result, the chain of furniture stores believes that its survival depends upon the efficiency of its business processes.

With the view of increasing the efficiency of its catalog marketing campaigns, the company has decided to develop a system of software tools that, working together, provide adequate support for the activities leading to the roll-out of its marketing campaigns. Because each of these activities is supported by a different software tool, altogether, eight tools have to be built in such a way that information made available by one tool may be used by others.

Unfortunately, due to lack of adequate funding, initially only two people have been selected to work on the development of the software tools. They are going to be named *Mimi* (a systems analyst) and *Ed* (a computer programmer). Table 1 shows the human resources required for the development of each tool.

Tool		**Resource**
Name	**Supported Activity**	**Required**
PdS	Product Selection	Mimi and Ed
PsS	Prospect Selection	Mimi
P	Pricing	Mimi and Ed
CD	Catalog Design	Ed
LP	Label Printing	Mimi and Ed
SC	Stock Control	Mimi
CP	Catalog Printing	Ed
CLM	Catalog Labeling & Mailing	Mimi and Ed

Table 1. Resources required.

Figure 1 presents the network of activities concerning the development of the software tools. In that figure *PdS* is the first tool to be developed and *CLM* the last. Moreover, an arrow connecting two activities such as *PdS* $\longrightarrow$ *P* indicates that the latter development efforts may only start when the former has been completed and all the necessary resources are available.

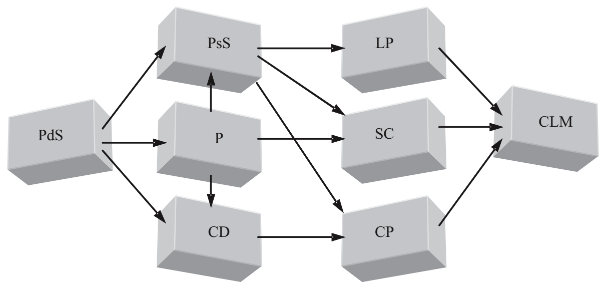


Figure 1. The project's network of activities.

As this project has to be completed within a strict timeframe, it is crucial that management is made aware of its expected makespan in respect to the current resource constraints and the duration of the different project activities. However, because these activities have not been executed yet, their duration can only be estimated. In this case, with the support of other experienced software project managers and a database of previously executed projects, a three point estimate has been established for each activity, indicating their minimum, most likely and maximum expected duration. Table 2 presents these figures.

Project	**Estimated Duration (time units)**		
Activity	**Minimum**	**Most Likely**	**Maximum**
PdS	2	3	6
PsS	3	6	8
P	4	6	7
CD	1	8	10
LP	7	9	11
SC	5	6	7
CP	4	5	8
CLM	1	2	4

Table 2. Duration of project activities.

Considering that the exact duration of each activity is unknown and that the available resources may be insufficient to ensure that all activities are executed when their precedents are completed, a Monte Carlo stochastic simulation model has been built to analyze the project makespan. In this model, the duration of each activity is described by a triangular probability density function, one of the most widely used functions to describe activity duration [4].

Subsequently the stochastic model was subjected to a simulation process, where resources were granted to the first activity to place a request for them. During simulation the duration of each activity was recorded in the form of a percentage of their respective time range, together with the outcome of each simulated scenario, indicating whether the project finished within the allowed time.

Variable	**Scenario**					
	1	**2**	**3**	**4**	...	**n**
PdS	0.912	0.475	0.660	0.132	...	0.432
PsS	0.534	0.574	0.942	0.444	...	0.236
P	0.160	0.856	0.650	0.270	...	0.260
CD	0.673	0.705	0.758	0.724	...	0.689
LP	0.135	0.617	0.602	0.515	...	0.502
SC	0.305	0.600	0.480	0.525	...	0.445
CP	0.207	0.127	0.192	0.797	...	0.217
CLM	0.383	0.596	0.156	0.456	...	0.560
#Ed	3	1	3	2	...	1
#Mimi	2	3	2	1	...	2
Outcome	In	Out	In	Out	...	Out

Table 3. Simulation results.

Table 3 presents the figures collected during simulation. Table 4 describes the meaning of the columns presented in Table 3. For example, as indicated by the variable *CD*, in the first simulated scenario the development of the *CD* tool consumed 67.3% of its time range, i.e. $1 + 0.673 \times (10 - 1) = 7.05$ time units.

Figure 2 shows the project makespan classification tree built with the help of Breiman *et al.* approach to classification [2]. In that figure the box labeled *Node 0* is the root of the classification tree. It contains the total number of observations available for analysis, i.e. 6,000 in this case. Also, it displays the number and proportion of the different scenarios in which the project finished and did not finish on time. For example, initially, in 50.2% of the generated scenarios the project finished on time, whilst in 49.8% it did not.

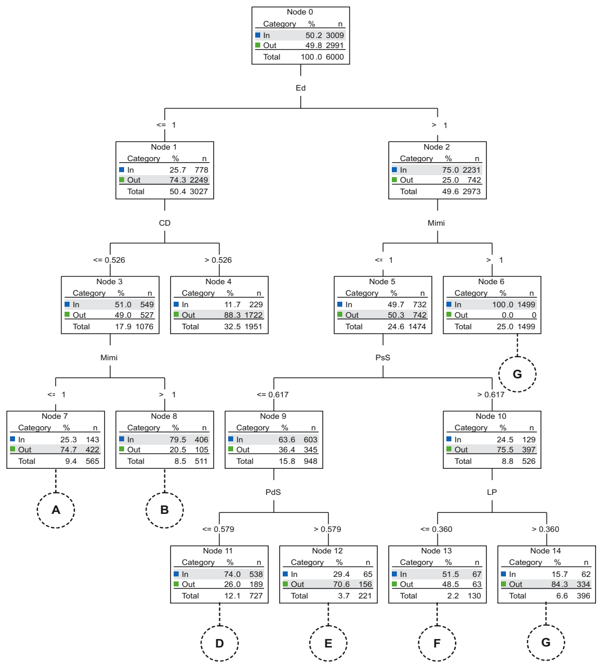
Variable	**Meaning**
PdS, *PsS*, $\cdots$, *CLM*	Duration of activities *PdS*, *PsS*, $\cdots$, *CLM* expressed as a percentage of their respective time range.
#Mimi and *#Ed*	Number of *Mimi* (systems analysts) and *Ed* (computer programmers) used in the project. These figures vary from 1 to 3.
Outcome	The outcome of a given scenario, where In indicates that the project finished within the allowed time-frame and Out indicates otherwise.

Table 4. The simulation variables.

It should be noted that the relation that is used to partition the initial set of scenarios is $Ed \leq 1$, and that, as a consequence, *Node 1* contains the 3,027 observations for which this relation holds and *Node 2* the remaining 2,973, i.e. 50.4% and 49.6% of the simulated scenarios respectively.

Because of the incremental way in which the tree is constructed, all relations that hold for the observations in a node also hold for the observations in its descendants nodes. For example, in all the scenarios that are part of *Node 3* the number of resources Ed used to run the project is smaller or equal to 1 and the time spent in the development of the *CD* tool is smaller than or equal to 52.6% of its estimated time range (*CD* ≤ 0.526). Table 5 shows all the relations that hold for the leaves of the tree presented in Figure 2, together with the proportion of In's and Out's, and the proportion of the total simulated scenarios for with those relations hold.

Due to the random nature of the Monte Carlo simulation, these proportions are actually estimated likelihoods of the software project finishing on time in difference circumstances [4]. For example, the proportion of simulated scenarios where *Ed* ≤ 1 and *CD* > 52.6 is 32.5% and in only 11.7% of these scenarios the project finished within the allowed time-frame, therefore the estimated likelihood of the occurrence of a scenario where these two relations hold is $0.325 \times 0.117 = 0.038$.



Node 0 Category % n; In 50.2 3009; Out 49.8 2991; Total 100.0 6000
Ed
<= 1
> 1
Node 1 Category % n; In 25.7 778; Out 74.3 2249; Total 50.4 3027
Node 2 Category % n; In 75.0 2231; Out 25.0 742; Total 49.6 2973
CD
Mimi
<= 0.526
> 0.526
<= 1
> 1
Node 3 Category % n; In 51.0 549; Out 49.0 527; Total 17.9 1076
Node 4 Category % n; In 11.7 229; Out 88.3 1722; Total 32.5 1951
Node 5 Category % n; In 49.7 732; Out 50.3 742; Total 24.6 1474
Node 6 Category % n; In 100.0 1499; Out 0.0 0; Total 25.0 1499
Mimi
PsS
G
<= 1
> 1
<= 0.617
> 0.617
Node 7 Category % n; In 25.3 143; Out 74.7 422; Total 9.4 565
Node 8 Category % n; In 79.5 406; Out 20.5 105; Total 8.5 511
Node 9 Category % n; In 63.6 603; Out 36.4 345; Total 15.8 948
Node 10 Category % n; In 24.5 129; Out 75.5 397; Total 8.8 526
A
B
PdS
LP
<= 0.579
> 0.579
<= 0.360
> 0.360
Node 11 Category % n; In 74.0 538; Out 26.0 189; Total 12.1 727
Node 12 Category % n; In 29.4 65; Out 70.6 156; Total 3.7 221
Node 13 Category % n; In 51.5 67; Out 48.5 63; Total 2.2 130
Node 14 Category % n; In 15.7 62; Out 84.3 334; Total 6.6 396
D
E
F
G

Figure 2. The makespan classification tree.

It follows that according to the information displayed in Figure 2, the likelihood of the software project finishing on time is 0.502, or 50.2%. See *Node 0*. Because this falls short of the 70% likelihood standard established by the company the project manager is compelled to act. There are several actions open to management. For instance, they can try to convince the stakeholders to extend the project duration. Also, at some extra cost, they can hire more people to work on the project and extend the working effort from 5 days of 6 hours a week to 6 days of 8 hours a week during the development of certain tools (an increase of 60% in the working hours).

Data collected during the simulation process (total project duration and the duration of the development of the *CD*, *PsS* and *PdS* tools) together with the dollars per time unit charged by Ed and Mimi (see Table 6), allow the construction of Table 7 connecting actions and extra cost to the leaves of the tree in Figure 2, helping management to consider the most suitable option to increase the chances of delivering the project on time [11].

5 Discussion

Bellow one finds the answer to some key questions about the implications of the insights provided by decorated classification trees for both project management and project planning.

5.1 What are the steps leading to decorated classification trees?

With the view of helping managers to anticipate changes in software project planning that increase the chances of finishing the project on time one may take the following steps: (a) build a stochastic model that properly represents the project's activities, their interdependency relations and the resource constrains to which they are subjected; (b) run a simulation process that, for each scenario, collects the project outcome indicating whether it finished on time or not; (c) use the data collected during the simulation process to construct a classification tree that has the project outcome as its target variable; (d) examine the tree carefully looking for ways to improve the chances of finishing the project on time; (e) decorate the tree connecting actions and extra cost to the leaves; (f) act on the most suitable actions [11], making favorable but unlikely scenarios more likely to happen.

Node	Rule	Outcome (%)		
		In	Out	Prop.
4	*Ed* ≤ 1 and *CD* > 52.6	11,7	88.3	32.5
6	*Ed* > 1 and *Mimi* > 1	100.0	0.0	25.0
7	*Ed* ≤ 1 and *CD* ≤ 52.6 and *Mimi* ≤ 1	25.3	74.7	9.4
8	*Ed* ≤ 1 and *CD* ≤ 52.6 and *Mimi* > 2	79.5	20.5	8.5
11	*Ed* > 1 and *Mimi* ≤ 1 and $PsS \leq 61.7$ and $PdS \leq 57.8$	74.0	26.0	12.1
12	*Ed* > 1 and *Mimi* ≤ 1 and $PsS \leq 61.7$ and $PdS > 57.8$	29.4	70.6	3.7
13	*Ed* > 1 and *Mimi* ≤ 2 and $PsS > 61.7$ and $LP \leq 36.0$	51.5	48.5	2.2
14	*Ed* > 1 and *Mimi* ≤ 1 and $PsS > 61.7$ and $LP > 36.0$	15.7	84.3	6.6

Table 5. Rules indicating the likelihood of the project finishing on time.

Weekly Working Hours	Dollars / Time Unit	
	Mimi	Ed
Regular (30h)	1,875	1,125
Extended (48h)	2,500	1,500

Table 6. Cost of Mimi and Ed

5.2 How do project managers benefit from decorated classification trees?

One of the main benefits of building an RCPS stochastic model is the insight it can provide on the dependency relations that exists among the duration of project activities and the project outcome. The understanding of these relations allows the identification of activities that most strongly influence the outcome of a project, prompting changes in planning that increase the chances of projects being delivered on time.

However, even the construction and analysis of small RCPS stochastic models require considerable knowledge of mathematics and statistics. As a result, many project managers are unable to enjoy the benefits of using this truly powerful tool, because they just lack the necessary technical skills. In addition, the two statistical methods that are most frequently employed to identify cause and effect relationships among variables in RCPS stochastic models are linear correlation and regression. The proper use of these methods to analyze the relationship that exists among the duration of project activities and project outcome requires these relations to be linear, which most often is not the case.

Moreover, linear regression requires predictive variables to be independent, i.e. the value that one of these variables may take may not depend upon the value that other predictive variables take. Unfortunately, there is a natural tendency that the duration of project activities depend upon each other. For example, in many industries time spent on planning and designing tends to strongly influence time spent on testing and delivery. Therefore, situations in which the proper use of linear regression is welcome do not come by as frequently as one may hope for.

Finally, both correlation and linear regression do not deal

Action	Description	Extra Cost	
		Min	Max
A, C, E, F and G	Talk the stakeholders into extending the allowed project duration.	$0.00	
B	Hire another *Mimi* and increase the working journey during the development of the *CD* tool.	$34,500	$66,500
D	Hire nother *Ed* and increase the working journey during the development of the *PsS* and *PdS* tools.	$26.500	$46,800
H	Hire another *Ed* and another *Mimi*.	$ 74,300	$ 120,000

Table 7. Possible actions and their respective cost with 80% degree of confidence.

easily with categorical variables. Linear regression requires these variables to be transformed into a set of independent variables. Linear correlation requires variables to be of ratio scale, while rank order correlation requires them to take value in an ordinal set. See [10] for an introduction to regression and correlation.

Although, decorated classification trees cannot make the construction of RCPS stochastic models any easier, they do help with the analysis of the simulation process, particularly with the extraction of information that favors tactical changes in project planning. The use of such trees does not require predictive variables to be independent nor to hold a linear relation with the target variable. Also, they can take value in any kind of set, despite its scale of measurement being nominal, ordinal, scalar or ratio. All of this not only greatly facilitates the analysis of RCPS stochastic models, but also makes it faster and less prone to errors.

5.3 Can the technique described in this article be used as a negotiation tool?

Despite all the efforts that project managers may place on their tasks, projects are full of uncertainties that make planning a difficult endeavor. Not only are projects concerned with interrelated events and activities that have not happened yet, but also once these activities have been completed, they will not be executed in the future exactly the same way as in the past. Hence, projects are very unlikely to run as planned, requiring frequent adjustments in the course of time. The more complex and lengthy the project, the more adjustments it is likely to require.

Because the rules generated by classification trees are conjunctions of logical expressions that are easy to read and understand, and can be presented to non-technical personnel, the decorated tree may be used as an awareness tool to make senior management more conscious of critical aspects of a project. In particular of those aspects where further investment is necessary.

6 Conclusion

This article demonstrates the viability of successfully combining decorated classification trees with RCPS stochastic modeling and simulation to provide project managers with the means to anticipate changes in project planning that favor projects being finished on time. This combination of mathematical tools may be used to analyze project makespan with many advantages of more classical methods.

Classification trees are not difficult to build as predictive variables are not required to have any particular distribution of values or hold any kind of relationship among themselves. Moreover, the rules generated by classification trees are easy to read, understand and communicate to all interested parties, helping to avoid mis-communication among team members and also with the identification of most needed adjustments in project planning.

Furthermore, easy to understand rules connected to financial information favor the involvement of senior management with critical aspects of project planning and execution, making it easier for project managers to ensure that the necessary investments are made where and when they are most needed. All of this makes decorated classification trees a very attractive tool to be used in combination with RCPS stochastic modeling and simulation to support the management of complex software projects in the real world.

References

[1] J. Blazewicz, J. Lenstra, and A. R. Kan. Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

[2] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman & Hall/CRC Press, January 1984.

[3] A. L. Castillo and D. F. Mu$\tilde{n}$oz. A decision support system to schedule operations in water heater manufacturing. In S. T. . G. Hart, editor, *IIE Annual Conference and Exhibition*, Houston, Texas, May 2004. Institute of Industrial Engineers.

[4] C. A. Chung. *Simulation Modeling Handbook: A Practical Approach*. CRC Press, July 2003.

[5] Y. A. Hosni and T. Khalil. *Management of Technology - Internet Economy: Opportunities and Challenges for Developed and Developing Regions of the World*. Elsevier Science, June 2004.

[6] H. Kerzner. *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*. John Wiley & Sons, 8^{th} edition, January 2003.

[7] R. Kolisch and S. Hartmann. Experimental investigation of heuristics for resource- constrained project scheduling: An update. *European Journal of Operational Research*, 174:23–37, 2006.

[8] W. Y. Loh, Y. S. Shin, and T. S. Lim. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning*, 40:203–228, September 2000.

[9] Info-Tech Research Group. *Effective Project Management: Tools, Templates & Best Practices*. Info-Tech Research Group, October 2003. Technical Report.

[10] J. T. McClave and T. Sincich. *Statistics*. Prentice Hall, 10^{th} edition, February 2005.

[11] J. X. Wang. *What Every Engineer Should Know About Decision Making Under Uncertainty*. CRC, July 2002.

[12] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, June 2005.

[13] G. Zhu, J. F. Bard, and G. Yu. Disruption management for resource-constrained project scheduling. *Journal of the Operational Research Society*, (56):365–381, 2005.

Towards a Reference Architecture for Software Testing Tools[*]

Elisa Yumi Nakagawa[†] Adenilso da Silva Simão, Fabiano Ferrari, and José Carlos Maldonado
Dept. of Computer Systems
USP - University of São Paulo
São Carlos/SP, Brazil
{elisa, adenilso, ferrari, jcmaldon}@icmc.usp.br

Abstract

Producing high quality software systems has been one of the most important software development concerns. Software testing is recognized as a fundamental activity for assuring software quality; however, it is an expensive, error-prone, and time consuming activity. For this reason, a diversity of testing tools has been developed, however, they have been almost always designed without an adequate attention to their evolution, maintenance, and reuse. In this paper, we propose an aspect-based software architecture, named RefTEST (Reference Architecture for Software Testing Tools), that comprises the knowledge to develop testing tools. This architecture is strongly based on separation of concerns and aspects, aiming at evolving, maintaining and reusing efforts to develop these tools. Our experimental results have pointed out that RefTEST can contribute to the development and reengineering of testing tools.

1. Introduction

Software engineering activities have been fundamental to achieve high quality systems development. In special, software testing is one of the most important activities to guaranteeing the quality and the reliability of the software under development [15]. In this context, the availability of tools makes the testing a more systematic activity and minimizes the cost, the time consumed, as well as the errors caused by human intervention. Testing automation is an important issue related to the quality and productivity of the software process. A diversity of testing tools — commercial, academic, and open source — automating software testing tasks can be found. However, these tools have almost always been implemented individually and independently, presenting its own architectures and internal structures. As a consequence, difficulty of integration, evolution, maintenance, and reuse of these tools are very common. These tools often focus on automating specific testing techniques and criteria; without considering the whole testing process. Besides, while software architecture has become an increasingly important research topic in recent years, contributing to software quality [19], the investigation and establishment of software architectures for testing tools have been largely ignored. As known, the establishment of architectures, specially reference architectures, for a given domain consolidates the knowledge about how to develop tools to that domain, contributing to the reuse of design expertise.

Recently, Aspect-Oriented Programming (AOP) has arisen as a new technology to support a better separation of concerns and to more adequately reflect the way developers reason about the system [11], to contribute to maintainability, reusability, and easiness to write software. Besides programming, aspects have also been explored in the early life cycle phases including the requirements analysis, domain analysis and architecture design phases.

In this paper, we investigate the use of aspect in architecture design phase and present an aspect-based reference architecture, named RefTEST (Reference Architecture for Software Testing Tools), that supports the development of software testing tools. Aspect-based architecture refers to architectures that use aspects — from AOP — as mechanism to establish the communication among the modules that compose this architecture. RefTEST is also strongly based on separation of concerns, aiming at providing reusability, maintainability, and capability of evolution to the testing tools built based on this architecture. For the purpose of communicating adequately the knowledge into RefTEST, architectural views were developed and are presented here.

The remainder of this paper is organized as follows. At first, the background about software architecture and related work are presented. Next, we present the proposed archi-

[*]This work is supported by Brazilian funding agencies CAPES, FAPESP and CNPq.

[†]She is also professor of the Dept. of Administrative Science and Technology, Uniara - Araraquara University Center, Araraquara/SP, Brazil.

tecture for testing domain, discussing its establishment and use. Finally, our conclusions and future directions are presented.

2. Background

Over the last decades software architecture has received increasing attention as an important subfield of Software Engineering [19]. According to Shaw and Clemments [19], in the near future, software architectures will attain the status of all truly successful technologies. As already highlight in [22], software architectures play a major role in determining system quality — performance and reliability, for instance —, since they form the backbone for any successful software-intensive system. Software architecture is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [2]. In this context, reference architectures have been investigated as a mechanism that promotes reuse of design expertise and achieves well-recognized understanding of a specific domain. Reference architectures for diverse domains — e-commerce, for instance — can be found. Although, there is a lack of adequate reference architectures for software engineering domain, including software testing.

In order to design correctly and document clearly software architectures, ADLs (Architecture Description Languages)[1], as well as architecture evaluation methods such as ATAM (Architecture Tradeoff Analysis Method)[2], have been proposed. Documentation based on architectural views [9] and UML (Unified Modeling Language)[3] have been investigated as a promising approach to document architectures.

2.1 Related Works

Few works related to software architecture for testing domain have been conducted [5, 23]. Eickelmann [5] established the first reference architecture for this domain. This architecture establishes six functions of testing tools: test execution, test development, test failure analysis, test measurement, test management, and test planning. However, the relationship among these functions, as well as practical and detailed considerations were not established. Based on Eickelmann's work, Yang [23] established a more detailed architecture; however, the focus of this work is the testing of web applications. As a consequence, it is concentrated in how to structure the tools to test web applications. In spite of the Yang's work not proposing a reference architecture, it has contributed with the establishment of a set of concrete and abstract class, as well as operations of testing tools. Another work established an architecture for testing tools deployed in web platform [7]; although this work does not address the activities directly related to testing tools core.

Besides these works, we can find architectures of specific testing tools, such as [12, 14]. These architectures did not consider issues related to evolution, reusability, and maintainability. Also, these architectures do not support or foresee the support to all activities of the testing process, such as configuration management, planning, documentation, among others. When supported, the functionalities related to these activities are implemented in the scattered and tangled way in the core of the testing tools. Also, those works exclusively related to architecture of testing tools [5, 23] do not support all activities of the testing process.

Considering not only the relevance of software architectures, but also the lack of works related to architectures in the software testing domain, we have proposed a reference architecture for that domain, presented next.

3. Reference Architecture for Testing Domain

RefTEST is based on a more generic reference architecture, named RefASSET (Reference Architecture for Software Engineering Tools) [16] that supports the development of tools and SEEs (Software Engineering Environments). Thus, before to present RefTEST, we will briefly discuss its principles next.

3.1. Principles of RefTEST

In order to support adequately the development of tools and SEEs, it is first important to understand the relationship among the software engineering activities. We have adopted the concept of separation of concerns, one of the key principles in Software Engineering. According to Harrison [8], an adequate separation of concerns is pointed out as the main mechanism to build evolvable and reusable SEEs.

In the context of software development process, we have considered each software engineering activity as a concern [18]. In order to identify all concerns, we adopted the international standard ISO/IEC 12207 (Information Technology - Software Life Cycle Processes) [10] that provides a comprehensive set of life cycle processes, activities, and tasks for developing and maintaining software system. This standard establishes and groups the activities (concerns) into three categories — primary, organizational, and supporting — depending on the role they play in the software development. According to its characteristics, a concern can be either a primary concern — that refers to activities that perform the development, operation, and maintenance of software systems — or a crosscutting concern, term used by

[1] http://www.sei.cmu.edu/architecture/adl.html
[2] http://www.sei.cmu.edu/architecture/ata_method.html
[3] http://www.uml.org

the AOSD (Aspect-Oriented Software Development) community and referred to concerns that are spread throughout or tangled with others concerns. It is important to highlight that an inherent characteristic of the crosscutting concerns (i.e. the organizational and supporting concerns) is their occurrence along all primary concerns, as illustrated in Figure 1. For instance, documentation is one of the concerns that must be considered from requirements specification to maintenance.

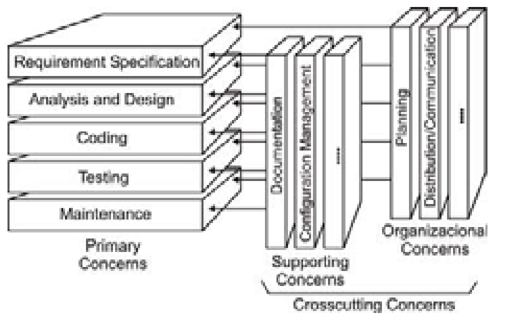


Figure 1. Relationship among Primary, Supporting and Organizational Concerns

The ideas discussed above were consolidated in RefASSET, illustrated in Figure 2. This architecture is not the focus of this work, but it is briefly described herein because it is the basis of RefTEST. In short, RefASSET is a reference architecture based on architectures already investigated, such as ECMA Reference Model [4] and architectures of interactive systems and Web systems (architectural pattern MVC (Model-View-Controller) and 3-tiers architecture[4]), considering the future perspective of Web as platform to provide diverse software systems, including software engineering tools.

It should be highlighted that the `Application Layer` in Figure 2 contains all identified concerns (`primary`, `supporting`, and `organizational`). In particular, the `primary concerns` part contains the core of the tools that implement the primary concerns.

RefASSET has been explored in order to facilitate the use and integration of tools, processes, and artifacts in SEEs. By considering separation of concerns in this architecture, we are pursuing easier integration, maintainability, quality, and reusability of the environments built based on this architecture.

3.2. Establishing RefTEST

RefASSET was specialized to software testing domain, resulting in RefTEST; thus, RefTEST inherited all RefASSET's characteristics, such as the separation of concerns and the architectures of interactive systems. In particular, the specialization concentrated in identifying the core of testing tools, i.e. the core concepts into `primary concerns` part. Three steps were conducted:

Step 1: Investigation of the Software Testing Knowledge

This step was conducted to identify the core activities performed during the testing. Knowledge into software testing domain was arisen, considering diverse information sources. Software testing processes, such as [20, 21], and the activities contained in these processes were investigated and identified as potential core activities. Software architectures proposed in the testing literature [5, 23] were also investigated and the knowledge (structure, modules, and their relationships) was considered. Besides that, as known, an ontology is a formal explicit specification of a shared conceptualization, providing a vocabulary for representing and communicating knowledge about some topic and a set of relationships which hold among the concepts in that vocabulary as well; thus, an ontology for software testing domain, named OntoTest [1], was basis to identify the concepts and relationships into testing domain. Moreover, the investigation of testing tools was the most important contribution for this step. The most known eight academic testing tools were considered, such as MuJava [12] and Jazz [14], and two broadly known and used commercial testing tools: those of the Mercury[5] and Rational[6]. Our experience in developing testing tools — for instance, published in [3] and [13] — was also important. As a result, 32 activities were identified, illustrated partially in Table 1. The information sources of each activity are also indicated in that table. For instance, activity "Generate test requirement" was identified through investigation of software testing processes (P), testing tools (T) and testing ontology (O). It is noticed that these activities refer to the core activities of testing; rather, the activities related to testers management and test planning, for instance, were set aside, since these are addressed by planning_management module, related to a crosscutting concern.

Step 2: Identification of the Core Concepts

Based on the identified activities, the core concepts were established. Each activity was related to a software testing concept. For instance, the activity "Include test cases" was related to the concept "test case", the activity "Generate test requirements" was related to "test requirement", and so on. It is important to highlight that only four concepts were identified and that they seem to be sufficient to represent the core elements of testing tools: test case, test requirement, test artifact, and test criteria. Figure 3 presents the conceptual model of the core of testing tools. Besides this

[4]http://www.sei.cmu.edu/str/descriptions/threetier.html

[5]http://www.mercury.com/us/products/quality-center/

[6]http://www-306.ibm.com/software/sw-bycategory/subcategory/SW730.html

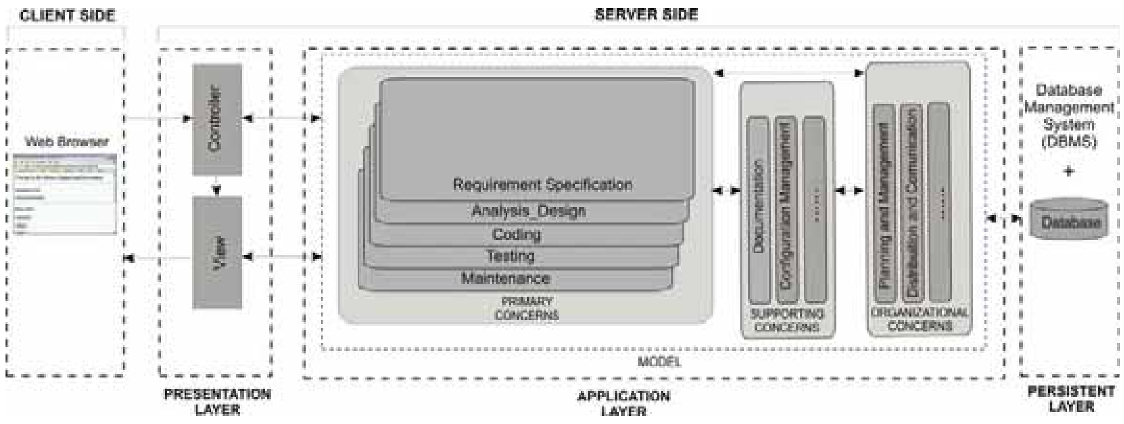


Figure 2. Reference Architecture of Software Engineering Environments

Table 1. Activities in the Testing Conduction

Core Activities	Source[a]
Deal with artifact to be tested	A, P, T, O
Execute artifact to be tested using test cases	A, P, T
Import test cases	P, T
Include test cases	P, T
Generate test requirements	P, T, O
Execute test requirements using test cases	P, T, O
Establish testing adequation criterion	A, O
Calculate test coverage	A, P, T
...	...

[a]P = testing processes; A = architectures; T = Testing Tools; O = OntoTest

model, definitions and related operations were established for each concept.

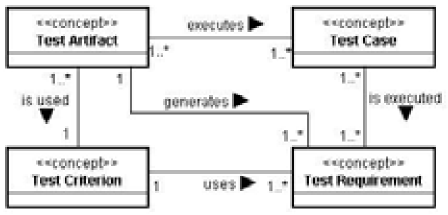

<<concept>> Test Artifact
executes
<<concept>> Test Case
1..*
1..*
is used
1
generates
is executed
<<concept>> Test Criterion
uses
<<concept>> Test Requirement

Figure 3. Core of Testing Tools

Step 3: Architectural Design

For the purpose of representing adequately the RefTEST, as recommended by [9], architectural views — module, runtime, deployment, among other views — were developed using UML 2.0. Also, extensions of the UML notation are required to support the aspects representation [17], since this architecture is based on aspects. For the sake of space, only the module and runtime views are presented herein.

The module view, in Figure 4, shows the structure of the software in terms of code units; packages and dependency relations are used to represent this view. While the core of testing tools (`testing_tool` package) can be implemented using classes, components, or subsystems, the modules that implement crosscutting concerns use aspects in their structures. For instance, the package `supporting_crosscutting_modules` contains modules that are implemented by classes and aspects. These aspects crosscut other modules and change the execution flow of these modules inserting functionalities related to a crosscutting concern. Thus, there are dependency relations, labelled with <<`crosscut`>>, among the package `supporting_crosscutting_modules` and other packages. It is important to highlight that we have explored the use of aspects to implement modules for SEE and used them as an integration and communication mechanism among modules in SEEs, including testing tools.

Differently from module view, the runtime view shows the structure of the system when it is executing. Thus, the interface among the modules is an important element that must be detached. In [17], the extensions required to UML to represent interfaces when aspects are used as a communication mechanism are discussed. In short, the Amodules (Aspect-based Modules) — i.e. modules that exclusively use aspects to communicate with other modules — have a filled circle attached by a solid line, representing their interface, named IMA (Interface Made by Aspects). This interface represents a declaration of a set of coherent features and obligations; in practice, it represents the characteristics — for example, objects and their operations — that modules must have to use the AModules. Also, it was proposed a half-square attached by a solid line to the module that is crosscuted or affected by aspects into AModules. Our experience points out that the establishment of the IMAs provides facilities to reuse AModules. Figure 5 presents the runtime view of RefTEST. In this figure, `Planning_management` and `Documentation` are AModules.

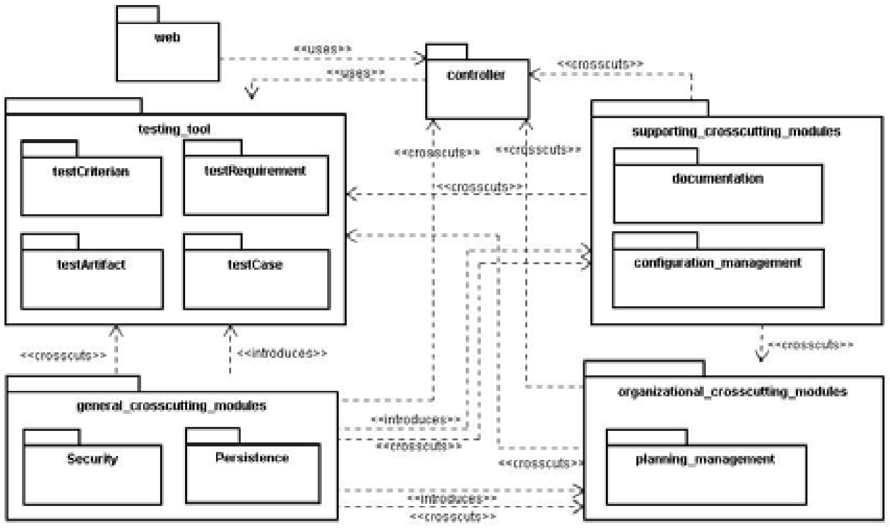


Figure 4. Module View of RefTEST

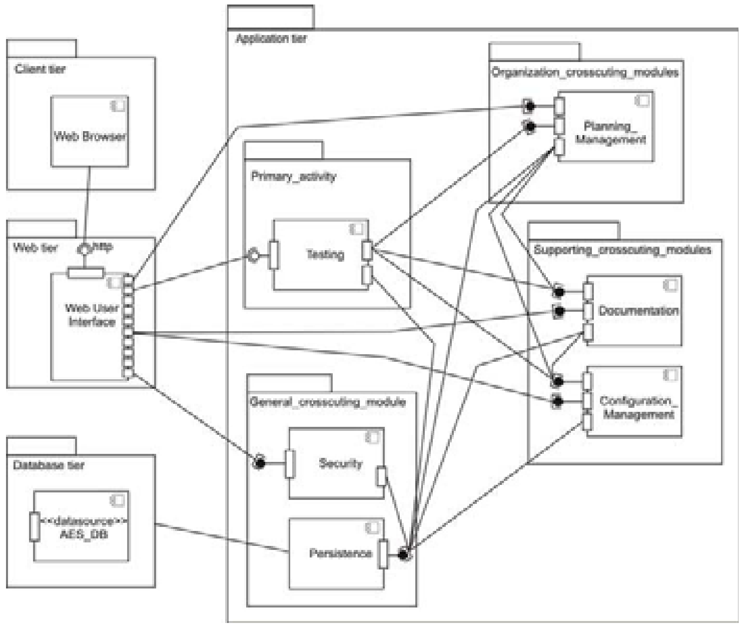


Figure 5. Runtime View of RefTEST

3.3. Use of RefTEST

We have used RefTEST to developing and reengineering of testing tools that are deployed in web platform [6]. Recent technologies to develop evolvable and reusable systems have been investigated and adopted; Java[7] and AspectJ[8] are in use. Also, we have developed modules that implement crosscutting concerns. For instance, a testing documentation module was implemented; the aspects collect information, such as the test cases and test requirement, during the tool runtime and format, update, and store these information. This module was designed and implemented in order to facilitate its evolution to other testing tools and tools of other domains, such as requirement specification and analysis/design, aiming at developing a documentation framework. In the end, we have pursuing a unique module that manages the documentation of all other software engineering activities that can be developed in an evolvable and

[7]http://java.sun.com/
[8]http://www.eclipse.org/aspectj/

incremental approach. Moreover, our experimental results have showed that the core of the tools can be independently developed of the modules that implement crosscutting concerns.

4. Conclusions

The systematization of the testing tools development is a real need of the testing research community. Considering the lack of recent works that establish reference architecture for testing domain, in this paper we proposed RefTEST, a reference architecture strongly based on separation of concerns, pursuing evolvability and reusability of the tools built based on this architecture. In special, this work contributed with the establishment of the testing tools core and also with a new point of view about how to address modules that implement crosscutting concerns. It is important to highlight that the use of aspects to develop and integrate these modules is another important contribution of this work. Our experience has pointed out that the development of testing tools using a well-established reference architecture is relevant. RefTEST have been an important guideline to minimize the efforts to the testing tools development.

References

[1] E. F. Barbosa, E. Y. Nakagawa, and J. C. Maldonado. Towards the establishment of an ontology of software testing. In *18th Int. Conf. on Soft. Engineering and Knowledge Engineering (SEKE'06)*, San Francisco Bay/USA, July 2006.

[2] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley Publishing Company, 2003.

[3] M. E. Delamaro, J. C. Maldonado, and A. P. Mathur. Interface mutation: An approach for integration testing. *IEEE Trans. on Software Engineering*, 27(3):228–247, Mar. 2001.

[4] ECMA and NIST. Reference model for frameworks of software engineering environments, Dec. 1991. Special Publication Report No. ECMA TR/55, 2nd Ed.

[5] N. S. Eickelmann and D. J. Richardson. An evaluation of software test environment architectures. In *18th Int. Conf. on Software Engineering (ICSE'96)*, pages 353–364, Berlin, Germany, 1996.

[6] F. C. Ferrari. *Supporting Structural and Mutation Testing of Object-Oriented and Aspect-Oriented Software*, 2006. Doctoral Work Plan, University of São Paulo, São Carlos, SP, Brazil. (in Portuguese).

[7] J. Gao, C. Chen, Y. Toyoshima, and D. Leung. Development an integrated testing environment using the World Wide Web technology. In *21st Inter. Computer Software and Applications Conference (COMPSAC'97)*, pages 594–601, Washington, USA, Aug. 1997.

[8] W. Harrison, H. Ossher, and P. Tarr. Software engineering tools and environments: a roadmap. In *13th Conf. on The Future of Software Engineering (ICSE'00)*, pages 261–277, New York, NY, USA, 2000.

[9] IEEE. 1471-2000 - Recommended practice for architectural description of software-intensive systems, Sept. 2000.

[10] ISO. ISO/IEC 12207. Information technology – software life-cycle processes, 1995.

[11] G. Kiczales, J. Irwin, J. Lamping, J. Loingtier, C. Lopes, C. Maeda, and A. Menhdhekar. Aspect-oriented programming. In M. Akşit and S. Matsuoka, editors, *Proc. of the European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, 1997.

[12] Y.-S. Ma, J. Offutt, and Y.-R. Kwon. Mujava: a mutation system for Java. In *ICSE'06 - 28th Int. Conf. on Software Engineering*, pages 827–830, New York, NY, 2006.

[13] J. C. Maldonado, M. E. Delamaro, S. C. P. F. Fabbri, A. S. Simão, T. Sugeta, A. M. R. Vincenzi, and P. C. Masiero. Proteum: A family of tools to support specification and program testing based on mutation. In *Mutation 2000 Symposium – Tool Session*, pages 113–116, San Jose, CA, Oct. 2000.

[14] J. Misurda, J. Clause, J. L. Reed, B. R. Childers, and M. L. Soffa. Demand-driven structural testing with dynamic instrumentation. In *27th Int. Conf. on Software Engineering (ICSE'05)*, St. Louis, Missouri, USA, May 2005.

[15] G. J. Myers, C. Sandler, T. Badgett, and T. M. Thomas. *The Art of Software Testing*. John Wiley & Sons, 2004.

[16] E. Y. Nakagawa. *A Contribution to the Architectural Design of Software Engineering Environments*. Phd thesis, University of São Paulo, São Carlos, SP, Brazil, Aug. 2006. (in Portuguese).

[17] E. Y. Nakagawa and J. C. Maldonado. Representing aspect-based architecture of software engineering environments. In *1st Workshop on Aspects in Architectural Description, 6th Int. Conf. on AOSD*, Vancouver, Canada, Mar. 2007.

[18] E. Y. Nakagawa, A. S. Simão, and J. C. Maldonado. Addressing separation of concerns in software engineering environments. In *IASTED Int. Conf. on Software Engineering*, Innsbruck, Austria, Feb. 2007.

[19] M. Shaw and P. Clements. The golden age of software architecture. *IEEE Software*, 23(2):31–39, Mar/Apr 2006.

[20] I. Sommerville. *Software Engineering*. Addison-Wesley Publishing Company, 6 edition, 2001.

[21] A. M. R. Vincenzi, M. E. Delamaro, A. S. Simão, and J. C. Maldonado. Muta-pro: Towards the definition of a mutation testing process. In *Proc. of the 6th IEEE Latin American Test Workshop*, Salvador, Bahia, Mar 2005.

[22] A. I. Wasserman. Toward a discipline of software engineering. *IEEE Software*, 13(6):23–31, Nov. 1996.

[23] J. Yang, J. Huang, F. Wang, and W. C. Chu. An object-oriented architecture supporting web application testing. In *23rd Annual Int. Computer Software and Applications Conference*, pages 122–127, Phoenix, Arizona, USA, Oct. 1999.

Controlling Restricted Random Testing: An Examination of the Exclusion Ratio Parameter

Kwok Ping Chan
Dept. of Comp. Sci.,
The Univ. of Hong Kong
Pokfulam Road,
Hong Kong, China
kpchan@cs.hku.hk

T. Y. Chen
Faculty of Information &
Communication Technologies,
Swinburne Univ. of Technology,
Hawthorn 3122, Australia
tychen@it.swin.edu.au

Dave Towey *
Division of Sci. & Tech.,
BNU–HKBU United International
College, Tangjiawan,
Zhuhai, 519085, China
davetowey@uic.edu.hk

Abstract

In Restricted Random Testing (RRT), the main control parameter is the Target Exclusion Ratio (R), the proportion of the input domain to be excluded from test case generation at each iteration. Empirical investigations have consistently indicated that best failure-finding performance is achieved when the value for the Target Exclusion Ratio is maximised, i.e. close to 100%. This paper explains an algorithm to calculate the Actual Exclusion Ratio for RRT*, and applies the algorithm to several simulations, confirming that previous empirically determined values for the Maximum Target Exclusion Ratio do give Actual Exclusion Ratios close to 100%. Previously observed trends of improvement in failure-finding efficiency of* RRT *corresponding to increases in Target Exclusion Ratios are also identified for Actual Exclusion Ratios.*

KEYWORDS: Software Testing; Random Testing; Adaptive Random Testing; Restricted Random Testing; Exclusion Ratio.

1. Introduction

Random Testing incorporating additional mechanisms to ensure more widespread distributions of test cases over an input domain have been called Adaptive Random Testing (*ART*) [2, 3, 4, 5, 6, 7]. It is considered a promising direction of automatic test case generation [8].

*Corresponding author

A version of *ART*, based on the use of exclusion, is the Restricted Random Testing (*RRT*) method [2]. By excluding regions surrounding previously executed test cases, and restricting subsequent cases to be drawn from other areas of the input domain, *RRT* ensures an even distribution, and guarantees a minimum distance amongst all cases. In experiments, the *RRT* method has outperformed *RT* by up to 80% on some occassions.

It has been observed that the failure-finding efficiency of *RRT* improved as the Target Exclusion Ratio (*R*) was increased, with the best failure-finding efficiency achieved when *R* was at a maximum [2]. The *Max R* refers to the maximum value for *R* beyond which the Actual Exclusion Ratio is too close to 100% for test cases to be generated.

The difference between Target and Actual exclusion is due to (1) *Overlapping* (*Olp*) of exclusion regions; and (2) portions of the exclusion regions falling *Out the Input Domain* (*OID*). Because of the importance of the *Max R*, the ability to accurately determine the Actual Exclusion Ratio for a given Target Exclusion Ratio was desirable.

In this paper, we explain an algorithm to calculate the Actual Exclusion Ratio and give the results of an application of the algorithm to estimate the expected Actual Exclusion Ratio for a given Target Exclusion Ratio. These results confirm that the

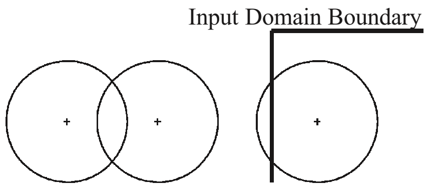


Figure 1. (a) Example of *Overlapping* (*Olp*) of exclusion regions (b) Example of portion of an exclusion region falling *Outside the Input Domain* (*OID*)

Actual Exclusion for Target Exclusion Ratio values near *Max R* is close to 100%. They also show that the Actual Exclusion does increase as Target Exclusion increases.

2. Maximum Target Exclusion

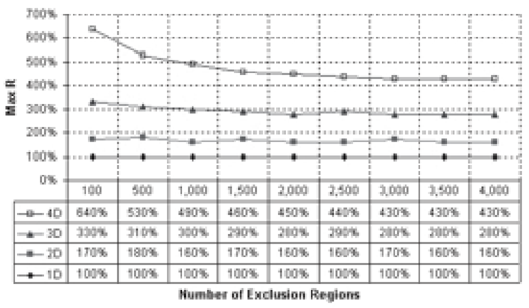
In *RRT*, given a test case that has not revealed failure, the area of the input domain from which subsequent test cases may be drawn is restricted. By employing a hyperspherical zone, a minimum distance (the radius of the exclusion zone) between all test cases is ensured.

All exclusion zones are of equal size, and this size decreases with successive test case executions. The size of each zone is related to both the size of the entire input domain, and the number of previously executed test cases.

The final (and most important) determinant of exclusion zone size in *RRT* is the Exclusion Ratio (*R*). This figure is applied to the total area of the input domain to obtain the total exclusion area.

During the execution of the *RRT* algorithm, the Actual Exclusion Ratio is usually less than the Target Ratio. This occurs when there is *Overlapping* (*Olp*) of exclusion regions (Fig. 1(a)); or when some portion of an exclusion region falls outside of the Input Domain, (*OID*) (Fig. 1(b)); or when some combination of both these situations occurs.

We defined the maximum target exclusion ratio (*Max R*) as the highest *R* at which it is still possible to generate test cases for a full sample size, *n*. Because of the difficulty of an analytical investigation of *Max R*, simulations were run to investigate what factors influenced it. In the simulations, the number of regions was varied from 100 to 4,000, the sample size (*n*) was set at 100, and *R* was incremented by 10% each time. A limit of 100,000 on the number of attempts to generate a valid test case was imposed for each test case. The input domain shape was homogeneous (square in 2D, cube in 3D, and hypercube in 4D). The results are summarized in Fig. 2. They indicate that, when the number of exclusion regions is lower, the maximum target exclusion (*Max R*) is higher.



	100	500	1,000	1,500	2,000	2,500	3,000	3,500	4,000
4D	640%	530%	490%	460%	450%	440%	430%	430%	430%
3D	330%	310%	300%	290%	280%	290%	280%	280%	280%
2D	170%	180%	160%	170%	160%	160%	170%	160%	160%
1D	100%	100%	100%	100%	100%	100%	100%	100%	100%

Figure 2. Results of Maximum Target Exclusion Rate (*Max R*) calculation for a homogeneous input domain. Sample size (n) is 100, and number of exclusion regions varies from 100 to 4,000

Because of the importance of the relation between Actual and Target Exclusion, an algorithm to calculate the Actual Exclusion Ratio was developed. As the best failure-finding performance was obtained when circular regions were used [1], our algorithm calculates Actual Exclusion for circular exclusion regions.

3. Actual Exclusion Ratio Calculation

In this section, the algorithm for calculating the Actual Exclusion Ratio is explained. The method examines the distribution of test cases and their exclusion regions, and calculates the loss of exclusion region area caused by *Overlapping* (*Olp*) of regions, and by portions of regions falling *Outside the Input Domain* (*OID*).

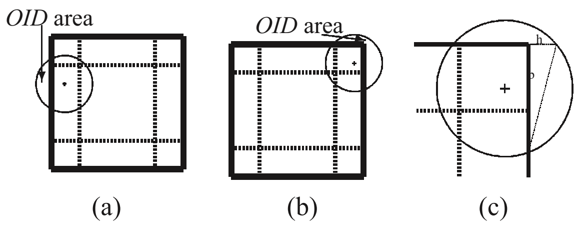


Figure 3. (a) *OID* when TC is in a non-corner region of the *gutter* (b) *OID* when TC is in a corner region (c) Partitioning of *OID* for area calculation

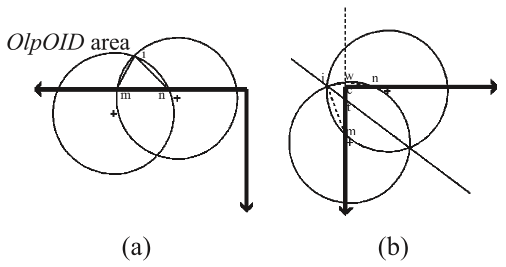


Figure 4. (a) Example of *OlpOID* and Input Domain on only one border (b) Example of *OlpOID* and Input Domain on two borders

3.1. Actual Exclusion Ratio Calculation Algorithm

The Actual Exclusion will differ from the Target Exclusion when any combination of the following occurs: (a) *OID* — occur when any previously executed test case (TC_x) lies within a distance r of the input domain border (this area of depth r inside the border is referred to as the *gutter*). (b) *Olp* — occur when any previously executed test cases (TC_x, TC_y) are within twice the exclusion zone radius of each other.

To calculate the Actual Exclusion Ratio, we first find the Target Exclusion Area, and then subtract the total area of *OID*, and then subtract the area of *Olp* inside the Input Domain.

3.2. Calculation of OID

First, the location of each previously executed test case is examined to determine whether it will have any portion of its exclusion zone lying outside the Input Domain. This occurs when the TC is within distance r (exclusion radius) of the Input Domain border. There are two cases.

The *OID* area for the TCs not lying in the corners of the *gutter* can be straightforwardly calculated from the area of the *circle segment* formed when the Input Domain border, acting as the *secant*, cuts the exclusion region circle.

Calculation of the *OID* area for the TCs which are in the corners of the *gutter* requires partitioning the *OID* region into three smaller regions, as shown in Fig. 3(b). Fig. 3(c) shows how this area is partitioned into a triangle and two *circle segments*.

3.3. Calculation of Olp

First, every pair of executed test cases is examined to see if there is any overlap between their exclusion regions. The overlap is calculated by bisecting the region with the line through the intersections of the circles, creating 2 identical *circle segments*.

Next, the location of the circles' intersection points (ip) is checked. If an intersection point is outside the Input Domain, some of the *Olp* area will also lie outside, and must be subtracted from the total *Olp* loss. The portion of *Olp* lying Outside the Input Domain is referred to as *OlpOID*.

For an intersection point (ip) outside the Input Domain, the area of the Overlap lying Outside the Input Domain (*OlpOID*) is calculated in one of two ways, according to whether the *OlpOID* area touches the Input Domain on one or two borders.

3.4. Calculation of OlpOID

When the *OlpOID* is on only one border (Fig. 4(a)), the area can be calculated by totaling the areas of the triangle formed by the intersection of the circles (i in Fig. 4(a)) and the intersections of the *arcs* from i to the border (m and n), and the two *circle segments* formed by the *chords* from the intersection of the circles to the intersections of the *arcs* and the border.

When the *OlpOID* is on two borders, the region is split into three triangles and three *circle segments*, as shown in Fig. 4(b). The area of these regions can be easily calculated.

Table 1. Target vs Actual exclusion ratios for *RRT*.

Target Excl. Ratio	**Actual Exclusion (%)**							
	Total Number of Exclusion Zones							
	10	**50**	**100**	**150**	**500**	**1000**	**5000**	**10000**
1%	0.98	0.99	0.99	0.99	1.00	1.00	1.00	1.00
20%	18.21	18.87	19.05	19.12	19.28	19.34	19.41	19.43
40%	34.15	36.03	36.51	36.72	37.14	37.29	37.50	37.55
60%	48.05	51.33	52.15	52.55	53.26	53.52	53.89	53.98
80%	60.22	64.62	65.69	66.17	67.20	67.59	68.09	68.21
100%	70.16	75.48	76.76	77.33	78.52	78.95	79.50	79.64
120%	78.29	83.69	84.97	85.51	86.61	86.97	87.47	87.58
140%	84.60	89.12	90.07	90.42	90.98	91.15	91.33	91.36
160%	88.72	91.68	91.88	91.75	91.37	91.12	90.70	90.57

4. Actual Exclusion Ratio

The algorithms described were applied to several simulations, varying the Target Exclusion Ratio, and the total number of executed test cases. The simulations were conducted within a square input domain.

Table 1 shows the results for different numbers of exclusion regions (10 to 10,000), averaged over 1,000 trials. In the table, Target Exclusion is the percentage area of the Input Domain which we attempt to exclude from random point generation, this was varied from 1% to 160%. Actual Exclusion is the average percentage of the Input Domain which is actually excluded by the exclusion zones.

As expected, there is a difference between the Target and Actual Exclusion, and this difference appears to become more pronounced as the number of test cases increases, and also as the Target Exclusion Ratio increases. It also confirms that the Actual Exclusion ratios increase with *R*, i.e., the improvement in failure-finding efficiency does correspond to increases in the Actual Exclusion.

By nature, *Max R* may vary with the sample size. Table 1 shows that around the *Max R* values, the Actual Exclusion Ratio is very close to 100%. However, with high Actual Exclusion, the number of attempts necessary to generate a test case outside the exclusion regions increases: e.g. 99% exclusion leaves only 1% of the input domain outside the exclusion regions, which would take an average of 100 (1/1%) trials to find.

5. Summary

In this paper, we presented an algorithm for calculating the Actual Exclusion Ratio, in 2D, for the *RRT* method. We also presented the results of simulations for various numbers of test cases, and varying Target Exclusion Ratios. These results confirmed that the Actual Exclusion does increase as Target Exclusion increases. Hence, the improvements for failure-finding efficiency are linked to increases in Actual Exclusion. Also, the Actual Exclusion Ratio for Target Exclusion Ratio near *Max R* is indeed close to 100%. The results provide stronger theoretical support for *RRT*.

Acknowledgement

This project is partially supported by an Australian Research Council Discovery Grant (DP0557246).

References

[1] K. P. Chan, T. Y. Chen, and D. Towey, "Adaptive Random Testing with Filtering: An Overhead Reduction Technique", *17th Int. Conf. on Software Engg. and Knowledge Engg. (SEKE'05)*, Taipei, Taiwan, July 14-16, 2005.

[2] K. P. Chan, T. Y. Chen, and D. Towey, "Restricted Random Testing: Adaptive Random Testing by Exclusion", *Int. J. of Software Engg. and Knowledge Engg.*, Vol. 16, No. 4, pp. 553–584, 2006.

[3] T. Y. Chen, F. C. Kuo, R. G. Merkel, S. P. Ng, "Mirror Adaptive Random Testing", *Inf. and Software Tech.*, Vol. 46, No. 15, pp. 1001–1010, 2004.

[4] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive Random Testing", *Lecture Notes in Comp. Sci.*, Vol. 3321, pp. 320–329, Jan 2004.

[5] J. Mayer, "Adaptive Random Testing by Bisection and Localization", *5th Int. Workshop on Formal Approaches to Testing of Software (FATES 2005)*, LNCS 3997, Springer-Verlag, pp. 72–86, 2006.

[6] J. Mayer, "Adaptive Random Testing by Bisection with Restriction", *Formal Methods and Software Engg, 7th Int. Conf. on Formal Engg. Methods (ICFEM 2005)*, LNCS 3785, Springer-Verlag, pp. 251–263, 2005.

[7] J. Mayer, "Lattice-Based Adaptive Random Testing", *20th IEEE/ACM Int. Conf. on Automated Software Engg.*, pp. 333–336, 2005.

[8] I. Ciupa, A. Leitner, M. Oriol, and B. Meyer, "Object Distance And Its Application To Adaptive Random Testing Of Object-Oriented Programs", *1st International Workshop on Random Testing (ISSTA-RT 2006)*, pp. 55-63, 2006.

An Approach to Software Testing of Machine Learning Applications

Christian Murphy
Dept. of Computer Science
Columbia University
New York, NY
cmurphy@cs.columbia.edu

Gail Kaiser
Dept. of Computer Science
Columbia University
New York, NY
kaiser@cs.columbia.edu

Marta Arias
Center for Computational Learning Systems
Columbia University
New York, NY
marta@ccls.columbia.edu

Abstract

Some machine learning applications are intended to learn properties of data sets where the correct answers are not already known to human users. It is challenging to test such ML software, because there is no reliable test oracle. We describe a software testing approach aimed at addressing this problem. We present our findings from testing implementations of two different ML ranking algorithms: Support Vector Machines and MartiRank.

1. Introduction

We investigate the problem of making machine learning (ML) applications dependable, focusing on software testing. Conventional software engineering processes and tools do not neatly apply: in particular, it is challenging to detect subtle errors, faults, defects or anomalies (henceforth "bugs") in the ML applications of interest because there is no reliable "test oracle" to indicate what the correct output should be for arbitrary input. The general class of software systems with no reliable test oracle available is sometimes known as "non-testable programs" [1]. These ML applications fall into a category of software that Davis and Weyuker describe as *"Programs which were written in order to determine the answer in the first place. There would be no need to write such programs, if the correct answer were known"* [2]. Formal proofs of an ML algorithm's optimal quality do not guarantee that an application implements or uses the algorithm correctly, and thus software testing is needed. Our testing, then, does not seek to determine whether an ML algorithm learns well, but rather to ensure that an application using the algorithm correctly implements the specification and fulfills the users' expectations.

In this paper, we describe our approach to testing ML applications, in particular those that implement ranking algorithms (a requirement of the real-world problem domain). Of course, in any software testing, it is possible only to show the presence of bugs but not their absence. Usually when input or output equivalence classes are applied to developing test cases, however, the expected output for a given input is known in advance. Our research seeks to address the issue of how to devise test cases that are likely to reveal bugs, and how one can indeed know whether a test actually is revealing a bug, given that we do not know what the output should be in the general case.

Our approach for creating test cases consists of three facets: analyzing the problem domain and the corresponding real-world data sets; analyzing the algorithm as it is defined; and analyzing the implementation's runtime options. While this approach is conventional, not novel, a number of issues arise when applying it to determining equivalence classes and generating data sets for testing ML ranking code.

We present our findings to date from two case studies: our first concerns the Martingale Boosting algorithm, which was developed by Long and Servedio [3] initially as a classification algorithm and then adapted by Long and others into a ranking algorithm called MartiRank [4]; we then generalized the approach and applied it to an implementation of Support Vector Machines (SVM) [5] called SVM-Light [6].

2. Background

We are concerned with the development of an ML application commissioned by a company for potential future experimental use in predicting impending device failures, using historic data of past device failures as well as static and dynamic information about the current devices. Classification in the binary sense ("will fail" vs. "will not fail") is not sufficient because, after enough time, every device will eventually fail. Instead, a ranking of the propensity of failure with respect to all other devices is more appropriate. The prototype application uses both the MartiRank and SVM algorithms to produce rankings; the dependability of the implementations has real-world implications, rather than just academic interest. We do not discuss the full application further in this paper; see [4] for details.

2.1. Machine learning fundamentals

In general, there are two phases to supervised machine learning. The first phase (called the *learning* phase) analyzes a set of *training* data, which consists of a number of *examples*, each of which has a number of *attribute* values and one *label*. The result of this analysis is a *model* that attempts to make generalizations about how the attributes relate to the label. In the second phase, the model is applied to another, previously-unseen data set (the *testing* data) where the labels are unknown. In a *classification* algorithm, the system attempts to predict the label of each individual example; in a *ranking* algorithm, the output of this phase is a ranking such that, when the labels become known, it is intended that the highest valued labels are at or near the top of the ranking, with the lowest valued labels at or near the bottom.

One complication in this effort arose due to conflicting technical nomenclature: "testing", "regression", "validation", "model" and other relevant terms have very different meanings to machine learning experts than they do to software engineers. Here we employ the terms "testing" and "regression testing" as appropriate for a software engineering audience, but we adopt the machine learning sense of "model" (i.e., the rules generated during training on a set of examples) and "validation" (measuring the accuracy achieved when using the model to rank the training data set with labels removed, rather than a new data set).

2.2. MartiRank and SVM

MartiRank [4] was specifically designed as a ranking algorithm with the device failure application in mind. In the learning phase, MartiRank executes a number of "rounds". In each round the set of training data is broken into sub-lists; there are N sub-lists in the N^{th} round, each containing $1/N^{th}$ of the total number of device failures. For each sub-list, MartiRank sorts that segment by each attribute, ascending and descending, and chooses the attribute that gives the best "quality". The quality of an attribute is assessed using a variant of the Area Under the Curve (AUC) [7] that is adapted to ranking rather than binary classification. The model, then, describes for each round how to split the data set and on which attribute and direction to sort each segment for that round. In the second phase, MartiRank applies the segmentation and the sorting rules from the model to the testing data set to produce the ranking (the final sorted order).

SVM [5] belongs to the "linear classifier" family of ML algorithms that attempt to find a (linear) hyperplane that separates examples from different classes. In the learning phase, SVM treats each example from the training data as a vector of K dimensions (since it has K attributes), and attempts to segregate the examples with a hyperplane of K-1 dimensions. The type of hyperplane is determined by the SVM's "kernel": here, we investigate the linear, polynomial, and radial basis kernels. The goal is to find the maximum margin (distance) between the "support vectors", which are the examples that lie closest to the surface of the hyperplane; the resulting hyperplane is the model. As SVM is typically used for binary classification, ranking is done by classifying each individual example (irrespective of the others) from the testing data according to the model, and then recording its distance from the hyperplane. The examples are then ranked according to this distance.

2.3. Related work

Although there has been much work that applies machine learning techniques to software engineering in general and software testing in particular (e.g., [8]), there seems to be very little published work in the reverse sense: applying software testing techniques to ML software. There has been much research into the creation of test suites for regression testing [9] and generation of test data sets [10, 11], but not applied to ML code. Repositories of "reusable" data sets have been collected (e.g., the UCI Machine Learning Repository [12]) for the purpose of comparing result quality, but not for the software engineering sense of testing. Orange [13] and Weka [14] are two of several frameworks that aid ML developers, but the testing functionality they provide is focused on comparing the quality of the results, not evaluating the "correctness" of the implementations.

3. Software Testing Approach

3.1. Analyzing the problem domain

The first part of our approach is to consider the problem domain and try to determine equivalence classes based on the properties of real-world data sets.

We particularly look for traits that may not have been considered by the algorithm designers, such as data set size, the potential ranges of attribute and label values, and what sort of precision is expected when dealing with floating point numbers.

The data sets of interest are very large, both in terms of the number of attributes (hundreds) and the number of examples (tens of thousands). The label could be any non-negative integer, although it was typically a 0 (indicating that there was no device failure) or 1 (indicating that there was), and rarely was higher than 5 (indicating five failures over a given period of time).

The attribute values were either numerical or categorical. Many non-categorical attributes had repeated values and many values were missing, raising the issues of breaking "ties" during sorting and handling unknowns. We do not discuss categorical attributes further (because we found no relevant bugs).

3.2. Analyzing the algorithm as defined

The second element to our approach to creating test cases was to look at the algorithm as it is defined (in pseudocode, for instance) and inspect it carefully for imprecisions, particularly given what we knew about the real-world data sets as well as plausible "synthetic" data sets. This would allow us to speculate on areas in which flaws might be found, so that we could create test sets to try to reveal those flaws. Here, we are looking for bugs in the specification, not so much bugs in the implementation. For instance, the algorithm may not explicitly explain how to handle missing attribute values or labels, negative attribute values or labels, etc.

Also, by inspecting the algorithm carefully, one can determine how to construct "predictable" training and testing data sets that should (if the implementation follows the algorithm correctly) yield a "perfect" ranking. This is the only area of our work in which we can say that there is a "correct" output that should be produced by the ML algorithm.

For instance, we know that SVM seeks to separate the examples into categories. In the simplest case, we could have labels of only 1s and 0s, and then construct a data set such that, for example, every example with a given attribute equal to a specific value has a label of 1, and every example with that attribute equal to any other value has a label of 0. Another approach would be to have a set or a region of attribute values mapped to a label of 1, for instance "anything with the attribute set to X, Y or Z" or "anything with the attribute between A and B" or "anything with the attribute above M". We could also create data sets that are predictable but have noise in them to try to confuse the algorithm.

Generating predictable data sets for MartiRank is a bit more complicated because of the sorting and segmentation. We created each predictable data set by setting values in such a way that the algorithm should choose a specific attribute on which to sort for each segment for each round, and then divided the distribution of labels such that the data set will be segmented as we would expect; this should generate a model that, when applied to another data set showing the same characteristics, would yield a perfect ranking.

3.3. Analyzing the runtime options

The last part of our approach to generating test cases for ML algorithms is to look at their runtime options and see if those give any indication of how the implementation may actually manipulate the input data, and try to design data sets and tests that might reveal flaws or inconsistencies in that manipulation.

For example, the MartiRank implementation that we analyzed by default randomly permutes the order of the examples in the input data so that it would not be subject to the order in which the data happened to be constructed; it was, however, possible to turn this permutation off with a command-line option. We realized, though, that in the case where none of the attribute values are repeating, the input order should not matter at all because all sorting would necessarily be deterministic. So we created test cases that randomly permuted such a data set; regardless of the input order, we should see the same final ranking each time.

SVM-Light has numerous runtime options that deal with optimization parameters and variables used by the different kernels for generating the hyperplane(s). To date we have only performed software testing with the default options, although we did test with three of the different kernels: linear, polynomial, and radial basis.

4. Findings

To facilitate our testing, we created a set of utilities targeted at the ML algorithms we investigated. The utilities currently include: a data set generator; tools to compare a pair of output models and rankings; several trace options inserted into the ML implementations; and tools to help analyze the intermediate results indicated by the traces.

Using our testing approach, we devised the following basic equivalence classes: small vs. large data sets; repeating vs. non-repeating attribute values; missing vs. non-missing attribute values; repeating vs. non-repeating labels; negative labels vs. non-negative-only labels; predictable vs. non-predictable data sets;

and combinations thereof. These equivalence classes were then used to parameterize the test case selection criteria applied by our data generator tool to automate creation of appropriate input data sets.

We first applied our approach to creating the selective test cases for execution by MartiRank. We then generalized the approach and applied it to SVM-Light. Here we describe our most important findings.

4.1. Testing MartiRank

The MartiRank implementation did not have any difficulty handling large numbers of examples, but for larger than expected numbers of attributes it reproducibly failed (crashed). Analyzing the tracing output and then inspecting the code, we found that some code that was only required for one of the runtime options was still being called even when that flag was turned off – but the internal state was inappropriate for that execution path. We refactored the code and the failures disappeared.

Our approach to creating test cases based on analysis of the pseudocode led us to notice that the MartiRank algorithm does not explicitly address how to handle negative labels. Because the particular implementation we were testing was designed specifically to predict device failures, which would never have a negative number as a label, this was not considered during development. However, the implementation did not complain about negative labels but produced obviously incorrect results when a negative label existed. In principle a general-purpose ranking algorithm should allow for negative labels (-1 vs. +1 is sometimes used in other applications).

Also, by inspecting the algorithm and considering any potential vagueness, we developed test cases that showed that different interpretations could lead to different results. Specifically, because MartiRank is based on sorting, we questioned what would happen in the case of repeating values; in particular, we were interested to see whether "stable" sorting was used, so that the original order would be maintained in the case of ties. We constructed data sets such that, if a stable sort were used, a perfect ranking would be achieved because examples with the same value for a particular attribute would be left in their original order; however, if the sort were not stable, then the ranking would not necessarily be perfect because the examples could be out of order. Our testing showed that the sorting routine was not, in fact, stable. Though this was not specified in the algorithm, the developers agreed that it would be preferable to have a stable sort for deterministic results – so we substituted another, "stable" sorting routine.

4.2. Regression testing

A desirable side effect of our testing has been to create a suite of data sets that can then be used for regression testing purposes. Development of the MartiRank implementation is ongoing, and our data sets have been used successfully to find newly-introduced bugs. For example, after a developer refactored some repeated code and put it into a new subroutine, regression testing showed that the resulting models were different than for the previous version. Inspection of the code revealed that a global variable was incorrectly being overwritten, and after the bug was fixed, regression testing showed that the same results were once again being generated.

4.3. Testing multiple implementations

Davis and Weyuker suggest a "pseudo-oracle" as the solution to testing non-testable programs, i.e. constructing a second implementation and comparing the results of the two implementations on the same inputs [2]. Should multiple implementations of an algorithm happen to exist, our approach could be used to create test cases for such comparison testing. If they are *not* producing the same results, then presumably one or both implementations has a bug.

There are conveniently multiple implementations of the MartiRank algorithm: the original written in Perl and then a faster version written in C (most of the above discussion is with respect to the C implementation, except the bug mentioned for regression testing was in the Perl version). Using one as a "pseudo-oracle" for the other, we noticed a difference in the rankings they produced during testing with the equivalence class of missing attribute values. Using traces to see how the examples were being ordered during each sorting round, we noticed that the presence of missing values was causing the known values to be sorted incorrectly by the Perl version. This was due to using a Perl starship comparison operator that assumed transitivity among comparisons even when one of the values in the comparisons was missing, which is incorrect.

4.4. Generalization to SVM-Light

After completing the testing of MartiRank, we then generalized the approach and applied it to SVM-Light.

We did not uncover any issues with respect to most of the test cases involving unexpected values (such as negative labels or missing attributes) or repeating attribute values. However, with the linear and polynomial kernels, permuting the training data caused SVM-Light to create different models for different input orders. This occurred even when all attributes and labels were distinct – thus removing the possibility that ties between equal or missing values would be broken depending on the input order. We confirmed that these models were not "equivalent" by using the same testing data with each pair of such different models, and indeed obtained two different rankings. The practical implication is that the order in which the training data happens to be assembled can have an effect on the final ranking. This did not happen for the radial basis kernel in any of our tests to date.

Our analysis of the SVM algorithm indicates that it theoretically should produce the same model regardless of the input data order; however, an ML researcher familiar with SVM-Light told us that because it is inefficient to run the quadratic optimization algorithm on the full data set all at once, the implementation performs "chunking" whereby the optimization algorithm runs on subsets of the data and then merges the results [15]. Numerical methods and heuristics are used to quickly converge to the optimum. However, the optimum is not necessarily achieved, but instead this process stops after some threshold of improvement. This is one important area in which the implementation deviates from the specification.

Our other key findings came from those test cases involving "predictable" rankings. We created a small data set by hand that should yield a perfect ranking in SVM: for the first attribute, every example that had a value less than X (where X is some integer) had a label of one; everything else had a label of zero. There were two other columns of random noise. All three kernels correctly ranked the examples. In another test, however, we changed the labels so that they were all different – simply equal to the value of that row's first attribute incremented by 1. The linear and radial basis kernels found the perfect ranking but the polynomial kernel did not. We assumed that this was because of the noise, so we removed the noise and it indeed found the perfect ranking. This was the only case in our testing in which noise in the data set caused SVM-Light to fail to find the perfect ranking.

In other test cases with predictable rankings, we noticed that different kernels exhibited different behaviors with respect to how they performed on different types of data sets. For example, the linear and polynomial kernels could find the perfect rankings when a particular attribute had a *range* of values that correlated to a label of 1, but the radial basis kernel only found the perfect rankings when an attribute had a *single* value that correlated. This difference is, after all, the motivation for multiple kernels, but from our perspective it shows that what is predictable for one kernel is not always predictable for another.

Finally, although there are multiple implementations of the SVM algorithm, our testing did not include comparison testing (using one as a "pseudo-oracle" for another). We leave this as future work.

5. Discussion

Our approach was successful in that it helped us discover bugs in the implementations and discrepancies from the stated algorithms. By inspecting the algorithms, we could create predictable data sets that should yield perfect rankings and indicate whether the algorithm was implemented correctly; we could also see where the imprecisions were, especially in the case of MartiRank with respect to sorting missing or repeated values. Lastly, by considering the runtime options, we conceived test cases that permuted the input data, revealing an inconsistency in SVM-Light.

Possibly the most important thing we discovered is that what is "predictable" for one algorithm will not necessarily lead to a perfect ranking in another. For instance, in cases when the examples with a 1 label have a particular attribute whose value is in the middle of a range, it is hard for MartiRank to get that example towards the top of the ranking, though this is possible for most SVM kernels.

Also, as noted, although MartiRank is based on sorting, it does not specify whether the sorting of attributes should use a "stable" sort, so we found problems with how repeated or missing attribute values were handled. We also noticed that the algorithm states that each partition should have the same number of failures, but it does not address how many *non-failures* should appear in each partition, i.e. whether the dividing point is above or below those non-failures, or how the failures should be split amongst partitions when it is impossible to do so evenly.

We also discovered that tracing of intermediate state can be useful, because even though we may not know what the final output should be, inspection of the algorithm could indicate what to expect from the intermediate results. In the case of MartiRank, we could inspect the rankings at the end of each round and see how the examples were being sorted; this led us to discover the unstable sorting problem.

Although our testing to date has focused only on two ML algorithms, by developing the testing approach for MartiRank and then applying it to SVM-Light, we have shown that our approach and even specific test cases can be generalized to other ML ranking algorithms, which are likely to require many of the same equivalence classes discussed here. The general approach also seems appropriate to software testing of the implementations of supervised ML classification algorithms. The primary difference is that classification algorithms seek to categorize each single example, not rank-order a group of them, but investigating the problem domain and considering the algorithm as defined as well as the code's runtime options (if any) should still apply.

6. Limitations and Future Work

We have not yet addressed the issue of test suite adequacy, e.g. to extend our data generation tool to automatically generate sets of test cases that collectively cover all statements, branches or paths. Further, mutation analysis could be used for evaluating and improving the effectiveness of a given test suite. We leave these as future directions.

Other future work could include the investigation of automatically generating data sets that exhibit the same correlations among attributes and between attributes and labels as do real-world data, such as in [16]. Additionally, since some ML algorithms are intentionally non-deterministic and necessarily rely on randomization, more detailed trace analysis techniques should be investigated towards determining software implementation correctness.

7. Acknowledgements

Numerous people contributed to this effort. We would particularly like to thank David Waltz, Wei Chu, John Gallagher, Philip Gross, Bert Huang, Phil Long and Rocco Servedio for their assistance and encouragement. Murphy and Kaiser are members of the Programming Systems Lab, funded in part by NSF CNS-0627473, CNS-0426623 and EIA-0202063, NIH 1 U54 CA121852-01A1, and are also affiliated with the Center for Computational Learning Systems (CCLS). Arias is fully supported by CCLS, with funding in part by Consolidated Edison Company of New York.

8. References

[1] E.J. Weyuker, "On Testing Non-Testable Programs", *Computer Journal vol.25 no.4*, November 1982, pp.465-470.

[2] M.D. Davis and E.J. Weyuker, "Pseudo-Oracles for Non-Testable Programs", *Proceedings of the ACM '81 Conference*, 1981, pp. 254-257.

[3] P. Long and R. Servedio, "Martingale Boosting", *Eighteenth Annual Conference on Computational Learning Theory (COLT)*, Bertinoro, Italy, 2005, pp. 79-94.

[4] P. Gross *et al.*,"Predicting Electricity Distribution Feeder Failures Using Machine Learning Susceptibility Analysis", *Proceedings of the Eighteenth Conference on Innovative Applications in Artificial Intelligence*, Boston MA, 2006.

[5] V.N. Vapnik, *The Nature of Statistical Learning Theory*. Springer, 1995.

[6] T. Joachims, *Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf and C. Burges and A. Smola (ed.), MIT-Press, 1999.

[7] J.A. Hanley and B. J. McNeil, "The meaning and use of the area under a receiver operating characteristic (ROC) curve", *Radiology vol.143*, 1982, pp. 29-36.

[8] T.J. Cheatham, J.P. Yoo and N.J. Wahl, "Software testing: a machine learning experiment", *Proceedings of the 1995 ACM 23rd Annual Conference on Computer Science*, Nashville TN, 1995, pp. 135-141.

[9] G. Rothermel *et al.*, "On Test Suite Composition and Cost-Effective Regression Testing", *ACM Transactions on Software Engineering and Methodology, vol.13, no.3*, July 2004, pp 277-331.

[10] B. Korel, "Automated Software Test Data Generation", *IEEE Transactions on Software Engineering vol.16 no.8*, August 1990, pp.870-879.

[11] C.C. Michael, G. McGraw and M.A. Schatz, "Generating Software Test Data by Evolution", *IEEE Transactions on Software Engineering, vol.27 no.12*, December 2001, pp.1085-1110.

[12] D.J. Newman, S. Hettich, C.L. Blake and C.J. Merz, UCI *Repository of machine learning databases*, University of California, Department of Information and Computer Science, Irvine CA, 1998.

[13] J. Demsar, B. Zupan and G. Leban, *Orange: From Experimental Machine Learning to Interactive Data Mining*, [www.ailab.si/orange], Faculty of Computer and Information Science, University of Ljubljana.

[14] I.H. Witten and E. Frank, Data Mining: *Practical Machine Learning Tools and Technique*s, 2nd Edition, Morgan Kaufmann, San Francisco, 2005.

[15] R. Servedio, personal communication, 2006.

[16] E. Walton, *Data Generation for Machine Learning Techniques*, University of Bristol, 2001.

Agile Methods and Quality Models: Towards an Integration in Requirements Engineering

Alexandre Lazaretti Zanatta

ICEG - Ciência da Computação
Universidade de Passo Fundo
Brazil

zanatta@upf.br

Patrícia Vilain

Departamento de Informática e Estatística
Universidade Federal de Santa Catarina
Brazil

vilain@inf.ufsc.br

Abstract. *Significant attention has lately been given to the integration between agile methods and software quality models. This is particularly true for the requirements engineering area, where agile methods neglect some of the documentation and control procedures traditionally used in its process. This work is an effort to extend Scrum so it becomes compatible with CMMI Requirements Management and Development process areas. We analyze Scrum and define some guidelines to meet the requisites of both process areas that are currently not addressed in Scrum. We applied and validated these guidelines in a software development organization. The results show that is possible to use agile methods with CMMI model, as long as the organization is prepared to apply new approaches.*

1. Introduction

The application of agile methods in the software development process has recently demanded the software engineering area to review some of its practices. This is especially true for the requirements engineering area, where some documentation and control procedures usually used in traditional processes are neglected by agile methods. Agile principles presented in the agile manifest [4] show that some of the practices related to requirements engineering are important, such as a clear understanding of requirements. But at the same time, agile methods are against the idea of generating too much documentation as they claim some documentation will never be read. In fact, such methods do not provide details on how requirements should be documented or how requirements traceability should be kept, which may have an impact in the quality of the software produced.

Discussions about the integration between agile methods and software quality models have become more usual, specially in the requirements engineering area [12]. [7], [2], [1], and [6] bring up a relatively recent discussion: Is Agile Software Development compatible with the CMMI model? They present differences and similarities in both approaches and consider that software engineering is going through a new phase called Traditional Software Development versus Agile Software Development. In fact, this is currently a polemic discussion and there has been no agreement whether agile methods are compatible to software quality models such as CMMI or not. [2] also emphasizes the difference in the meaning of quality work between agile methods and quality models. In quality models, quality is mostly defined by the conformance to processes and specifications, while in agile methods quality is determined by customer satisfaction. Further, software quality models, as SW-CMM (Capability Maturity Model for software) and CMMI[1] (Capability Maturity Model Integration), were created for large organizations that usually apply rigid quality standards. [11] comments that, despite the existence of distinct characteristics between agile methods and CMMI model, both have specific plans to try to produce software with quality in the organization.

According to [3], the purpose of CMMI is to establish a guide for an organization to improve its process and its capability for managing, acquiring and maintaining products and services. This work proposes an extension to the agile method Scrum, called xScrum, as an effort to provide compatibility between Scrum and CMMI in Requirements Management and Development process areas. To accomplish this goal, Scrum is analyzed and some guidelines are defined in order to fulfill the requisites that are currently not addressed in Scrum for those two process areas.

Scrum was chosen because it is the agile method with the better capacity to be adapted to larger projects [2] and because it is deficient in mechanisms to control the requirements process. The goal of the agile method Scrum is to define a process for object-oriented software development focused on people and it is indicated for environments where requirements rapidly change. It does not require or provide any specific technique or method for the software development phase. It simply establishes a set of management rules and practices that must be adopted for the project to be successful.

This paper is organized as follows. Section 2 analyzes the conformance of Scrum to Requirements Man-

[1] CMM and Capability Maturity Model are registered in the U.S. Patent and Trademark Office. CMM Integration, CMMI, are service marks of Carnegie Mellon University.

agement and Development process areas of CMMI. Guidelines that extend Scrum with the purpose of addressing the practices in CMMI Requirements Management and Development process areas are presented in section 3. Section 4 describes a real-life application of xScrum. Finally, final remarks are presented in section 5.

2. Scrum Conformance to CMMI

The agile method Scrum is considered a specific method for managing the software development process and it is based in principles such as small teams with at most seven people, requirements that are unstable or unknown and short iterations. A Sprint is a set of development activities conducted over a pre-defined period, which is defined to be 30 days. Product Backlog is the starting point of Scrum and it considered the practice responsible for collecting requirements [9]. Meetings with all project stakeholders take place in order to point out items with all business needs and technical requirements to be implemented, that is, the Product Backlog is a list of functionalities likely to be developed during the project. Scrum was evaluated according to the process areas REQM (Requirements Management) and RD (Requirements Development) of the Engineering category of CMMI.

Each typical work product resulting from specific practices (SP) of these process areas was analyzed to determine if Scrum can obtain that given typical work product in conformance to CMMI. In order to do that, an ordered scale that rates how Scrum addresses such specific practices was defined. The rating scale contains three categories:

NA: Not Addressed – there is very little evidence that the specific practice is presented in Scrum;
PA: Partially Addressed – there are evidences that the specific practice is presented in Scrum;
A: Mostly Addressed – there are strong evidences that the specific practice is presented in Scrum.

For each specific practice, Scrum was rated according to the three categories above, as a result of an individual analysis of typical work products in each practice. The results and interpretation of this analysis are presented next for both process areas REQM and RD.

2.1. Requirements Management (REQM) Process Area

In the following, Scrum is rated according to specific practices of the process area REQM in the Engineering category of the CMMI model.

Table 1 presents the typical work products resulting from each specific practice, with corresponding ratings on how work products are addressed by Scrum.

SP 1.1-1 Obtain an Understanding of Requirements. In Scrum requirements are gathered directly with customers, and selection criteria for these requirements are allocated to each Sprint to be utilized in the division of tasks. Requirements are understood by project participants and placed in a list of items called Product Backlog which represents "what" needs to be analyzed and developed by the team. Therefore, the specific practice REQM SP1.1-1 of the CMMI model is considered Addressed in Scrum.

Table 1. Work products of the REQM process area

SP	Typical Work Products	NA	PA	A
1.1-1	1	-	-	x
	2	-	-	x
	3	-	-	x
	4	-	-	x
1.2-2	1	-	-	x
	2	-	-	x
1.3-1	1	-	-	x
	2	x	-	-
	3	x	-	-
1.4-2	1	x	-	-
	2	x	-	-
1.5-1	1	x	-	-
	2	-	-	x
	3	x	-	-

SP 1.2-2 Obtain Commitment to Requirements. Scrum Master and Product Owner work so that all project participants have a common agreement about the understanding of requirements that will be analyzed in the project. An assessment of the impacts requirements will have on a project is performed by the Product Owner, and the documentation about requirements is registered within accomplished activities during the definition of the Product Backlog and the Sprint Backlog. That way, both work products in practice REQM SP1.2-2 of the CMMI model are addressed by Scrum. Therefore, this practice is considered Addressed in Scrum.

SP 1.3-1 Manage Requirements Changes. During Sprint, the Daily Scrum is used to keep a status of which requirement is being developed. Scrum does not explicitly state that a requirements database or some other alternative ways for storing requirements should be utilized. It also does not mention that there should be a database to support decision making in regard to requirements. This practice is then considered Partially Addressed by Scrum.

SP 1.4-2 Maintain Bidirectional Traceability of Requirements. Scrum does not provide a traceability matrix and it also does not implement a requirement tracking system. Thus, the specific practice REQM SP 1.4-2 of the CMMI model is considered Not Addressed by Scrum.

SP 1.5-1 Identify Inconsistencies between Project Work and Requirements. Since Scrum is based on principles that requirements are unstable and the project is not totally defined, any possible inconsistencies can be verified by the Scrum team during Sprint, The project reaches its end when all the functionality described in the Product Backlog is implemented. The corrective actions are performed in the Sprint Review in the presence of the

Scrum Master and the customer. That way, Scrum minimizes the existence of inconsistencies between Project Work and requirements, but it does not mention how project inconsistencies should be documented. On the other hand, it includes corrective actions that must be taken after the delivery of the Product Increment, which is originated from the final Sprint results. Therefore, the specific practice REQM SP 1.5-1 of the CMMI model is considered Partially Addressed by Scrum.

2.2. Requirements Development (RD) Process Area

Scrum is also rated according to specific practices of the process area RD in the Engineering category of the CMMI model.

Table 2 presents the typical work products resulting from each specific practice, with corresponding ratings on how work products are addressed by Scrum.

Table 2. Work products of the RD process area

SP	Typical Work Products	NA	PA	A
1.1-2	1	x	-	-
1.2-1	1	-	-	x
	2	-	-	x
	3	-	-	x
2.1-1	1	-	-	x
	2	-	-	x
	3	-	-	x
2.2-1	1	x	-	-
	2	x	-	-
	3	x	-	-
	4	x	-	-
	5	x	-	-
2.3-1	1	x	-	-
3.1-1	1	-	-	x
	2	-	-	x
	3	-	x	-
	4	-	x	-
	5	x	-	-
	6	-	-	x
3.2-1	1	-	-	x
	2	x	-	-
	3	x	-	-
3.3-1	1	x	-	-
	2	-	-	x
	3	-	-	x
	4	x	-	-
3.4-3	1	x	-	-
3.5-1	1	-	-	x
3.5-2	1	-	-	x

SP 1.1-1 Collect Stakeholder Needs. The Product Owner is responsible for creating and including the list of Stakeholder needs into the Product Backlog. Therefore, as stakeholder needs are identified and collected, the specific practice RD SP 1.1-1 is considered Addressed by Scrum. This practice does not define work products and is applied only in the Continuous representation of the CMMI model.

SP 1.1-2 Elicit Needs. Scrum does not propose the use of techniques to elicit needs, such as use cases, prototypes, JAD (Joint Application Development), etc. Therefore, the specific practice RD SP 1.1-2 is considered Not Addressed by Scrum.

SP 1.2-1 Develop the Customer Requirements. The Sprint Backlog that results from the Sprint Planning Meeting defines what needs to be developed during the 30 days of Sprint. In Scrum, the development of customer requirements is monitored by the Product Owner, whose assignment is to determine whether requirements are being developed according to what was requested by the customer. Customer constraints on the conduct of verification and validation are conducted by both Product Owner and Scrum Master, respectively. That way, as all work products are addressed, the specific practice RD SP 1.2-1 is considered Addressed by Scrum.

SP 2.1-1 Establish Product and Product-Component Requirements. In Scrum product and component requirements are detailed in the Product Backlog, along with all functionality, features, infrastructure, architecture and technology the product must offer. The definition of costs and budgets for the project being developed in Scrum is pointed out by [9], thus indicating that the work product of item 1 is also addressed by Scrum. Therefore, the specific practice RD SP 2.1-1 is considered Addressed by Scrum.

SP 2.2-1 Allocate Product-Component Requirements. The allocation of requirements to product components is not a detailed process in Sprint as Scrum does not define techniques for this phase. Even though the work products of this specific practice may be present during development, these are not specified by Scrum. The specific practice RD SP 2.2-1 is then considered Not Addressed in Scrum.

SP 2.3-1 Identify Interface Requirements. Requirements for interfacing with other systems are not detailed in Scrum. Therefore, the specific practice RD SP 2.3-1 is considered Not Addressed.

SP 3.1-1 Establish Operational Concepts and Scenarios. The Product Backlog contains a description of business needs and requirements that are the basis for product development. It is also expected to describe the sequence of events that can occur while using the system, which correspond to a scenario that describes the use of the product being developed. The Product Backlog and the Scrum Master makes available to stakeholders all concepts utilized during the project. Although the elaboration of scenarios is not detailed by Scrum, the sequence of events handled by the system is described. Given that Scrum does not define techniques to elicit requirements, such as use cases. On the other hand, new requirements can be added after the Sprint Review Meeting, when the Scrum Master reports work results to other participants. At that point, in case a new requirement is identified, it is added to the project. It is not very clear which rate must be assigned to Scrum for this specific practice. Therefore, the specific practice RD SP3.1-1 is considered Addressed by Scrum.

SP 3.2-1 Establish a Definition of Required Functionality. In Scrum, the functional architecture is indicated when the Product Backlog is created. However, it is up to the developer to apply object oriented analysis techniques such as use cases and activity diagrams, as well as the identification of services, this specific practice RD SP3.2-1 is considered Not Addressed by Scrum.

SP 3.3-1 Analyze Requirements. In Scrum, the process of requirement analysis tries to determine if all system functionalities/features described in the Product Backlog List (which results from the analysis of the Product Backlog) can be resolved by the listed requirements and if these requirements do not overlap or conflict with each other. However, it does not produce a defect report as suggested this specific practice. Also, this practice is addressed in Scrum because requirements are developed according to stakeholder needs so that key or fundamental requirements are then elicited. Requirements analyzed during the elaboration of functionalities do not refer to technical performance measures to be followed during project developments. Since it is not clear this practice is addressed in Scrum, the rating for the work product closest to its goal was taken into account, which in this case is work product 2. Therefore, the specific practice RD SP3.1-1 is considered Addressed by Scrum.

SP 3.4-3 Analyze Requirements to Achieve Balance. Scrum does not define models, simulations or prototypes to analyze risks in requirements. It assumes that possible risks should be quickly eliminated during daily meetings or at the rapid system delivery. Since Scrum does not analyze risks, this work product is not addressed. Therefore, the specific practice RD SP 3.4-3 is considered Not Addressed in Scrum.

SP 3.5-1 Validate Requirements. Requirements validation in Scrum takes place after the phase known as Post Sprint Demonstration and Meeting, when requirements are verified to assure they are properly described. The results of requirements validation, the only work product of this practice, are presented to the Scrum team before they start the development process. Therefore, the specific practice RD SP 3.5-1 is considered Addressed in Scrum.

SP 3.5-2 Validate Requirements with Comprehensive Methods. This specific practice differs from the previous only because it requires a method for requirements validation. The meeting known as Post Sprint Demonstration and Meeting is the method utilized in Scrum to validate requirements. Therefore, the specific practice RD SP 3.5-2 is considered Addressed in Scrum.

3. xScrum: Extending Scrum to Conform with CMMI

This section proposes extensions to the agile method Scrum that allows it to address all specific practices in the process areas REQM and RD, thus trying to become compliant to CMMI in those areas. We call this extension “xScrum”. For each specific practice not addressed or partially addressed by Scrum (according to the ratings presented in the previous section), a guideline is proposed with the necessary additions in order to address that practice. All templates and documentation defined in the follow guidelines are available at http://vitoria.upf.br/~zanatta/xScrum.

Guideline 1: Manage Requirements Changes

This guideline aims to address the specific practice REQM SP 1.3-1 of CMMI. In order to manage changes in requirements, this guideline suggests the creation of a document called Requirements Record. This document (i) records new requirements or the changes in existing ones and (ii) keeps a log of the impact caused by each new or changed requirement on others requirements. The Scrum Master is responsible for maintaining this document, which is used during Sprint after the definition of the Product Backlog. The proposed document consists of three sections. The first section (Revision Log) contains the general data about a given requirement: date, author’s name and, change operation (inclusion, modification or exclusion) and revision number (i.e. version). The second section (Requirements Descriptions) is the most important one. It keeps specific data about each requirement, such as its type, priority, source, version, etc. Finally, the third section (Impact on Requirements) registers the impacts of this change on other system requirements.

Guideline 2: Identify Inconsistencies between Project Work and Requirements

The purpose of this guideline is to address the specific practice REQM SP 1.5-1 of CMMI. It proposes the use of another document called Component Product Backlog. This document is utilized to identify inconsistencies between existing project work and requirements, and it also helps the detection of deviations, as well as the visualization of inconsistencies and the identification of design elements that are out of the development scope. The proposed document consists of a date, the requirement identification and a brief description of the inconsistency of such requirement in relation to components. The Scrum Master is the responsible for maintaining this document, which is created during the Sprint and validated during the Sprint Review Meeting.

Guideline 3: Maintain a Traceability Matrix

The purpose of this guideline is to address the specific practice REQM SP 1.4-2 of CMMI. It recommends the creation of a traceability matrix that records the relationships among the requirements, between requirements and stakeholders, and between requirements and project modules [10]. The traceability matrix, along with documents proposed in guidelines 1 and 2, acts as a requirement tracking system (which is also a work product REQM of SP 1.4-2). The proposed traceability matrix contains two sections. The upper section (Review Log) contains a date, a project code and a sprint code where requirements will be traced. The lower section (Requirements Traceability) is used to track any inconsistencies ("I") between a given component and requirement, dependencies ("D") between two requirements, or if a component allocates ("A") another component. This matrix serves as a basis to evaluate the impact of changes and to estimate the amount of effort and cost for modifying a project artifact. To avoid maintaining a large traceability matrix, it is possible to define one matrix for each Sprint, referring to requirements and artifacts defined in other Sprints whenever needed. The Scrum Master is responsible for maintaining the traceability matrix, which is used during the Sprint.

Guideline 4: Allocate Product-Component Requirements

The purpose of this guideline is to address the specific practice RD SP 2.2-1 of CMMI. The idea is to define, for each project work, a document to keep information about product components and the requirements allocated to these components. This document also contains information about temporary requirements, derivations of requirements and their relationships. The proposed document is called Requirements Allocation and it consists of two parts. The first one contains general data about the component allocation: date, project code plus component, and a brief description of the component. The second part contains the identification, description, derivations, and relationships of each allocated requirement. The Scrum Master is the responsible for maintaining this document, which is used during the Sprint Review Meeting.

Guideline 5: Identify Interface Requirements

The purpose of this guideline is to address the specific practice RD SP 2.3-1 of CMMI. It suggests the use of a document that keeps details about software requirements, including constraints on product features, interfaces with others applications, a description of the domain, and additional information about the product matter. This document is called View Documentation and it serves as a kind of contract between users and developers, so it must be written in high level language. The proposed document consists of two sections. The first section (General Data) contains the revision log including its date, author's name and revision (i.e. version). The second section (View Documentation) contains a date, project code, general description of the product, product view, and name of the person responsible for the project. The Scrum Master and the Client are responsible for maintaining this document, which is used during the elaboration of the Product Backlog.

Guideline 6: Utilize a Technique to Elicit Requirements

The purpose of this guideline is to address the specific practice RD SP 1.1-2 of CMMI. It proposes to use of the technique to elicit requirements called JAD (Joint Application Development). This technique was chosen because it is strongly based on meetings like as in Scrum and also because it has been applied in another agile method called ASD (Adaptive Software Development) [4]. JAD is a group dynamic technique originated at IBM laboratories in the end of the 60's and it has been successfully used in ASD [4].

Guideline 7: Analyze Risks

The purpose of this guideline is to address the specific practice RD SP 3.4-3 of CMMI. Since Scrum does not define any models, simulators or prototypes to analyze the risks of stakeholder needs and their constraints, the idea proposed here is to define a document that describes the risks produced by a requirement if the functional architecture changes. This document also describes the predicted impact. The proposed document is called Risk Analysis and it consists of three parts. The first part contains general information about the project and requirement in question. The second part keeps information about risk: priority, type, probability and stability. The third part contains an evaluation of risks produced in case a change occurs in the functional architecture besides what has to be done. Scrum Master and Client are both responsible for maintaining this document, which is used during the Sprint.

Guideline 8: Establish a Definition of Required Functionality

The intent of this guideline is to address the specific practice RD SP 3.2-1 of CMMI. Scrum identifies required functionalities of the system through the Product Backlog. However, it does not mention explicitly that the use case technique should be utilized. Use cases are the main work product in RD SP 3.2-1. Hence, this guideline simply suggests the inclusion of the use case technique in Scrum.

4. Case Studies

In order to validate the extensions proposed to Scrum, we have been used xScrum in two case studies developed in a software development organization.

The first case study was on an internet portal dedicated to provide weather data to farmers. It included the development of the following functionalities: (a) to maintain a database about farmers and producers; (b) to incorporate a module about farmers' alimentation; and (c) to develop an authorization module. The main goal of the second case study was to develop a system to obtain climatic information from a website containing weather data and insert it into a local database.

In the following, we present just the first case. The client required an internet portal release within 30 days. So, the features were prioritized and put into a backlog queue. A small team in the organization was given the responsibility to establish a mechanism for everyone and to determine how things are going. At the Daily Scrum Meetings, management as certained progress and the team members making progress on their work. The team demonstrated what they have accomplished.

During the Sprints, an internet portal was built using Java. At the end of the first Sprint, the database about farmers and producers was evaluated and the manager, along the team, decided that the next step should to incorporate a module about farmers's alimentation. Empirically, based on what they have been able to complete, and what is now the most important work to perform, the next Sprint is set up. The end of Sprint Meeting after this Sprint was a great celebration. The weather data to farmers had shown it to be capable of delivering a real solution of scrum to a real business. This case study was done in approximately 22 days.

At the end of the case studies, we could conclude that an organization that uses Scrum as its software development process and desires to adapt this method to support the REQM e RD process areas of CMMI can apply the extensions proposed here, which would help such organization to get a quality certification for its development process.

5. Final Remarks

In this paper we analyzed Scrum against CMMI to verify if agile methods and software quality models can coexist considering they are based on so distinct principles and ideas. The results demonstrated that Scrum does not completely satisfy CMMI practices. Hence, to allow Scrum to be in conformance to REQM and RD process areas of the CMMI model, we proposed an extension to Scrum, called *xScrum*, with a set of guidelines. These guidelines are based on the following principles: all practices of REQM e RD process areas must be addressed in Scrum; all solutions must conform to Scrum philosophy; and any known software engineering practices that can help solve the problems found must be applied.

Although there is some previous research on integrating agile methods with quality models, the idea of proposing extensions to provide conformance between Scrum and CMMI is believed to be knew. The work of [7] compares Scrum to SW-CMM and not CMMI. It claims that the Requirements Management process area is addressed by Scrum (as opposed to our work). It also claims that only four of the eighteen process areas of SW-CMM are addressed by Scrum, but it does not suggest any extensions for Scrum to address the other process areas. [11] attests that there are significant differences between agile methods and CMMI, and observe that 42% of process areas in CMMI are in conflict with practices adopted by agile methods. They believe that these differences can be minimized with the adoption of organizational policies in the software development process. Existing work in general compares agile methods to SW-CMM instead of CMMI, usually taking Extreme Programming (XP) as the agile method in question. [7] points out that XP can be used with SW-CMM, but emphasizes that not all XP practices conform to SW-CMM. [8] comments that XP practices and SW-CMM are compatibles but does not present guidelines of how to establish such compatibility. [5] also attests that the use of XP with SW-CMM is valid, but they focus specifically on level 2.

As a continuation of work presented here, we are extending Scrum to address remaining process areas of the *Engineering* category: *Technical Solution*, *Product Integration*, *Verification* and *Validation*. We also intend to perform a systematic validation of this method in others software organizations while defining specific metrics to evaluate xScrum as an agile method and comparing its efficiency with its predecessor Scrum.

References

[1] Ahern, Dennis M.. CMMI distilled: appraisals for process improvement. Upper Saddle River: Addison-Wesley, 2005. 218 p.

[2] Boehm, B., DeMarco, T.: The Agile Methods Fray. IEEE Computer Science, 6 (2002) 90-91

[3] Chrissis, M.B., Konrad M., and Shrum S.: CMMI: Guidelines for Process Integration and Product Improvement. Addison Wesley (2003)

[4] Highsmith, J.: Agile Software Development Ecosystems. Addison Wesley, (2002)

[5] Nawrocki, J., Walter, B., Wojciechowski, A.: Comparasion of CMM level 2 and extreme programming. In: Proceedings of 7th European Conference on Software Quality, Helsinki, (2002) 288-297

[6] Paetsch, F., Eberlein, A., Maurer, F.: Requirements Engineering and Agile Software Development. In: Proceedings of the IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (2003) 308-313

[7] Paulk, M.C.: Extreme Programming from a CMM Perspective. IEEE Software, 18, 6 (2001) 19-26

[8] Reifer, D.J.: XP and the CMM. IEEE Software, Vol 20, 3 (3), May/June pp. 14-15 (2003)

[9] Schwaber, K., Beedle, M.: Agile Software Development with SCRUM. Prentice Hall, (2002)

[10] Sommerville, I.: Software Engineering. Addison-Wesley (2003)

[11] Turner, R., Jain, A.: Agile Meets CMMI: Culture clash or common cause. XP/Agile Universe (2002) 153-165

[12] Zanatta, A. L., Vilain P. Uma análise do método ágil Scrum conforme abordagem nas áreas de processo Gerenciamento e Desenvolvimento de Requisitos do CMMI In: VII Workshop em Engenharia de Requisitos, 2005, Porto. Proceedings of WER05. Porto - Portugal: FEUP - Faculdade de Engenharia da Universidade do Porto, 2005. v.1. p.209 – 220 (in Portuguese)

A Framework of Hierarchical Requirements Patterns for Specifying Systems of Interconnected Simulink/Stateflow Modules

Changyan Zhou[†] , Ratnesh Kumar[‡] , Devesh Bhatt[♮] , Kirk Schloegel[♮] , Darren Cofer[♮]
[†]Industrial & Enterprise Systems Eng., University of Illinois at Urbana-Champaign, IL
[‡]Electrical & Computer Engineering, Iowa State University, Ames, IA. (rkumar@iastate.edu)
[♮]Honeywell Aerospace, 3660 Technology Drive, Minneapolis, MN 55418
({devesh.bhatt,kirk.schloegel,darren.cofer}@honeywell.com)

Abstract

Simulink/Stateflow is a graphical modeling and simulation tool (developed by Matlab) for systems with continuous dynamics mixed with switching logics, and is being widely used in industry for controls and diagnosis applications. Motivated by the need for developing a requirements specification approach for systems of interconnected Simulink/Stateflow modules that supports traceability and formal analysis (such as consistency, completeness, correctness), and that is scalable and also easy to use and to modify, we propose a hierarchical patterns-based approach. The approach supports modularity and hierarchy to handle complex systems. Further, it supports both the formal analysis and the traceability of the requirements, and also facilitates the design process by being mapped to the underlying design space. The patterns are developed following a component-oriented paradigm, and as a result they also facilitate their reuse. We develop and present a number of requirements patterns for commonly used Simulink/Stateflow modules and show how they can be hierarchically composed.

Keywords: Software engineering, requirements engineering, patterns, components reuse, formal methods

1 Introduction

Due to the evidence that requirements errors (such as incorrect, misunderstood or omitted requirements) are difficult to debug and expensive to correct later in the developmental life-cycle, writing complete, correct, consistent, testable, and traceable software requirements specifications is an important exercise in the software development process. The importance of good requirements specification has led the researchers and software industry to develop approaches and tools for capturing and managing them.

Examples of approaches for specifying requirements include problem frames [6], intent specification [13], and multiple views [16]. Requirements documentation and management tools include DOORs, Requisite Pro, and Cradle. The higher-level requirements are generally expressed in natural language, diagrams [1], charts [5, 19], and tables, and lower-level ones in various specification/design/modeling languages such as UML, architecture description language (ADL), specification description language (SDL), and Hardware Description Language (HDL). Natural language is easily understandable but the use of it in describing complex, dynamic systems has the problems of ambiguity, inaccuracy, incompleteness, and inconsistency. Formal languages, on the other hand, can be difficult to learn and expensive to apply correctly since the requirements must be understood by both the developers and the customers, and many are not fluent in a formal language.

State-diagram/table notations have been demonstrated to offer a precise, relatively compact notation for specifying system requirements in a wide range of applications, including avionics systems. State-diagrams/tables can be assigned a precise mathematical semantics and thus can be analyzed mechanically to expose defects in requirements specifications. State-diagram/table notations based requirements specification languages include Software Cost Reduction (SCR) [12], Requirements State Machine Language (RSML) [15, 17] and its simplified versions RSML-e (RSML without event) [18], and SpecTRM-RL (Specification Toolkit and Requirements Methodology Requirements Language) [14]. Software tools utilizing the state-diagram/table notations include Scade/Lustre, State-

Chart/StateMate, and Esterel.

Using state-diagram/table notations, the system requirements are primarily captured at discrete or logic level. While this can be appropriate for specifying certain types of systems at a high-level of abstraction, this may not be of much help for specifying requirements at later phases of design, i.e., at lower-levels of abstraction involving both the discrete and the continuous dynamics. The system of requirements patterns we develop supports specifications also at such lower-levels of abstraction.

Recently there has been interest in the so called patterns-based approaches, where a pattern is a template for capturing a specific requirement. There has been a growing interest in the identification and use of design patterns [3], examples of which include interface patterns, proxy patterns, composite patterns, state patterns, and scheduler patterns. Pattern using limited vocabulary, grammar and semantic domain have also been developed [7, 4].

Patterns for specifications as well as for requirements are also being considered. Dwyer et al. [2] observed that most temporal logic specification formulas of LTL, CTL, and QRE (quantified regular expression) fall into a relatively small number of categories, and accordingly developed a set of specification patterns that can be used for specifying properties. In the same spirit, [10] introduced a collection of real-time specification patterns corresponding to the logics of MTL (metric temporal logic), TCTL (timed CTL) and RT-GIL (RT-graphical interval logic). In [9, 8] researchers investigated (high-level) requirements patterns for certain embedded systems. Specifically, they explored how object-oriented modeling notations can be used to capture structural requirements.

2 Our Approach to Requirements Specification

The requirements governing systems with time-driven dynamics are often expressed in terms of time-domain behaviors such as rise-time, overshoot, settling-time, or in form or frequency-domain behaviors such as bandwidth. The existing requirements specification approaches are mainly for event-driven dynamics (one that can be modeled using automata extended with discrete-valued variables), and are not adequate for time-driven dynamics (one that is modeled using differential or difference equations). This is one of the motivation for our research.

We develop formalized requirements for commonly used Simulink/Stateflow modules. Simulink/Stateflow is a widely accepted tool for modeling, design, and analysis of control and diagnosis systems such as those used in aircrafts, automobiles, medical devices, power system, nuclear and chemical plants, and manufacturing factories. Such systems possess both timed-driven (modeled using Simulink modules) and event-driven (modeled using Stateflow modules) dynamics. These formalized requirements are what we call requirements patterns. A requirements pattern describes the inputs, the outputs, the behavior, and the constraints of a basic Simulink/Stateflow module. This patterns-based approach is motivated by the ease of use, modularity, reuse, and modifiability. (Changes in the requirements can be made easily and in a cost-effective manner by replacing any of the existing patterns or their parameters with new ones.) To support traceability and complexity-management, the approach is kept modular and hierarchical. Finally for the fact that the patterns are formalized our approach supports a formal analysis (such as completeness, consistency, and correctness).

3 An Example

To illustrate our approach we present an example of a localizer signal processing unit along with its current set of requirements, discuss some of the limitations, and present a new set of requirements according to our own approach. The example of the localizer signal processing unit presented has been taken from the context of a flight control system developed by Honeywell Aerospace. The unit is used to filter the incoming signal when it is "valid for use", and otherwise it produces a constant output. We next give the currently used requirements for the localizer signal processing unit (listed as R1-R5 below), whereas Figure 1 shows the Simulink design model of the unit.

R1: The incoming localizer signal shall be declared valid for use by the system if the raw signal valid is true continuously for at least one second.
R2: The incoming localizer signal shall be declared invalid for use by the system if the raw signal valid is false continuously for at least 0.5 seconds.
R3: The incoming localizer signal shall be attenuated by -3dB at a frequency of 5 Hz and by -20dB at 50 Hz.
R4: When the raw localizer signal is invalid, the input to the attenuation filter shall be held to the last valid value of the raw input.
R5: When the incoming localizer signal is declared invalid for use by the system, the output of the value to the system shall be set to 600 microAmps.

From the system requirements and its Simulink model, we observe that the requirements R1 and R2 refer to the same Simulink module "Debounce", R4 refers to two different Simulink modules, and R3 refers

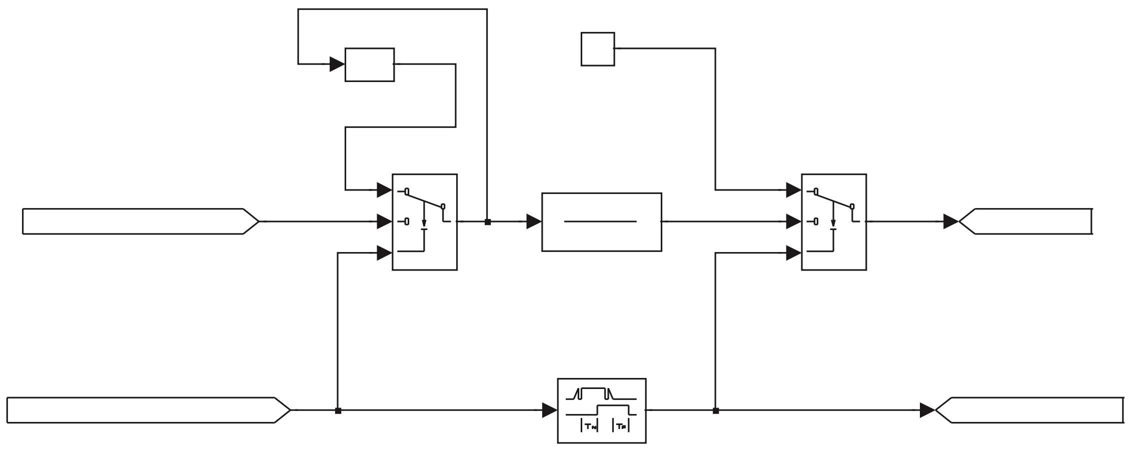


Figure 1. Localizer unit in Simulink

to the Simulink module "lag". The following issues become evident when examining the above requirements.

1. When two or more requirements refer to the same module (such as R1 and R2 refer to the module "Debounce"), a reasonable question is whether there exists any relation between the requirements so they can be combined and written as a single requirement.
2. R4 maps to two different modules, whereas it can be implemented using a single module in the HAM library.
3. It is questionable how the requirement R3 is mapped to the "lag" module with the specific transfer function $\frac{1}{0.0318s+1}$, for R3 specifies only two points in the frequency-domain. It seems that the designer is using a mental model with additional details of the lag module, implying that the requirement R3 is ambiguous and incomplete.
4. R4 seems to be not about *what* the system should do, rather *how*, whereas a good set of requirements should only specify what a system should do and how.
5. No specific format is followed for writing the requirements.

The approach we propose is able to avoid the issues raised above.

4 Basic Requirements Patterns

For illustration, we present examples of some basic and composite requirements patterns that we have developed [20]. The basic patterns are for some of the existing Simulink modules in the HAM library. Each pattern is associated with a *requirements description* and a *temporal logic representation*. The requirements description part describes the interface (inputs/outputs and their types), the parameters, the functionality, and the constraints. The functionality is written in a natural language so it can be easily understood by all users. The temporal logic representation part first describes the behavior in a language that is amenable for a temporal logic specification. Next, an encoding in the first-order metric temporal logic is provided. (The syntax and semantics of the metric temporal logic can be found in a standard reference such as [11].) Having been encoded within a requirements pattern, the temporal logic properties become ready-to-use for a formal analysis. The requirements description is standardized, and for a specific application only the corresponding inputs, outputs, and parameters need to be identified and specified.

IsValid Pattern

- `Requirements Description:`
  `Inputs:` *control input* In(type)
  `Outputs:` *control output* Out(type)
  `Parameters:` *debounce-on time* tON(type), *debounce-off time* tOFF(type)
  `Functionality:` When the *control input* is ON for at least *debounce-on time, a certain signal* shall be valid for use, and otherwise when the *control input* is OFF for at least *debounce-off time, the said signal* shall be invalid for use.
  `Constraints:` Initial value of the output must be set to either 0 or 1, and tON and tOFF must be the multiples of the sampling period.

- `Temporal Logic Representation:`
  `Behavior:` It is always the case that if the input is 1 for at least tON time unit, then the output is set to 1, and if the input is 0 for at least tOFF time unit, then the output is set to 0.
  `Instantiation:`
  $\Box_{[0,\infty)}[(\Box_{[0,tON]} In.Value = 1 \Rightarrow \Diamond_{\leq 1} Out.Value = 1)$
  $\wedge (\Box_{[0,tOFF]} In.Value = 0 \Rightarrow \Diamond_{\leq 1} Out.Value = 0)]$

Filter Pattern

- `Requirements Description:`
  `Inputs:` *input signal* In(type)
  `Outputs:` *output signal* Out(type)
  `Parameters:` *passing band* $[f_1, f_2]$ with *attenuation* K_1db, *roll-off band* $[f_1 - \triangle, f_1]$ or $[f_2, f_2 + \triangle]$ with *attenuation* K_2db/octave, and *attenuation* K_3db for *non-passing band*
  `Functionality:` Within the *pass-band* $[f_1, f_2]$, the *output* shall be the *input* with attenuation of less than K_1dB, within the *roll-off band* $[f_1 - \triangle, f_1]$ or $[f_2, f_2 + \triangle]$, the *output* shall be attenuated at a rate of K_2dB/octave, and otherwise the *output* shall be attenuated more than K_3dB.
  `Constraints:` None
- `Temporal Logic Representation:`
  `Behavior:` It is always the case that if the frequency is within the range of $[f_1, f_2]$, then the output signal shall be passed with attenuation of less than K_1dB; if the frequency is within the roll-off regions $[f_1 - \triangle, f_1]$ or $[f_2, f_2 + \triangle]$, then the output signal shall be attenuated at a rate of K_3dB per octave; otherwise, the output signal shall be attenuated more than K_3db.
  `Instantiation:`
  $\forall f : [f_1 \leq f \leq f_2 \Rightarrow Out.Magnitude(f)/In.Magnitude(f) \leq K_1]$
  $\wedge[(f_1 - \triangle \leq f \leq f_1) \vee (f_1 \leq f \leq f_2 + \triangle) \Rightarrow \frac{d}{df}(Out.Magnitude\ (f)/In.Magnitude(f)) \leq K_2]$
  $\wedge[(f \leq f_1 - \triangle) \vee (f \geq f_2 + \triangle) \Rightarrow Out.Magnitude(f)/In.Magnitude(f) \geq K_3]$

Sample-&-Hold Pattern

- `Requirements Description:`
  `Inputs:` *input signal* In(type), *control signal* CtrIn(type)
  `Outputs:` *output signal* Out(type)
  `Parameters:` None
  `Functionality:` The *input signal* shall be sampled if and only if it is valid for use.
  `Constraints:` The output type is the same as the input type. The control input is compared to threshold value of 0.5 to determine if it is logic 0 or 1.
- `Temporal Logic Representation:`
  `Behavior:` It is always the case that if the control input CtrIn is logic 1, then the output signal equals the input signal; otherwise (i.e., CtrIn is logic 0), the output remains constant at its last sample value.
  `Instantiation:`
  $\forall t : [(CtrIn = 1 \Rightarrow Out.Value[t] = In.Value[t]) \wedge (CtrIn = 0 \Rightarrow Out.Value[t] = Out.Value[t-1])]$

Delay Pattern

- `Requirements Description:`
  `Inputs:` *input signal* In(type)
  `Outputs:` *output signal* Out(type)
  `Parameters:` *delay time* T(type)
  `Functionality:` The *input signal* shall be delayed for T *time unit.*
  `Constraints:` T must be a multiple of the sampling rate. Initially, the output signal is set to either ON or In.
- `Temporal Logic Representation:`
  `Behavior:` Initially, the output signal is set to either ON or In. It is always the case that if the input signal is ON (resp., OFF) for at least T seconds, then the output signal is set to ON (resp., OFF); otherwise, the output is set to OFF (resp., ON).
  `Instantiation:`
  $[(Out.InitialValue = 1) \vee [\Box_{[0,\infty)}(\Box_{[0,T]} In.Value = 1 \Rightarrow \Diamond_{\leq 1} Out.Value = 1)]] \vee$
  $[(Out.InitialValue = 0) \vee [\Box_{[0,\infty)}(\Box_{[0,T]} In.Value = 0 \Rightarrow \Diamond_{\leq 1} Out.Value = 0)]]$

5 Composite Requirements Patterns

By *composite patterns* we mean the patterns that are formed by composing the basic ones. A complex system is normally composed of several interconnected components. Correspondingly, requirements patterns can be composed to derive requirements patterns for system composed of basic modules. Here we present examples of two types of composite patterns, denoted as $A\ then\ B$ (cascade of modules A and B), and $A\ with\ B\ in\ feedback$ (feedback composition of modules A and B). Other composition mechanisms, specially those of Stateflow modules, are also possible and will be developed as part of the project.

A then B Pattern

- `Requirements Description:`
  `Inputs:` *input signal* In(type)
  `Outputs:` *output signal* Out(type)

`Parameters:` None
`Functionality:` The composition shall be cascade of A and B.
`Constraints:` *input* of A *then* B = *input* of A, *output* of A *then* B = *output* of B, and *input* of B = *output* of A.

- `Temporal Logic Representation:`
  `Behavior:` It is always the case that the output Out is equal to the output of B, the input of B is equal to the output of A, and the input of A is equal to the input In.
  `Instantiation:`
  $\forall t : [In.Value(t) = A.In.Value(t) \wedge A.Out.Value(t) = B.In.Value(t) \wedge Out.Value = B.Out.Value(t)]$

A with B in feedback Pattern

- `Requirements Description:`
  `Inputs:` *input signal* In(type)
  `Outputs:` *output signal* Out(type)
  `Parameters:` None
  `Functionality:` The composition shall be feedback of A and B.
  `Constraints:` (*input* of A *with* B *in feedback* − *output* of B) =*input* of A, *output* of A *with* B *in feedback* = *output* of A = *input* of B.
- `Temporal Logic Representation:`
  `Behavior:` It is always the case that the output Out is equal to the output of A, the input of B is equal to the output of A, and the input of A is equal to the difference of the input In and the output of B.
  `Instantiation:`
  $\forall t : [(In.Value(t) - B.Out.Value(t)) = A.In.Value(t) \wedge Out.Value(t) = A.Out.Value(t) = B.In.Value(t)]$

6 Specifying System Requirements and an Illustration

The requirements for a system can be specified using *conditional assertions* (or rules) of the form:

> *When inputs/past-outputs of the system satisfy a certain condition, the present/future outputs of the system shall be computed in accordance with certain basic or composite requirements patterns.*

We revisit the localizer signal processing unit example and present the requirements for it using our approach.

Localizer Signal Processing Unit Requirements:

`Inputs:` rawLocalizer(float), rawLocalizerValid(boolean)
`Outputs:` Localizer(float)

High-level requirements:

`When` the input signal rawLocalizer is valid for use, output signal Localizer `shall be` sampled then filtered input. `Otherwise`, output `shall be` 600 μA.

Low-level requirements:

`Req1 (IsValid Pattern)`: When rawLocalizerValid is ON for at least 1s, the input signal rawLozalizer shall be valid for use; when rawLocalizer is OFF for at least 0.5s, the input signal rawLozalizer shall be invalid for use.
`Req2 (Filter Pattern)`: Within the pass-band [0, 5], the output Localizer shall be the sampled input with attenuation of less than -3dB, within roll-off band [5, 50], the output Localizer shall be attenuated at a rate of -17/45dB/octave, and otherwise the output Localizer shall be attenuated more than -20dB.
`Req3 (Sample-&-Hold Pattern)`: The rawLocalizer shall be sampled if and only if it is valid for use.

It follows from the above composite requirements that the localizer signal processing unit could consist of three basic HAM modules, namely, IsValid, Filter, and Sample-&-Hold. Also it is evident that the issues associated with the previously used requirements (refer Section 3 are largely resolved. Issue 1 is resolved by having a single requirement (Req1) that captures both the conditions under which the input signal is valid for use. Issue 2 is resolved by rephrasing R4 using a single Sample-&-Hold requirements pattern (Req3). This is made possible by taking a patterns-based approach to requirements. Issue 3 is resolved by using a Filter requirements pattern (Req2), which, by virtue of being pre-defined, insists that a complete set of filter parameters be provided. Issue 4 is resolved since our requirements only state what the system should do, and not how. Finally, issue 5 is resolved since we follow a systematic approach in which all requirements are formalized through the use of patterns.

7. Conclusion

We proposed a hierarchical patterns-based approach for requirements specification for systems that are being designed in the Simulink/Stateflow environment. The requirements patterns will be formalized and so a user only needs to specify the inputs, the outputs, and the parameters, making those patterns easy to use. On the other hand, the patterns will encapsulate formal temporal logic specifications, making them amenable for a formal analysis. We have presented examples of the some of the requirements patterns.

The patterns-based approach is motivated from their ease of use and understanding in specifying requirements. To support modularity, modifiability, and reuse, the approach is component-based in the sense that patterns are developed for certain basic components (or modules). To support complexity management, the approach is hierarchical that uses certain "composition patterns" to generate higher-level composite patterns from the basic ones. This further supports traceability. To support verifiability and formal analysis (such as completeness, consistency, and correctness), patterns embed encodings of the input-output behaviors in the first-order metric temporal logic. Finally to support specification of systems possessing both discrete and continuous dynamics, patterns for certain commonly used Simulink/Stateflow modules developed by Honeywell researchers is provided.

Acknowledgment: The work was done while the first author was a graduate research intern at Honeywell Technology Center, Minneapolis, MN in summer 2006, and the internship support of Honeywell is acknowledged. The second author acknowledges the support from NSF under the grants NSF-ECS-0218207, NSF-ECS-0244732, NSF-EPNES-0323379, NSF-ECS-0424048, and NSF-ECS-0601570.

References

[1] W. R. Cyre. Capture, integration, and analysis of digital system requirements with conceptual graphs. *IEEE Trans. Knowl. Data Eng.*, 9(1):8–23, 1997.

[2] M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in property specifications for finite-state verification. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 411–420, 1999.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[4] J. J. Granacki and A. C. Parker. Phran-span: a natural language interface for system specifications. In *DAC '87: Proceedings of the 24th ACM/IEEE conference on Design automation*, pages 416–422, New York, NY, USA, 1987. ACM Press.

[5] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, and A. Shtul-Trauring. Statemate: a working environment for the development of complex reactive systems. In *ICSE '88: Proc. of the 10th international conference on Software engineering*, pages 396–406, Los Alamitos, CA, USA, 1988.

[6] M. Jackson. *Software Requirements and Specifications: A Lexicon of Practice.* Addison Wesley, 1995.

[7] R. Kittredge and J. Lehrberger. *Sublanguage: Studies of Language in Restricted Semantic Domains.* de-Gruyter, New York, 1982.

[8] S. Konrad, B. Cheng, and L. A. Campbell. Object analysis patterns for embedded systems. *IEEE Transactions on Software Engineering*, 30(12):970–992, Dec. 2004.

[9] S. Konrad and B. H. Cheng. Requirements patterns for embedded systems. In *Proc. of the 10th Anniversary IEEE Joint international Conference on Requirements Engineering*, pages 127–136, Washington, DC, Sept. 2002.

[10] S. Konrad and B. H. C. Cheng. Real-time specification patterns. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 372–381, New York, NY, USA, 2005. ACM Press.

[11] R. Koymans. Specifying real-time properties with metric temporal logic. *RealTime Systems*, 2(4):255–299, 1990.

[12] G. Lee, J. Howard, and P. Anderson. Safety-critical requirements specification and analysis using spectrm. In *Proc. of the 2nd Meeting of the US Software System Safety Working Group*, Feb. 2002.

[13] N. G. Leveson. Intent specifications: An approach to building human-centered specifications. *IEEE Trans. Software Eng.*, 26(1):15–35, 2000.

[14] N. G. Leveson, M. P. Heimdahl, and J. D. Reese. Designing Specification Languages for Process Control Systems: Lessons Learned and Steps to the Future. In *Seventh ACM SIGSOFT Symposium on the Foundations on Software Engineering*, volume 1687 of *LNCS*, pages 127–145, September 1999.

[15] N. G. Leveson, M. P. E. Heimdahl, H. Hildreth, and J. D. Reese. Requirements specification for process-control systems. *IEEE Trans. Softw. Eng.*, 20(9):684–707, 1994.

[16] B. Nuseibeh, J. Kramer, and A. Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *Transactions on Software Engineering*, 20(10):760–773, October 1994.

[17] V. Ratan, K. Partridge, J. Reese, and N. Leveson. Safety analysis tools for requirements specifications. In *Proc. of Eleventh Annual Conference on Computer Assurance*, pages 149–160, 1996.

[18] A. Tribble, D. Lempia, and S. Miller. Software safety analysis of a flight guidance system digital avionics systems. In *IEEE*, 2002.

[19] F. Vahid, S. Narayan, and D. Gajski. Speccharts: A language for system level synthesis. In *Proc. of Computer Hardware Description Languages*, pages 145–154, 1991.

[20] C. Zhou. Hierarchical requirements patterns for systems built using honeywell autocode modules. Technical report, Honeywell Technology center, Minneapolis, MN, 2006.

In the Requirements Lies the Power

Rand Waltzman
Department of Computer Science
Royal Institute of Technology
rand@nada.kth.se

Kristina Winbladh, Thomas A. Alspaugh
Debra J. Richardson
Department of Informatics
Bren School of Information and Computer Sciences
University of California, Irvine
{awinblad,alspaugh,djr}@ics.uci.edu

Abstract

System requirements expressed as scenarios represent a rich source of knowledge about a system and the context in which it is used. This is because the scenarios are the result of extensive collaborative efforts of a wide variety of stakeholders and are in a form to which all can relate. Ideally, they serve to represent the interests of all stakeholders at each stage of the development life cycle. Our focus in this paper is system testing against requirements. In particular, we show (1) how the knowledge represented in scenarios (using ScenarioML) can be directly transformed into an operational knowledge base in a rule-based programming language (JESS), (2) how this knowledge base can be used in system testing to compute, manage, and compare expectations of system behavior to actual system behavior relative to the requirements, and (3) how this can be achieved in a manner that is transparent to all stakeholders. The power of this approach derives from the peculiarly reflective character of knowledge based systems and their explicit use of meta-information and meta-information processing. We demonstrate the viability of our approach by its application to the AquaLush system in which we detected several violations of the system's stated requirements.

1 Introduction

One of the most important goals of software testing is to determine whether a system behaves the way the stakeholders want it to behave. From the stakeholders' point of view, this behavior is ideally described by a set of high-level requirements expressed so that they can be understood with a modicum of effort and without getting bogged down in formalisms that are obscure to all but a small group of specialists. Studies of industry practice [15] have shown that scenarios successfully satisfy these demands on high-level requirements. It is equally important to be able to test these requirements in a way that is transparent to the stakeholders. The stakeholders should be able to look at the test results and see that the system behaves in the way they expect based on the requirements. This demand, however, is not widely being met in practice in industry today. In this paper, we show how to meet this demand by viewing scenario-based requirements as a rich source of knowledge about a system and the context in which it is being used. We use that knowledge source to create a knowledge-based system that tests system behavior directly against the requirements.

The structures that make up a knowledge-based system (KBS) are qualitative models that describe the system's domain [6]. These models describe the domain in terms of causal, compositional, or sub-typical relationships among objects and events. They represent a partition of the world that provides a coherent picture of the domain and permits selective views of the domain for particular purposes. The KBS uses the models as a basis for computing actions in the domain. When we talk of inference in a KBS, we are really referring to a strategy for model manipulation.

The Artificial Intelligence (AI) community recognized early on that the acquisition of qualitative models, the knowledge of a KBS, is extremely difficult and a major bottleneck to construction of such systems [9]. A key observation of this paper is that a set of scenario-based requirements for a system is very close to being the kind of qualitative model required for a KBS that could manage system testing against requirements. A set of requirements scenarios represents the collective and agreed upon understanding and expertise of a wide variety of stakeholders. In the process of formulating scenarios, stakeholders have to hammer out definitions and specifications of domain concepts, terms, events, and processes. In particular, these include causal, compositional, and sub-typical relationships among objects and events of the domain — the same basic elements in the description of the qualitative models that make up a KBS. Since the scenarios explicitly represent the system's behavior, it is reasonably straightforward to translate them into a KBS that models the systems behavior at the same qualitative level. The central type of inference, or model manipulation, done by the KBS in our approach is management and generation of expected system behavior and its comparison with actual system behavior during testing.

The explicit use of meta-information and meta-information processing distinguish knowledge-based systems from most other types of information processing systems and results in the ability of a KBS to reflect upon itself in the sense that the KBS knows what it knows, how it applies that knowledge to a particular problem, and how to explain that application. In our case, that means, for example, that if the KBS detects a mismatch between expected and observed system behavior during system test, it can use its requirements-based qualitative model of system behavior to explain the nature of the mismatch directly in terms of the requirements. This provides the transparency of the testing process previously mentioned. Such an explanatory facility could also be the basis of a tool that allows stakeholders to query the model and perform what-if type experiments in order to interactively explore the requirements.

Because we generate the KBS directly from requirements scenarios, we can do so at an early stage in system development. We can then use the KBS to assist in the design of requirements-based test scenarios. There are several advantages to designing such test scenarios at an early stage of system development. In general, the requirements receive additional validation before requirements problems can cause costly misunderstandings later in the development process [8]. In particular, we could discover (1) requirements that are not testable, (2) behaviors that are not directly observable and that might require the construction of special equipment for observing those behaviors, and (3) conflicting, incomplete, or ambiguous requirements.

In the remainder of this paper, we describe the transformation of the rich knowledge model represented by scenario-based requirements into a KBS that manages the process of testing the system against requirements in a way that is transparent to stakeholders. We show how this technique facilitates stakeholder involvement in two important phases of system development: (1) requirements formulation and evaluation and (2) system testing. We applied our approach to a sample system, AquaLush [7], a project developed elsewhere with a full range of artifacts. AquaLush is an automatic irrigation system that controls irrigation based on soil moisture levels rather than timing. We will use AquaLush artifacts and concepts to demonstrate our approach throughout the paper and validate our approach. In section 2, we summarize related work, and in section 3 we describe the requirements specification format that our approach uses. We present the details of our approach in section 4. In section 5 we briefly describe the results of our validation study and in section 6 we present our conclusions. Finally, we present some thoughts about future work in section 7.

2 Related Work

Testing research has focused mainly on *code-based testing*, in which tests are developed and chosen in order to achieve coverage of the implementation code. Although code-based testing can successfully detect faults in the code, it might not detect faults that produce plausible behavior but fails to meet the system's requirements. Furthermore, code-based testing implicitly excludes stakeholder participation.

Specification-based testing, on the other hand, is a testing technique whose purpose is to confirm the extent to which a system under development meets its specifications. Most specification-based testing approaches have focused on the component level and are typically expressed in the form of Labeled Transition Systems, Finite State Machines, state charts, or message sequence charts [10]. However, as in the case of code-based testing, stakeholders are implicitly excluded from participation as a result of the formal complexity of these representations. Since high-level requirements typically are less formal and more abstract than component specifications, requirements-based testing has been applied with only limited success [5, 11].

The KBS that we describe in this paper can be thought of, at least at some level, as what is known as a *test oracle*. A test oracle consists of two main parts: (1) expected output from the system under test, and (2) a procedure that compares the expected output with the actual output [12]. Oracles can either be human (i.e., manual checking of output) or automated (e.g., software), and although they seem essential to testing, they are often not easy to come by. Software oracles are not used extensively in common industrial practice. However, to the extent that software oracles are discussed, they completely lack the fundamental reflective characteristics of a KBS that are essential for transparent operation. The quality of human oracles and their time and effort is almost never taken into account in the evaluation of testing methods even though human testers are frequently unsure of the correctness of test output and must repeat their work every time the tests are run. This indicates, as recent work also shows, that test oracles can have a significant impact on test effectiveness and efficiency [16]. This finding highlights the importance of our KBS approach.

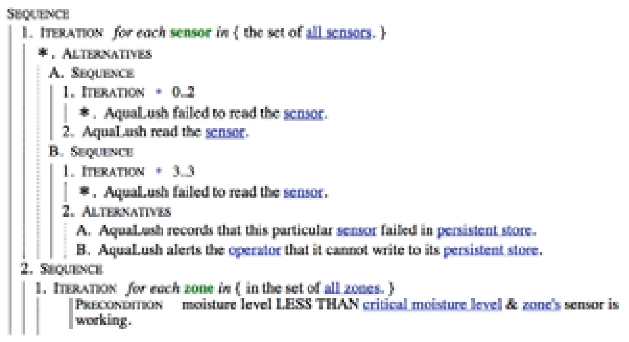
Our view that a set of scenario-based requirements is a rich source of system knowledge and a precursor to a KBS suggests that knowledge acquisition techniques developed over the years in the KBS community should be extremely relevant to the requirements engineering community. Reubenstein and Waters [13], among others, made this same observation in their work on requirements acquisition.

Little work has focused on using a KBS for requirements-based testing. A notable exception is Samson [14]. She suggests that the quality of a test plan can be greatly improved by using a KBS to help match requirement types with test types. Her KBS is not based on the requirements themselves, but rather on a general knowledge of how to map classes of requirements onto classes of tests.

3 Scenarios

A *scenario* is a semi-formal description of uses of a system in terms of situations, interactions between agents, and events unfolding over time. ScenarioML is an XML language for scenarios [3, 4]. ScenarioML expresses scenarios with a combination of events, ontologies, references, and scenario parameters. The events are recursively structured from simple text events as a basis and include compound events grouping several events in a particular order (total or partial), event schemas such as iterated events and sets of alternative events, and episodes that specify another scenario as an event. Allen's interval algebra relations [2] express the temporal relationships among the parts of a compound event. This approach to events supports automated recognition of scenarios happening in a domain, derivation of one scenario from another (such as one or more test scenarios, or paths through a requirements scenario), and other automated processing. Ontologies give a way to describe the kinds of entities that can exist in a domain, define specific entities, and express the relationships among them. We use ontologies to help give the context of scenarios and (through scenario parameters) specify the range of entities that can appear in a specific scenario. We have seen that without ontologies and scenario parameters, it is difficult to derive adequate tests from requirements scenarios because there is no information with which to make them concrete, and there is little opportunity for automation of the process.

Fig. 1 shows a snippet of the "IrrigateScenario" from AquaLush. The scenario shows that the system should try to read each sensor three times, if it fails after three times the sensor is marked as failed. The next sequence in the scenario (Sequence 2) has a pre-condition based on the result of the sensor reads in the top of the scenario.



```
SEQUENCE
  1. ITERATION  for each sensor in { the set of all sensors. }
     *. ALTERNATIVES
        A. SEQUENCE
           1. ITERATION  *  0..2
              *. AquaLush failed to read the sensor.
           2. AquaLush read the sensor.
        B. SEQUENCE
           1. ITERATION  *  3..3
              *. AquaLush failed to read the sensor.
           2. ALTERNATIVES
              A. AquaLush records that this particular sensor failed in persistent store.
              B. AquaLush alerts the operator that it cannot write to its persistent store.
  2. SEQUENCE
     1. ITERATION  for each zone in { in the set of all zones. }
           PRECONDITION    moisture level LESS THAN critical moisture level & zone's sensor is
                           working.
```

Figure 1. Scenario snippet

4 Requirements-Based Test Harness

Our KBS together with its interface to the system under test are components of what we call the test harness. In this section we describe the overall architecture and operation of the test harness.

4.1 Architecture

Fig. 2 shows the architecture of the test harness and its relation to a requirements scenario and its test scenarios. Its basic operation is to drive the system through a variety of paths through the scenarios, compare events output from the system to events in the requirements scenario, and evaluate whether or not they match. If there are mismatches between the expected and received events, the test harness alerts the tester and provides an explanation produced by the KBS. This process is illustrated in Fig. 3.

From the requirements scenario we map the scenario events to input and output functions that will connect the system under test and the test harness. We then use the requirements scenario and the mapping to create several test scenarios, with each test scenario tracing a particular path through the requirements scenario.

The events that we monitor typically involve both the system under test and the testing environment. The events are divided into a system output part, which the test harness should be monitoring occurrences of, and a system input part, which is the response to the output generated by the test harness and that drives the system. The test harness functions that monitor system output and generate system input are collectively referred to as the test harness API. The API is the primary interface between the system under test and the KBS. As an example, consider the scenario event "Failed to read sensor". The system output is "Read sensor", and maps to test harness function `SimSensorDevice.read()`. The KBS decides that the resulting system input is "fail", which maps to the return statement of `SimSensorDevice.read()`.

4.2 Test Scenarios and Test Cases

A *test scenario* corresponds to a particular path through a requirements scenario, and provides information about event responses along that path. A *test case* is a specialization of a test scenario created by adding concrete input data for the system under test. Each test case is an ordered list of event-responses. Each *event-response* contains information about the type of event, particular event parameters, and particular input that the KBS uses to drive the system.

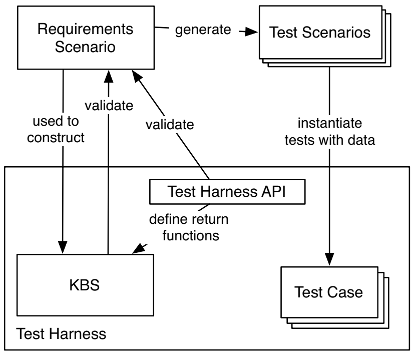


Figure 2. Test Harness Architecture

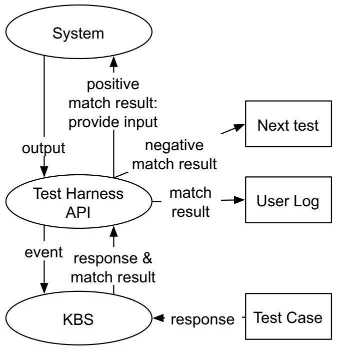


Figure 3. Test execution flow

For example, in the requirements scenario in Fig. 1, a path could be chosen corresponding to the case where the moisture level is below the critical value (see precondition for Sequence 2). The test scenario corresponding to that path contains that information without deciding concretely what the moisture level is. A test case for this test scenario is a further specialization and has concrete input data, such as a particular instance of a sensor and a moisture level of 10% when the critical moisture level is 50%. An event-response element in a test case for this test scenario would therefore look like: (event-response (event-type "Read sensor") (parameter S1) (outcome 10)), where event-type is the output event from the system to the test harness, and outcome is the input data from the harness to the system. Particular objects such as sensor S1 are instantiated instances of the ScenarioML ontology InstanceType element. The order of event-responses is derived from the sequence of events in the test scenario. There could for example, be two "Failed to read sensor" events followed by one "Success to read sensor" event. The KBS keeps track of appropriate input for each event. Fig. 4 shows a test case for the scenario snippet in Fig. 1. Although these test cases are executable code, our experience indicates that they are understandable with a modest effort by non-technical stakeholders.

```
(event-response (event-type "Read sensor") (parameter S1) (outcome fail))
(event-response (event-type "Read sensor") (parameter S1) (outcome fail))
(event-response (event-type "Read sensor") (parameter S1) (outcome 20))
(event-response (event-type "Read sensor") (parameter S2) (outcome fail))
(event-response (event-type "Read sensor") (parameter S2) (outcome fail))
(event-response (event-type "Read sensor") (parameter S2) (outcome fail))
(event-response (event-type "Read sensor") (parameter S3) (outcome 60))
(event-response (event-type "Read sensor") (parameter S4) (outcome 40))
(event-response (event-type "Read sensor") (parameter S4) (outcome fail))
...
```

Figure 4. Test case snippet

4.3 KBS

The KBS performs the following functions: (1) compute expected results based on test case and runtime information, (2) match events coming into the test harness from the system under test to expected events in the test case, and (3) use the information from the event-responses to drive the system.

The KBS is implemented in the rule-based programming language JESS (Java Expert System Shell) [1]. Fig. 5 shows a sample KBS rule. The qualitative model of the KBS is structured around different sets of tasks described by the requirements scenarios. For example, the rule in Fig. 5 partially implements the task of activating a zone for watering. We see this from the task condition pattern on the left hand side (LHS) of the rule. The remaining condition patterns on the LHS of the rule are satisfied if (1) the KBS has received a "Read sensor" event for the zone in question, (2) the number of times the system has attempted to read this sensor is below the maximum number allowed (three according to the requirements scenario), and (3) there exists an event-response in the test case that matches the event and the zone and that has not yet been processed, and whose outcome has the value of 'fail'. If the conditions in the LHS of the rule are satisfied, the KBS takes the actions specified in the right hand side of the rule. These actions are to update the model by (1) incrementing the number of times this sensor was read by 1, (2) noting that the requested sensor read failed, (3) noting that the event has been processed, and (4) noting that the event-response has been processed.

The KBS contains sufficient meta-information to be able to explain its model in a form understandable to stakeholders by (1) producing natural language versions of the rules, and (2) stating the fairly direct relation between the rule components of the model and the requirements in the scenario. As we illustrate in the next section, the KBS is also capable of generating explanations of expectation violations in terms that stakeholders can understand.

```
(defrule process-read-sensor-request-2
       ;System made read sensor request and sensor failed.
       (task (name activate-zone) (object ?zone))
       ?e <- (event (text "Read sensor") (parameter ?zone) (status matched))
       ?nsrwm <- (num-sensor-reads (value ?nsr&:(< ?nsr ?*max-sensor-reads*)))
       (zone (name ?zone))
       ?er <- (event-response (event-type "Read sensor") (parameter ?zone)
             (outcome fail) (status wait))
=>
       (modify ?nsrwm (value (+ ?nsr 1)))
       (bind ?*sensor-read-result* -1)
       (modify ?e (status processed))
       (modify ?er (status done))
)
```

Figure 5. JESS rule

5 Validation Study

In this section we briefly describe our experience constructing the test harness, generating tests, and running the tests on a sample software system, AquaLush [7].

We used the requirements and use cases to produce seven ScenarioML scenarios. We will use one of these scenarios, IrrigateScenario, to describe our work. We chose this scenario because it describes the main functionality of the system and relates it to several stakeholder goals. IrrigateScenario contains two major event sequences where the second one itself consists of three event sequences. The first major event sequence is illustrated in figure 1. From IrrigateScenario we derived 6 test scenarios and created 6 test cases out of the test scenarios. Finally, we ran our test cases against the released version of the system.

5.1 Testing Released Version

We detected a system event mismatch when we ran our first test case on the released version of the system (see test output below). For this test case, the test harness made the system think that sensor S1 failed to read three times in the first event sequence of the scenario. The system read the other sensors and then moved on to open the valves in the zone with sensor S3. Once the valves were open, the system should have started reading sensor S3 every minute until irrigation stopped. What happened instead was that after the valves opened in the zone with sensor S3, the system attempted to read sensor S1, even though this sensor was not in the zone that was being irrigated at the time and had actually been reported as not working.

```
...

An open valve event for V10 was received as
expected.

An unexpected request to read the sensor
from zone S1 was received. It was unexpected
since the system is currently irrigating zone S3.
This action is not in accord with Item 2.1
of IrrigateScenario.

A read sensor event for zone S3 was received as
expected while irrigating.

...
```

The KBS produced a moderately detailed explanation of the expectation violation including a reference to the item in IrrigateScenario that is the basis of the violation. This is a good example of the type of transparent behavior that facilitates stakeholder participation in the testing process.

Running the other test cases revealed additional problems, including some implementation faults, that we do not describe here due to lack of space.

6 Conclusion

We have shown how using the qualitative models of a KBS enables stakeholder participation in requirements-based testing. We have demonstrated the effectiveness of our approach by identifying a number of ways in which the published version of the AquaLush system does not satisfy its requirements. The explanations produced by the KBS comparing the actual system behavior with expected behavior clearly indicate the nature of the requirements violations in each of our six test cases.

The strength of our approach is derived from the synergy of two complementary technologies. On the one hand, we have ScenarioML. It provides a semi-formal language for describing requirements that is particularly well suited to the mechanization of their manipulation. These requirements are a rich source of stakeholder knowledge. On the other hand, we have the qualitative models of KBS technology implemented with the rule-based programming language JESS. These two technologies are deeply connected in that the KBS is directly derived from the ScenarioML requirements. JESS rules are highly modularized components of KBS models. A powerful feature of these rules is their direct relation to requirements scenarios. That is partly where their power lies in terms of directly exposing requirements violations. The data structures used to drive our rule-based programs are directly derived from the ScenarioML representations of the requirements. At this early stage of the research, we derive them by hand. But our experience so far has convinced us that the process of deriving JESS rules from ScenarioML scenarios can be completely automated.

7 Future Work

Our evaluation study has provided us with evidence that our approach is effective in revealing both potential specification problems and implementation faults. However, a testing approach also needs to be efficient. In order to make our approach both effective and efficient, we are currently working on different ways to automate the process.

One substantial task in our process has been to manually write the ScenarioML scenarios. Our group is currently implementing an Eclipse plug-in that will ease this task by providing a graphical interface for editing scenarios. We are also investigating ways to automatically generate test scenarios that cover all branches of a ScenarioML scenario. We will use instantiations of InstanceType elements in the scenario ontology and pre- and post-conditions as constraints and search through the scenario paths with those constraints until we have enough test scenarios to cover all branches and conditions. Since we view scenarios as a knowledge source, we also plan to investigate the possibility of adapting knowledge acquisition techniques from the KBS community. We also plan to auto-generate test cases in the form of JESS event-responses from the test scenarios by either random selection of input data that satisfies the path, manual selection of data, or heuristic selection based on boundary value analysis of the ontology.

As we indicated in the last section, the KBS is currently created manually. In the future we expect to be able to generate it automatically from a set of requirements expressed in ScenarioML. We are aware of the possible need to extend the ScenarioML language to be able to express information needed for this task that is not necessary for the requirements themselves. We also need tools that will allow stakeholders to statically (i.e., not in the context of a running test) inspect and understand the qualitative models of the KBS - especially in terms of the requirements.

Once automation is in place, we will perform a more substantial validation study to evaluate the effectiveness of test cases that are auto-generated from ScenarioML. We also want to evaluate the efficiency of using our approach with automation in place compared to other testing approaches. In order for such an evaluation to be convincing we need to develop a framework for evaluating different testing approaches with regard to effectiveness and efficiency. It is generally known that faults that stem from requirements misunderstandings are expensive to fix late in the development cycle. Time spent on specifying and validating the requirements for the system could therefore be gained by a lower number of severe faults later in the cycle. We intend to develop a framework that classifies the different types of faults an approach can find and measures time spent on specification development and testing all activities.

References

[1] JESS (Java Expert System Shell). `http://herzberg.ca.sandia.gov/jess/`.

[2] J. F. Allen. Maintaining knowledge about temporal intervals. *CACM*, 26(11):832–843, 1983.

[3] T. A. Alspaugh, S. E. Sim, K. Winbladh, M. Diallo, H. Ziv, and D. J. Richardson. The importance of clarity in usable requirements specification formats. Technical Report UCI-ISR-06-14, Inst. for Softw. Res., Univ. of Cal., Irvine, 2006.

[4] T. A. Alspaugh, B. Tomlinson, and E. Baumer. Using social agents to visualize software scenarios. In *SoftVis'06*, pages 87–94, 2006.

[5] L. C. Briand and Y. Labiche. A UML-based approach to system testing. *Softw. and Syst. Mod.*, 1(1):10–42, 2002.

[6] W. J. Clancey. Viewing knowledge bases as qualitative models. Technical Report STAN-CS-8, Stanford University, Computer Science Department, May 1986.

[7] C. Fox. *Introduction to Software Engineering Design: Processes, Principles and Patterns with UML2*. Addison Wesley, 2006.

[8] D. Graham. Requirements and testing: Seven missing-link myths. *IEEE Softw.*, 19(5):15–17, 2002.

[9] F. Hayes-Roth, Waterman, and D. Lenat. *Building Expert Systems*. Addison-Wesley, 1983.

[10] H. Muccini, M. S. Dias, and D. J. Richardson. Systematic testing of software architectures in the C2 style. In *FASE'04*, volume 2984 of *Lect. Notes in Comp. Sci.*, pages 295–309, 2004.

[11] C. Nebut, F. Fleurey, Y. L. Traon, and J. M. Jezequel. Automatic test generation: A use case driven approach. 2006.

[12] T. O. O'Malley, D. J. Richardson, and L. K. Dillon. Efficient specification-based oracles for critical systems. In *CSS'96*, 1996.

[13] H. B. Reubenstein and R. C. Waters. The Requirements Apprentice: Automated assistance for requirements acquisition. *IEEE Trans. on Softw. Eng.*, 17(3):226–240, 1991.

[14] D. Samson. Knowledge-based test planning: Framework for a knowledge-based system to prepare a system test plan from system requirements. *The Journal of Syst. and Softw.*, 20(2):115–124, 1993.

[15] K. Weidenhaupt, K. Pohl, M. Jarke, and P. Haumer. Scenarios in system development: Current practice. *IEEE Softw.*, 15(2):34–45, 1998.

[16] Q. Xie and A. Memon. Designing and comparing automated test oracles for GUI-based software applications. *ACM Trans. on Softw. Eng. and Method.*, 2006.

Data and Process Analyses of Data Warehouse Requirements

Estella Annoni
IRIT-SIG Institute (UMR 5505)
118 Route de Narbonne
F-31062 Toulouse Cedex 9
annoni@irit.fr

Franck Ravat
IRIT-SIG Institute (UMR 5505)
118 Route de Narbonne
F-31062 Toulouse Cedex 9
ravat@irit.fr

Olivier Teste
IRIT-SIG Institute (UMR 5505)
118 Route de Narbonne
F-31062 Toulouse Cedex 9
teste@irit.fr

ABSTRACT

As opposed to classical information systems, Data Warehouses (DW) have existing sources in addition to actor requirements as inputs. DW design methods have to take them both into account. Besides, ETL processes may consume up to 80% of the development time in a DW project and their costs are estimated to be at least one third of the effort. Therefore, ETL and others processes we will define should be handled at the first step of DW engineering. All these processes allow actors to validate relevant and required data. However, DW developpment methods which take into account existing sources do not address these processes at the early stage of process engineering. Thus, they are not represented and validated during the analysis step.

Hence, we provide a model and an original approach to analyse requirements and data sources by specifying data and processes at the analysis stage. We specify DW properties through a graph which simplifies the confrontation between user requirements and existing data sources.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Software Engineering Decision Support

Keywords

data warehouse, analysis, process, method

1. INTRODUCTION

Nowadays, processes which are evaluated in Data Warehouses (DW) projects concerning data extraction, data transformation from existing data sources and loading into DW are called ETL processes. As argued in [8], ETL processes can represent 80% of development time in a Data Warehouse (DW) project.

In I-D6 company [12], we notice that these processes are taken into account after actor requirements analysis which is too late. Moreover, their representations cannot be validated by all DW actors because the model is not close to the user vision of data. In addition, ETL processes are not represented in the same scheme of DW requirements (user requirements and existing data sources). This can explain the 55 % mentioned by Inmon [13] which represents the ETL cost regarding the total cost of DW runtime.

Moreover, based on I-D6 projects, we notice that other processes are required to improve user acceptation and satisfaction of DW. In addition to ETL processes which are related to the DW technical component, these processes are related to the DW decision component. They evaluate the validity, the criticity and more generally the environment of data required to improve and to ensure decision-making process.

However, there are few researches which deal with the modeling processes [7, 23]. Their schemes represent only ETL processes and are only readable by practitioners but not by users. Moreover, the schemes are cluttered with details which disturb user understanding. Above all, they are defined after DW conceptual scheme.

Hence, we provide a model and an approach to analyse DW requirements. The requirements model represents data and processes expressed by actors. It can be evaluated by all of the actors because it is close to user vision of data. In order to simplify the synchronization between user requirements and sources, we propose to represent DW requirements *via* the same model. More specifically, our contribution lies in the proposal of:

1. a model that is close to user vision of data, that handles data and processes validation by all DW actors,
2. an approach of data source analysis which allows designers to handle processes at the early stage of DW development and to define quantitatively candidate multidimensional concepts.

In the following sections, we present progressively our running example. This paper is organized as follows. In section 2, we discuss related work about DW processes. Section 3 presents the model for data source analysis. Then in section 4, we detail how to derive from an E-R schema a model for data source analysis. Finally, Section 5 points out the conclusion and future work.

2. RELATED WORK

2.1 DW development design and processes

Several works deal with DW conceptual design and propose models and approaches to define a scheme of DW. These works can be classified by their approaches in three groups which are those based on demand-driven (take user requirements into account [14, 22]), based on data-driven (take data sources into account [10, 6, 11]) and mixed (take user requirements and data sources into account)approaches.

The latter derive one ideal multidimensional scheme from user requirements and several candidate multidimensional schemes from data sources. The schemes are compared during a step called "Confrontation". This task is very important because it consists in synchronizing the differences between the two schemes. [4] propose to derive several candidate schemes; it is time-consuming. [15] do not derive a schema, they consider directly UML class diagrams of data sources. One specificity of multidimensional model as oppose to classical database ones is de-normalization [21]. Thus, ideal scheme and data source schemes are not comparable. Hence, we propose to derive only one scheme from data sources and we propose to represent user requirements and data source *via* the same model by integrating their respective specificities. These works propose to de termine candidate fact according to user evaluation or number of numeric attributes per entity and relation.

Moreover, most of the methods based on the three approaches are concerned only with the DW static aspect. They represent only data but not the environment around the data, e.g. DW processes.

Concerning processes in DW design, only ETL processes are taken into account. Moreover, a few proposals deal with the specificity of ETL activities. They define mapping between the DW scheme and data source schemes. They are not evaluated during the analysis step but after the DW design schema. The authors of [5] distinguish refreshment processes from ETL processes because the first is dynamic and evolves according to user requirements, whereas the latter, called the "initial loading", defines with respect to current requirements and sources. They do not provide a formal model to specify the activities. We also mention the work of [7], where the authors handle the mapping between sources and DW at logical level. In any case, the problems of ETL processes are not taken into account at the beginning. In [23], the authors provide a model to specify the frequently used ETL activities by using a sui-generis model which allows the definition of attributes'relationships. However, the model is readable by practitioners but not by final users because it is not close to their data vision. The authors of [15, 16] provided a DW method based on UML which takes into account ETL activities at four levels (database, data flow, table and attribute). This method will be heavy in the case of an important project due to the number of levels. The table level, which is not very detailed, gives a view only of data but not of processes. The attribute level gives information about one process (e.g. calculation) but the scheme is unreadable because notes are used instead of UML method specification.

Hence, these methods define DW processes only after DW schema design. Thus, DW users can not validate these processes during the analysis step. Moreover, they tackle only ETL processes which are not sufficient to meet user requirements.

Many tools have been provided to implement them like, Sunopsis [20], OWB [19] and SSIS [18]. These tools allow designers to describe ETL processes during the implementation but not before because they need DW conceptual schemes. All these tools provide a large variety of ETL activities but do not handle these problems upstream from the DW development.

2.2 Context

For these reasons, we provide models and an approach to analyse DW requirements in order to achieve a scheme of data and processes readable for DW users. We distinguish three groups of actors to distinguish decision-makers and to consider designers as actors. Tactical actors are decision-makers who express requirements related to the business area and some related to technical equipment. Strategic actors are decision-makers who express requirements related to company orientation and some requirements about security of the technical equipment. However, system actors are decision-makers and designers who express requirements about the data sources, meaning technical equipment, and some requirements related to the DW decision component. Tactical and strategic actors are DW users.

This proposal is integrated in our complete method based on a mixed approach called Decisional Trident [2]. Our approach is in three steps during an incremental and iterative lifecycle. The first step is composed of three parallel tasks which are analysis of tactical requirements, analysis of strategic requirements and analysis of existing data sources where each results in a derived Decisional Diagram. This step ends with the confrontation task where the tactical decisional diagram, the strategic decisional and the system decisional diagram are compared. Our methods to derive the tactical decisional diagram and the strategic decisional have been proposed in [3]. In order to present our complete method and to tackle specificities of data source analysis, we present in this article system decisional diagram.

The second step transforms the DW decisional diagram into a multidimensional conceptual schema of data and processes. We consider all the multidimensional concepts in order to represent hierarchies, dependences between multidimensional concepts.

The last step is the implementation of the prototype in order to validate the DW design. After deployment, another iteration begins in order to improve the DW.

To validate our proposition, we present an extract of a medical project where we apply it in the I-D6 company [12].

3. SYSTEM DECISIONAL DIAGRAM

In this section, we present our model to represent existing data sources analysis. We develop our running example progressively. From an existing data sources scheme, which is more generally E-R-based, and a property graph we derive a model called System Decisional Diagram (SDD).

A property graph an is oriented graph whose vertex are properties of the DW dynamic aspect. We propose this graph in our previous work [1] to guide user interviews because it is an empirical task. In fact, there is no method and it is not based on a formal support. We represent the properties with an oriented graph without cycles in order to express the relation between the properties and to comment on each vertex. The graph is composed of two subgraphs. One consists in the technical properties group and the other consists in the decision properties group. Both of the subgraphs are composed of non-leaf vertices at two levels. To annotate a graph, we consider each class of properties via a depth first search. We begin with the sub-graph Decision and finish with the sub-graph Technical

Non-leaf vertex of level 2 are classes of properties. We define a coefficient which evaluates the importance of these property classes for a group of actors in the DW development. The coefficient is a number between 1 and 4, where 4 is the highest level. It equals 1 by default. Coefficients are used for the confrontation task in order to match DW requirements. The coefficient indicates the principle of design by contract [17].

Leaf vertices contains an assertion or a sentence in natural language which specifies the property of the project. There is a graph per actor group. Height and order of the property graphs are always the same for each decisional project. Hence, we propose to define ten processes related to management of data and its environment. These annotations yield process parameters. These processes can be divided into 5 groups which deal with extraction-load, error control, data validity, data transformation and data access. All processes are not valid for all actor groups. Thus, we define them per actor groups in the previous work aforementioned. The system property graph of our running example is shown in figure 1

This derivation approach is a rule-based mechanism which allows us to define facts, measures, dimensions, parameters, weak attributes and hierarchy. At the beginning, designers select all relevant data sources related to the application and the environment project. More precisely, they choose sources which concern business areas of the project. We consider that all schemes are in 3-NF as presented in figure 2.

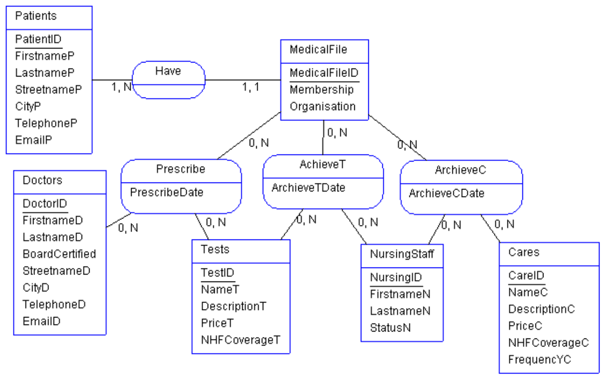


Figure 2: Medical scheme ER

The model, called "System Decisional Diagram", is an extension of the UML class diagram. We use a notation close to [15]'s with only two multidimensional classes. The classes are based on the two main multidimensional concepts well-known by researchers and practitioners which are fact and dimension (cf. figure 3). This model specifies fact (resp. dimension) entity or fact (resp. dimension) relationship as fact (resp. dimension)-classes in a multidimensional way. Data and processes are represented as attributes and methods of theses classes. To indicate that a class or an attribute can be subject to a process, we define the "informativity" concept. The informativity concept is a sign associated to a process before an attribute name to indicate that this attribute participates to this process. We have introduced these concepts of attribute method and informativity in our previous work [3].

4. DATA SOURCES ANALYSIS

We filter the syntactic redundancies of data sources schemes as follows: at first, we prefix with entity or relation name attributes which have the same name in a scheme. We also complete the description of these attributes in the data dictionary. Problems of wrong data selection are avoided with understandable names. This contributes to data quality for a good decision-making. Besides, datatypes of an attribute, any codification, any item or rule must be known for data sources analysis; Thus, they must be specified in property graph. For instance, we prefix the attributes which have the same name like Doctors.FirstnameD, Patients.FirstnameP.

To derive a system decisional diagram from an E-R scheme, we define four sets of rules. Thus, we must apply them in the following order : selection rules, structuring rules, well-formedness rules and merge rules.

4.1 Selection rules

Selection rules allow designers to define candidate facts from E-R scheme. Contrary to the others works [10, 6, 11, 15] where the authors define qualitatively facts we define them according to a quantitative rule by taking into account attribute constraints in addition to numerical attributes. The selection of candidate facts CF$_k$ $\in F$ (F is the set of DW facts) are defined according to table 1. The number of rows (called N_att) of this table is equal to the maximum number of numeric attributes of entities and relationships which are not primary key (PK). We define the arity of an entity or a relationship as the number of relations with other concepts. The number of columns (called N_rel) is equal to the maximum arity of entities and relationships. The N_rel varies between the highest arity and 1. The N_att varies between the highest number of numeric attributes Max(Natt) and 1.

		N_rel		
		3	2	1
N_att	3			Cares
	2	Tests		
	1			

Table 1: Candidate fact table

Cell(n, p) is filled with names of entities and relationships whose its arity equals to p and which have n numeric attributes not PK. We define a function $ArityR : CF \rightarrow CF$ which specifies the set of entities or relationships in relation with the candidate fact CF$_k$. We notice that right facts are in the top part of the table, but we define the following rules to determine formally multidimensional concepts. CF$_k$ with CM$_{CF_k}$ and CD$_{CF_k}$ is a candidate fact CF$_k$ with a candidate measure CM$_{CF_k}$ and a candidate dimension CD$_{CF_k}$. Whereas, f$_k$ with m$_{f_k}$ and d$_{f_k}$ is a right fact f$_k$ with a right measure m$_{f_k}$ and a right dimension d$_{f_k}$.

Selection rules for fact (SF) and measure (SM) selection:

- SF1: All relationships listed in the table are candidate facts CF$_k$. Thus Cares, Tests $\in CF$.
- SF2: If CF$_i$ $\in ArityR(CF_k)$ then CF$_k$ is a f $\in F$, CF$_i$ $\notin F$. Relationships are evaluated in prior,
- SF3: If CF$_k - > CF_i$ (functional dependence) then CF$_k$ is a f $\in F$, CF$_i$ $\notin F$. There is no functional dependence between Cares and Tests. Thus Cares and Tests $\in F$,
- SM1: Numeric attributes of a CF$_k$, which are not PK, are CM$_{CF_k}$,
- SM2: All numeric attributes of a f$_k$, which are not PK, are m$_{CF_k}$ $\in M$. Cares.PriceC, Cares.NHFCoverage, Cares.Frequency, Tests.PriceT, Tests.NHFCoverage are measures,
- SM3: If CF$_i$ $\in ArityR(f_k)$ thus CM$_{CF_k}$ $= \{CM_{CF_i}\} \cup (\{CM_{CF_k}\} \setminus \{CM_{CF_i}\})$. Candidate facts Cares and Tests are not in relation with another candidate fact.

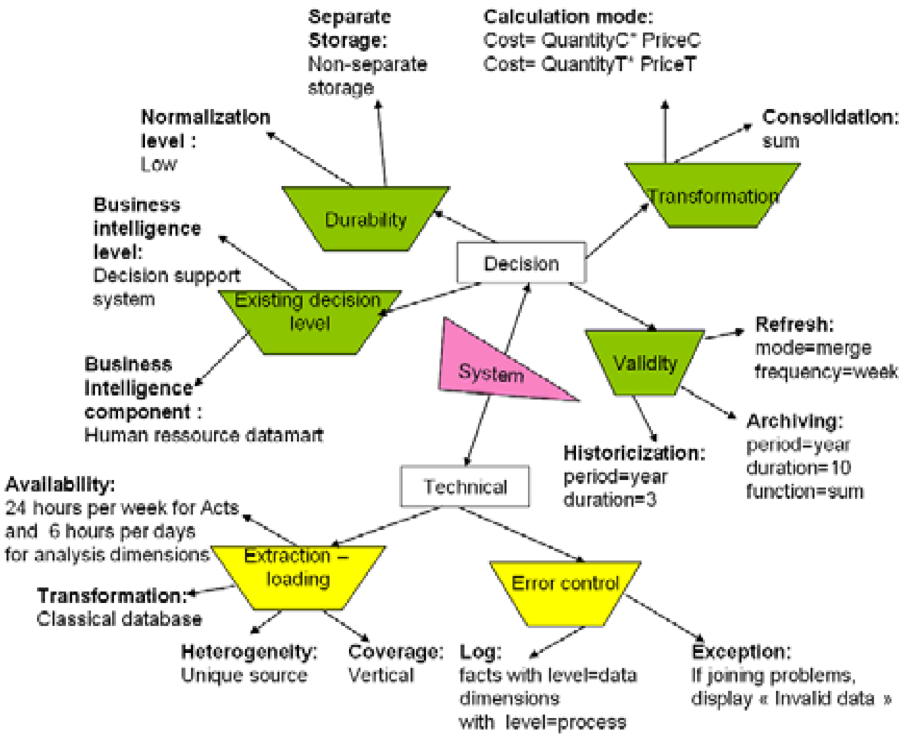

Separate Storage: Non-separate storage
Calculation mode: Cost= QuantityC* PriceC Cost= QuantityT* PriceT
Normalization level : Low
Consolidation: sum
Durability
Transformation
Business intelligence level: Decision support system
Decision
Existing decision level
Business Intelligence component : Human ressource datamart
System
Validity
Refresh: mode=merge frequency=week
Archiving: period=year duration=10 function=sum
Historicization: period=year duration=3
Technical
Availability: 24 hours per week for Acts and 6 hours per days for analysis dimensions
Extraction – loading
Error control
Transformation: Classical database
Heterogeneity: Unique source
Coverage: Vertical
Log: facts with level=data dimensions with level=process
Exception: If joining problems, display « Invalid data »

Figure 1: system property graph

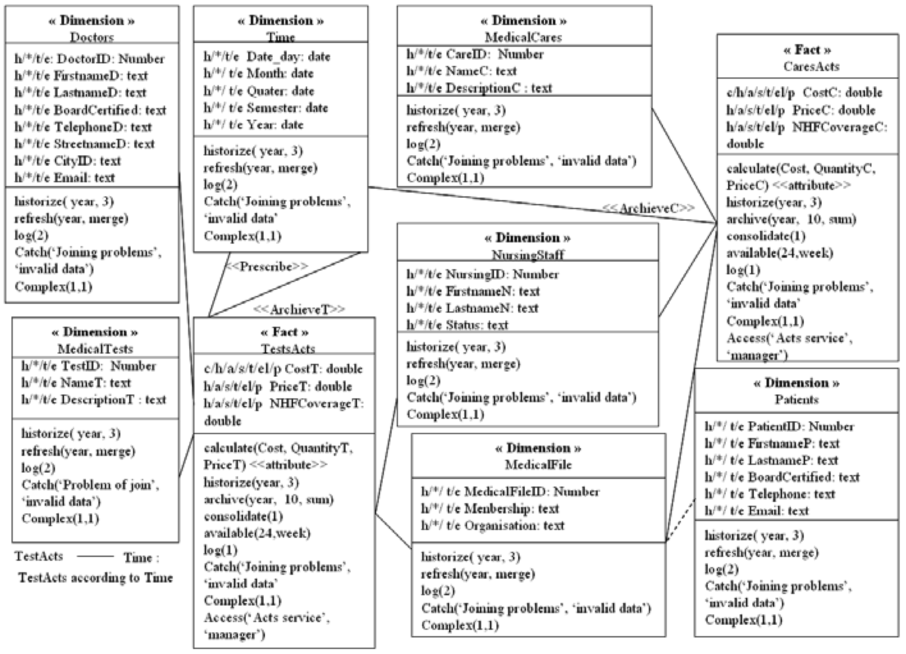

« Dimension » Doctors
h/*/t/e: DoctorID: Number
h/*/t/e FirstnameD: text
h/*/t/e LastnameD: text
h/*/t/e BoardCertified: text
h/*/t/e TelephoneD: text
h/*/t/e StreetnameD: text
h/*/t/e CityID: text
h/*/t/e Email: text
historize(year, 3)
refresh(year, merge)
log(2)
Catch('Joining problems', 'invalid data')
Complex(1,1)
« Dimension » Time
h/*/t/e Date_day: date
h/*/ t/e Month: date
h/*/ t/e Quater: date
h/*/ t/e Semester: date
h/*/ t/e Year: date
historize(year, 3)
refresh(year, merge)
log(2)
Catch('Joining problems', 'invalid data'
Complex(1,1)
« Dimension » MedicalCares
h/*/t/e CareID: Number
h/*/t/e NameC: text
h/*/t/e DescriptionC : text
historize(year, 3)
refresh(year, merge)
log(2)
Catch('Joining problems', 'invalid data')
Complex(1,1)
« Fact » CaresActs
c/h/a/s/t/el/p CostC: double
h/a/s/t/el/p PriceC: double
h/a/s/t/el/p NHFCoverageC: double
calculate(Cost, QuantityC, PriceC) <<attribute>>
historize(year, 3)
archive(year, 10, sum)
consolidate(1)
available(24,week)
log(1)
Catch('Joining problems', 'invalid data'
Complex(1,1)
Access('Acts service', 'manager')
<<ArchieveC>>
<<Prescribe>>
<<ArchieveT>>
« Dimension » NursingStaff
h/*/t/e NursingID: Number
h/*/t/e FirstnameN: text
h/*/t/e LastnameN: text
h/*/t/e Status: text
historize(year, 3)
refresh(year, merge)
log(2)
Catch('Joining problems', 'invalid data')
Complex(1,1)
« Dimension » MedicalTests
h/*/t/e TestID: Number
h/*/t/e NameT: text
h/*/t/e DescriptionT : text
historize(year, 3)
refresh(year, merge)
log(2)
Catch('Problem of join', 'invalid data')
Complex(1,1)
« Fact » TestsActs
c/h/a/s/t/el/p CostT: double
h/a/s/t/el/p PriceT: double
h/a/s/t/el/p NHFCoverageT: double
calculate(Cost, QuantityT, PriceT) <<attribute>>
historize(year, 3)
archive(year, 10, sum)
consolidate(1)
available(24,week)
log(1)
Catch('Joining problems', 'invalid data'
Complex(1,1)
Access('Acts service', 'manager')
« Dimension » Patients
h/*/ t/e PatientID: Number
h/*/ t/e FirstnameP: text
h/*/ t/e LastnameP: text
h/*/ t/e BoardCertified: text
h/*/ t/e Telephone: text
h/*/ t/e Email: text
historize(year, 3)
refresh(year, merge)
log(2)
Catch('Joining problems', 'invalid data')
Complex(1,1)
« Dimension » MedicalFile
h/*/ t/e MedicalFileID: Number
h/*/ t/e Menbership: text
h/*/ t/e Organisation: text
historize(year, 3)
refresh(year, merge)
log(2)
Catch('Joining problems', 'invalid data')
Complex(1,1)
TestActs —— Time : TestActs according to Time

Figure 3: Source decisional diagram

Selection rules for dimension selection:

- SD1: All entities with a 0-N or 1-N relation with a CF$_k$ and which are not in CF or F are CD$_{CF_k}$. Medical file is a CD$_{CF_{department}}$,
- SD2: All entities with a 0-N or 1-N relation with a f$_k$ and which are not in CF or F are d$_{CF_k}$ $\in D$. Thus {Medical file, Nursing staff} $\in d_{Cares}$ and {Medical file, Nursing staff, Doctors} $\in d_{Tests}$,
- SD3: Non-numeric attributes and numeric attributes, which are PK, of a f$_k$ $\in F$ are associated to d$_{f_k}$ $\in D$. For example Cares fact have a PK numeric attribute "CareID" and two non-numeric attributes "NameC" and "DescriptionC". They are related to the same concepts. Thus, only one dimension will be created. We called it MedicalCares. Idem for Tests,
- SD4: All entities connected by a relationship 0-N or 1-N to a d$_k$ and which are not in F is a d$_{d_k}$ $\in D$. This

rule handles hierarchies definition recursively. Medical file is connected to Patients by a relationship 1-N . Thus Medical file $\in d_{Cares}$ and $\in d_{Tests}$,

- SD5: datetime attributes are extracted from facts to create Time dimension. Each fact has n (n=number of datetime attributes) relations with Time dimension. If n>1, the role is specified for each relation. There are two connections between Tests fact and dimension Time (roles Prescribe and AchieveC) and one connection between Cares fact and dimension Time (role AchieveC),
- SD6: if there is not a Time dimension, then a dimension Time $d_t \in D$ is added,
- SD7: check functional dependences between fact and dimension. If there is one, all attributes of a fact, that depend on one or several attributes of a dimension, are moved to this class-dimension,
- SD8 : represent functional dependences between dimension by dotted arrow between them.

Selection rules for parameter (SP) and weak attribute (SW) selection:

- SP1: All numeric attributes of a d_{f_k} are parameters $p_k \in P$. Attribute primary key of each dimension is transformed into a parameter,
- SP2 : attributes of dimension which contribute to calculate attributes of fact are removed for data consistency,
- SW1: All non numeric attributes of a d_{f_k} are weak attributes $a_k \in A$. The other attributes of the dimensions are transformed into weak attributes,

4.2 Structuring rules

Structuring rules allow designers to represent the multidimensional concepts defined above. The information issued at class level is specified through methods of class whereas information issued at attribute level is specified through methods of attribute.

Structuring rules for data:

- SRD1: All facts f_k are transformed into a fact-class with their attributes which are m_{CF_k},
- SRD2: All dimensions d_{f_k} are transformed into dimension-classes with their attributes which are parameters p_k and weak attributes a_k,
- SRD3: All fact-classes f_k are related to their dimension-classes by a dashed line which specifies that fact-class can be analysed according to these dimension-classes,
- SRD4: All dimension-classes d_{d_k} are related to their dimension-classes d_k by a dotted arrow which specifies that dimension-class d_k includes d_{d_k}.

Structuring rules for processes:

- SRP1: method **Available(d, f, c)** with a duration d, a frequency f and a constraint c. For fact Cares, we define Available(24, week, NULL) because the data source required are available 24 hours per week,
- SRP2: method **Complex(t, h)** with the transformations t performed on data sources and h heterogeneity of data sources. Complex(1, 1) because the transformations required are low (level between 1 and 4) and the heterogeneity is also low (level between 1 and 4),
- SRP3: method **Log(l)** at level l. For the dimensions, the method of log is log (2) and for the fact Cares, the method is log(1),
- SRP4: method **Catch(e,m)** to catch error e and display the message m. Catch ('joining problem', 'Invalid data') because designers want to catch this error to evaluate the validity of data,
- SRP6: method **Calculate**$(\{v_i\}^+)$ which indicates how an attribute is calculated from data v_i, Calculate(Cost, Quantity, PriceC)<<attribute>> means the attribute "Cost" is calculated with the parameters Quantity and PriceC. The operation is at attribute level,
- SRP7: method **Historicize(p, d, c)** with a period p for a duration d and a constraint c. Historicize (year, 3) means the attributes are historicized for the three previous years without constraint,
- SRP8: method **Refresh(f, m)** with a frequency f given and a refresh mode m. Refresh(week, merge) means the attributes are refreshed every week in merge mode,
- SRP9 : method **Archive(p, c, fct, cond**) with a period p with a duration d, a constraint c and an aggregate function fct. Archive (year, 10, sum) means the attribute of fact-class Cares is archived for ten years by summing without constraint.

4.3 Well-formedness rules

Well-formedness rules control schema validity. They allow designers to check decisional diagram consistency.

- WR1: A fact-class cannot be related to another fact-class,
- WR2: A dimension-class d_k related to another d_i must satisfy a functional dependency,
- WR3: A dimension-class can be related to a fact-class by a dashed line or to another dimension-class by a dotted line.

4.4 Merge rules

Merge rules allow designers to merge decisional diagram which can be gathered with respect to business areas of the environment of the project. The rules are based on the syntax, e.g. multidimensional objects with the same name are added to the decisional diagram. For instances, we specify that an E-R scheme is associated to a system decisional diagram. Therefore, if there are several SDD, we apply the merge rules according to the project environment to obtain one SDD. SDD can always be merged because there is at least dimension Time in common.

- MR1: Fact-classes with the same name are merged by adding attributes and methods. If parameter values are not the same, we consider according to organization directives the more or the less restrictive values,
- MR2: Dimension-classes with the same name are merged by adding attributes and methods. For different parameter values, we consider according to organization directives the more or the less restrictive values,

- MR3: For dimension-classes and fact-classes with different names, in the case of decisional diagrams of the same type, dimension-classes connected to fact-classes (represented in the two schemes) are added. Fact-classes connected to at least one common dimension of the two schemes are added,
- MR4: For dimension-classes and fact-classes with different names, in the case of decisional diagrams of different types, the application depends on the confrontation type.

5. CONCLUSION

In this paper we provide a model and an approach of existing data source analysis. The model of Decisional Diagram represents the requirements of each actor in a multidimensional way where data and processes are handled. These information are represented closely to the user vision of data and take advantage of specifying methods in the same scheme by UML class diagram. Thus all DW project actors (decision-makers and non IT practitioners) can take part of DW requirements confrontation and can evaluate data and processes of projects.

We also provide property graphs which guide designers in the analysis step, more precisely for user interviews. The coefficient of importance of each DW class of properties and property assertions can be specified. These coefficients simplify the task of confrontation because they indicate which class of property should be integrated in priority. Assertions describe project properties and they are needed for process definition. In our complete approach, the DW conceptual scheme is derived from the DW decisional diagram. Hence, dynamic properties of DW are handled at the early stage of DW development and are represented in the same requirement scheme. Thus, this contributes to reduce the estimated development time regarding to the important runtime rate of DW process. It contributes also to improve decision-maker satisfaction in order to reduce the 80% of projects which do not meet user requiments [9].

Our future work will be devoted to provide a user friendly and understandable approach at logical level. We are also working on a semi-automated or automated approach which we aim to implement in a tool.

6. ACKNOWLEDGMENTS

We would like to thank I-D6 for its collaboration during the four last years. Hence, we can evaluate the feasibility of our complete method and benefits yielded.

7. ADDITIONAL AUTHORS

Additional authors: Gilles Zurfluh, IRIT-SIG Institute (UMR 5505), 118 Route de Narbonne F-31062 Toulouse Cedex 9, email : `zurfluh@irit.fr`

8. REFERENCES

[1] E. Annoni, F. Ravat, and O. Teste. Traitements l'origine des systmes d'information dcisionnels. *Revue des Sciences et Technologies de l'Information*, 11(6), 2006.

[2] E. Annoni, F. Ravat, O. Teste, and G. Zurfluh. Les systèmes d'informations décisionnels : une approche d'analyse et de conception à base de patrons. *revue RSTI srie ISI, Méthodes Avancées de Développement des SI* , 10(6), 2005.

[3] E. Annoni, F. Ravat, O. Teste, and G. Zurfluh. Towards multimensional requirement design. In *Data Warehousing and Knowledge Discovery*, volume Volume 4081/2006, pages 75–84, 2006.

[4] A. Bonifati, F. Cattaneo, S. Ceri, A. Fuggetta, and S. Paraboschi. Designing data marts for data warehouses. *ACM Trans. Softw. Eng. Methodol.*, 10(4):452–483, 2001.

[5] M. Bouzeghoub, F. Fabret, and M. Matulovic-Broqué. Modeling the data warehouse refreshment process as a workflow application. In S. Gatziu, M. A. Jeusfeld, M. Staudt, and Y. Vassiliou, editors, *DMDW*, volume 19 of *CEUR Workshop Proceedings*, page 6. CEUR-WS.org, 1999.

[6] L. Cabibbo and R. Torlone. A logical approach to multidimensional databases. *Lecture Notes in Computer Science*, 1377:155–162, 1998.

[7] D. Calvanese, G. D. Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. A principled approach to data integration and reconciliation in data warehousing. In *Design and Management of Data Warehouses*, page 16, 1999.

[8] M. Demarest. The politics of data warehousing, 1997.

[9] P. Giorgini, S. Rizzi, and M. Garzetti. Goal-oriented requirement analysis for data warehouse design. In I.-Y. Song and J. Trujillo, editors, *DOLAP*, pages 47–56. ACM, 2005.

[10] M. Golfarelli and S. Rizzi. Methodological framework for data warehouse design. In *DOLAP '98, ACM First International Workshop on Data Warehousing and OLAP, November 7, 1998, Bethesda, Maryland, USA, Proceedings*, pages 3–9. ACM, 1998.

[11] B. Husemann, J. Lechtenborger, and G. Vossen. Conceptual data warehouse modeling. In *Design and Management of Data Warehouses*, page 6, 2000.

[12] I-D6. I-d6 is a company specialized in decision-making where estella annoni lead her phd thesis., 2006.

[13] B. Inmon. The data warehouse budget, 1997.

[14] R. Kimball. *The data warehouse toolkit: practical techniques for building dimensional data warehouses*. John Wiley & Sons, Inc., New York, NY, USA, 1996.

[15] S. Luján-Mora and J. Trujillo. A comprehensive method for data warehouse design. In *DMDW*, 2003.

[16] S. Luján-Mora, P. Vassiliadis, and J. Trujillo. Data mapping diagrams for data warehouse design with uml. In P. Atzeni, W. W. Chu, H. Lu, S. Zhou, and T. W. Ling, editors, *ER*, volume 3288 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 2004.

[17] B. Meyer. *Object-Oriented Software Construction*. Prentice Hall PTR., 1997.

[18] Microsoft. Sql server integration services., 2006.

[19] Oracle. Oracle warehouse builder user's guide 10g release 1., 2006.

[20] Sunopsis. Sunopsis etl for data warehousing., 2006.

[21] R. Torlone. Conceptual multidimensional models. In *Multidimensional Databases: Problems and Solutions*, pages 69–90. 2003.

[22] A. Tsois, N. Karayannidis, and T. K. Sellis. Mac: Conceptual data modeling for olap. In D. Theodoratos, J. Hammer, M. A. Jeusfeld, and M. Staudt, editors, *DMDW*, volume 39 of *CEUR Workshop Proceedings*, page 5. CEUR-WS.org, 2001.

[23] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Conceptual modeling for etl processes. In D. Theodoratos, editor, *DOLAP*, pages 14–21. ACM, 2002.

Requirement Analysis Evolution through Patterns

Luca Vetti Tagliati
LucaVT@gmail.com

Roger Johnson
rgj@dcs.bbk.ac.uk

George Roussos
gr@dcs.bbk.ac.uk

Computer Science and Information Systems
Birkbeck University of London

Abstract

This paper presents a strategy, based on requirement patterns (RP), aimed at improving the requirement analysis discipline by allowing business analysts (BA) to produce more reliable SW requirements in a significantly shorter time, minimising the overall requirement risks. In numerous business organisations, IT systems are increasing their strategic significance. In extremely competitive environments, such as investment banking -where this methodology has been tested- modern and advanced IT systems can enable the organisation to obtain and to maintain a predominant position in the market, which in turn results in a greater ROI.

Regrettably a number of academic and industrial studies depict a catastrophic picture about SW projects: most of them are likely to fail and, logically, the probability of failure grows with the size of the project. The project failure factor varies within a range of 50% - 70%. Furthermore, such studies clearly show that requirements is the area where the major risks reside. The proposed strategy is based on the introduction of elegant, well-proven, technology-agnostic, architecturally-compatible, simple and reusable patterns that, focusing on the functional requirements, expand on other requirement analysis artefacts such as domain object model (DOM), business rules (BR), user interface (UI) and glossary.

Keywords: requirement patterns, functional requirements, non-functional requirements, domain object model, business rules, use cases, UML.

1. Introduction

This paper considers the Use Case (UC) formalism as the foundation of the requirement analysis discipline. However, it also provides practitioners with a number of other requirement-related artefacts that are beneficial to BAs regardless of the process and formalism employed.

The RP core consists of UC models, including their specifications, which are linked to other artefacts like DOM, UI design, BR and glossary. Furthermore, this paper demonstrate the opportunity to associate the RPs with artefacts belonging to other models, like system test cases (see Fig. 1). According to Ross Collard [1], UCs and test cases make an effective combination in two ways: when the UCs are complete, accurate and clear, the process of deriving the test case is mechanical. If the UCs are not in good shape, deriving test cases facilitate debugging the UCs. Therefore, while UCs describe in detail the services that the system will have to deliver, the test cases ensure that the system provides these services as agreed.

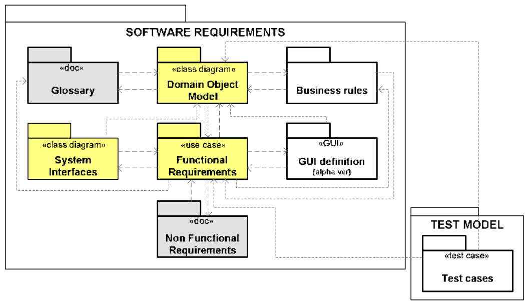


Figure 1. Requirement models relationships

2. Rationale

The requirements patterns concept proposed by this research focuses on the functional requirements modelled through the UC notation as described in the OMG UML specification [2]. This is central to all other artefacts. The behaviour of each UC included in the corresponding diagrams is given in terms of a structured natural language i.e. a template. This allows practitioners to illustrate the sequence of interactions between actors and the systems necessary to achieve the UC goal. Restricting the functional requirements modelling exclusively to UC diagrams and the corresponding specifications would be too rigid not only for the requirements patterns concept but also for ordinary UC models. Therefore, it is necessary to design a mechanism to parameterise the UCs and requirement models to enable their convenient re-use.

This mechanism, consistently with the UC notation, has to:

- present a variable level of formality (it is important to remember that one fundamental audience of UC models is the user community);
- be organised in a core part that cannot be easily changed plus a number of parametric sections whose definition represents the customisation of each single pattern to the specific need.

The solution envisages using the BR document to delegate the definition of the customisable parts, which are directives that differentiate the use of specific requirements. In this way it is possible to define UC scenarios with parametrical sections whose specification is delegated to well-defined entries (paragraphs) included in the business rules document. Therefore, the traditional use of BR is enhanced to include the specification of the parameterisable behaviour.

The adoption of this technique to model UCs presents a number of advantages, independently from the usage of the requirements patterns. For example, it reduces redundancy. Typically, the same BR are referred to by several UCs and by other artefacts (e.g. design model). Therefore, instead of copying and pasting the same BR across a number of different artefacts, with evident problems related to maintainability and traceability, it is possible to refer to the same one stored in the BR document. Furthermore, BR modelling, depending on their nature (constraints, algorithm, etc.), can require different notations. For example, one of the most effective ways of expressing an algorithm is to use the UML activity diagram notation, while some market regulations are better expressed in natural language. Therefore, UC notation is not always the most appropriate tool to express BRs. For the above-mentioned reasons, BRs must be stored in a single artefact and then be referenced by all the others.

The main mechanism for customising the proposed RPs, referred to as a light-weight customisation, consists of specifying the content of the BRs referred to by the requirements themselves or simply accepting the proposed ones. However, this is not the only way as, in fact, it is also possible to change everything else including the UC specification itself (this is a heavy-weight customisation). However, these kinds of changes are pervasive and therefore they should be used only when it is absolutely necessary.

Another advantage of this technique is related to its capability to distinguish, clearly from the source, the part of the requirements that do not change often from the ones that vary more frequently (i.e. sections defined in the business rules document). This should provide development teams with an important input for the design and implementation of reusable business components.

RPs focus on UCs and expand on other models like DOM, UI and glossary. Therefore, when a pattern that provides the solution for a specific requirement refers to well-defined, interrelated business entities (i.e. a portion of the DOM) and/or a UI model, these can also be incorporated in the corresponding requirements model. Furthermore, it is possible to include in the glossary an explanation, given in natural language, of the concepts presented in the class diagrams.

A typical approach for business requirements analysis and documentation consists of focusing first on the services that the system will have to deliver, modelled via the corresponding UCs, and then validating them considering the organisation of the corresponding business entities, modelled via class diagrams. Typically, the definition of these class diagrams requires a review of the UCs. This is the case, for example, where the business entity structure initially assumed in the UC was not fully correct or complete.

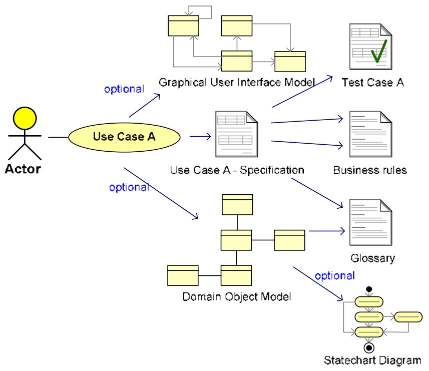


Figure 2. Models of requirement patterns

The DOM is a key artefact for SW development not only in balancing the related UCs, but also in using as an input for the production of other fundamental artefacts like: components design, database design, system messages design and user interfaces. Furthermore, the analysis of a number of business services is better approached using the opposite strategy: defining the business entities' organisation and then consequently modelling the UCs that, manipulating these entities, are able to provide users with requested services. RPs also fully supported this approach: BAs can decide to include in their models specific section of a DOM and then design their own UCs. Therefore, although the RPs idea focuses on UCs, the other models also assume a significant relevance (Figure 2).

The overall RP idea consists of enabling BAs to search through RP collections for specific issues/domain problems and to extract the model required. Each pattern can comprise a number of models:

- one or more UC diagrams;
- a set of UC specifications (templates) which specify the dynamics of each UC present in the diagram mentioned above;
- a number of pre-defined paragraphs to be included in the BRs document whose definition represents the main mechanism to customize the RP. Furthermore, UC specifications can contain topics to be added into the overall glossary;
- a class diagram which models the business entities, including their relationships, referred to by the UC. A textual description of the mentioned entities can be included in the glossary. Some business entities can present a well-defined lifecycle modelled by a corresponding UML statechart diagram which can be included in the RP as well;
- an optional class diagram which models the UI structure including the navigation associated with the services described in the UC;
- a few test cases that describe the test to be performed to verify that the implementation of the specified services are correct and robust.

3. Case study

The following paragraphs discuss a small portion of a case study in order to provide readers with the practical aspect of the proposed theory. In particular, this methodology has been successfully employed in a global investment bank for the development of a security system designed to implement authentication, authorisation and data privacy services. The experiment employed patterns previously designed and extracted from requirements successfully used for similar projects. These included twenty eight UCs, a large DOM, an extensive BRs document, etc. From this large pattern collection, we have selected an example which is related to what is commonly and **incorrectly** perceived to be a simple service: user authentication. This service was selected because: everybody is familiar with it, it allows the presentation of a pattern that includes a number of different diagrams, it is a service that everybody initially would consider extremely simple and straightforward, but a more detailed analysis highlights a number of important aspects that not everybody would think about. In the employed approach, BAs would start from the UC diagram depicted in figure 3.

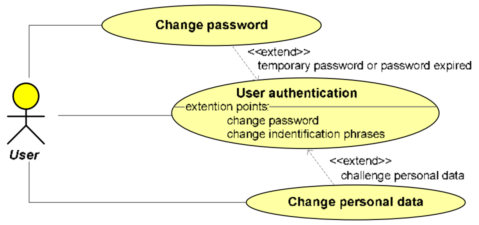


Figure 3. Authentication use case diagram

For each of the presented UCs, the RP, include the corresponding specification. In this paper, due to space limitations, only a small fragment of first one (User Authentication) is presented (see fig. 4).

USE CASE UC: SEC.AUTHENT		**User authentication**	**Date:** 29/Aug/2005
			Version: 0.00.001
Description:		The Log-in service allows users to gain access to the system. The system verifies the credentials inserted by the user and, if these are valid, then the user is authenticated, otherwise the system executes a well-define procedure depending on the number of consecutive failed log-in attempts. The authentication process is only a pre-condition for the execution of the sensitive system's services: authenticated users have to be also authorised.	
User priority:		Medium	
Primary actor:		**User.** This is a generic user (abstract). His/her interest in this use case is to log into the system.	
Preconditions:		The system is available.	
Post-conditions on success:		The system authenticates the user.	
Post-conditions on failure:		The system refuses access to the user and it performs the corresponding management actions.	
Trigger:		The user requests to log into the system.	
		MAIN SCENARIO	
1	System:	Displays the initial "log-in" screen.	**UI:** SEC::LOGIN
2	User:	Specifies the requested credentials.	**BR:** SEC::login_credentials
3	System:	Determines that the user login is valid.	**BR:** SEC::login_validation
4	System:	Determines that the user status is valid.	**BR:** SEC:: login_user_status_validation
5	System:	Verifies that the inserted password matches the corresponding internal one.	**BR:** SEC:: password_matching
6	System:	Verifies that the user's password is in a valid status.	**BR:** SEC::password_status_valdation
7	System:	Determines that the same credentials are not currently in use.	**BR:** SEC::credentials_not_in_use
8	System:	Resets the consecutive unsuccessful logins counter.	
9	System:	Logs the login action into the security audit trail.	
10	System:	Loads the user's profile.	
11	System:	Verifies that the user's profile is valid.	**BR: SEC::**user_profile_status_validation
12	System:	Insert the user' login in the "users current logged in" list. **DOM:** LoggedIn.users.add(current_user)	
13	System:	Shows the user's menu.	
14	System:	The use case ends.	
Alternative Scenario:		**User does not specify the credentials.**	
3.1	System:	Shows an error message.	
3.2	System:	Resumes at point 1.	
Alternative Scenario:		**User login not valid.**	
3.1	System:	Determines that the maximum number of consecutive failed attempts from the same connection has not been reached.	**BR:** SEC::maximum_attempts_connection
3.2	System:	Increases the number of consecutive failed attempts associated with the connection.	
3.3	System:	Shows an error message.	
3.4	System:	Logs the login action into the security audit trail.	
3.5	System:	Resumes at point 1.	
Alternative Scenario:		**Specified password does not match the internal one.**	
5.1	System:	Determines that the maximum number of consecutive failed attempts related to the user has not been reached.	**BR:** SEC:: maximum_attempts_against_user
5.2	System:	Increases the number of consecutive failed attempts associated with the user and the ones associate with the connection.	
5.3	System:	Shows an error message.	
5.4	System:	Logs the login action into the security audit trail.	
5.5	System:	Resumes at point 1.	

Figure 4. Authentication UC specification (Main and some alternative scenarios)

The proposed version is particularly appropriate for enterprise systems. This is because it includes the logic necessary to detect possible intrusion attempts (from a specific location and/or against a precise user) and to check that other users are not currently logged-in with the same credentials. Furthermore, there are extensive controls related to the status of the user, his/her profile and password, which are complex objects with a well-defined cycle of life, etc. The authentication UC specification presents a number of areas where the corresponding default behaviour (i.e. business logic) can be redefined. The definition of the corresponding BRs is the main lightweight mechanism to customize the UC specification. In particular, the proposed UC specification presents the following business rules: login_credentials, login_validation, login_user_status_validation, password_matching, password_ status_valdation, credentials_not_in_use, user_profile_status_ validation,maximum_attempts_connection, maximum_attempts_ against_user, maximum_attempts_against_user. These allow practitioners to define simple behaviour, like the maximum number of consecutive failed attempts allowed from the same remote address, or more complex ones like the conditions that, if verified, force the user to change his/her password.

The default rule states that the user's password has to be changed if its status is temporary, which means that the password has been automatically issued by the system, or its validity time window has expired. These are examples of parts of the service that are subject to change from one implementation to another, and therefore their implementation requires a degree of flexible. Other UCs foresee more complex business rules whose definition is given in term of algorithm modelled by UML activity diagrams, like the calculation necessary to generate unique user id.

The authentication UC specification also includes a number of references to the pre-defined corresponding part of the DOM (figure 5), highlighted by the string "DOM" written in bold. As mentioned before, UCs and DOM describe two different projections of the same "entity", which, in this case, is the authentication service requirement.

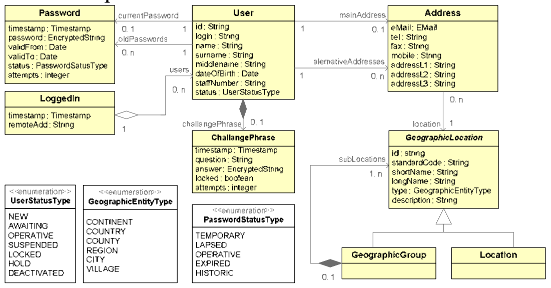


Figure 5. Part of the DOM

The BA, as usual, can decide to integrate the propose sections of the DOM referenced by the UC specification, or to use his/her own. In this case, the most significant entities: user, profile and password, are provided with the corresponding UML statechart diagrams that can be included in the BA model.

Moreover, the UC specification refers to a well-defined user interface (UI:SEC::USER_AUTH) whose object oriented model is a part of the user requirement as well.

Finally, the pattern adds in a number of terms (for example, *Authentication* and *Authorisation*), including the corresponding definition, which can be included in the Glossary document.

4. Requirements patterns categories

The concept of pattern in the SW community has been used with a number of different meanings. The pattern notion considered by this paper presents a high level of compatibility with the original Alexandrian idea [12] that each pattern describes a recurring problem in the particular problem domain including the corresponding solution. This provides practitioners with the possibility of reusing the solutions a number of times without having to study the same problem over and over again [3]. Therefore, the real core idea is to produce a catalogue of elegant, well-proven, extensible and re-usable requirements patterns in the same way that the authors of the book [4] did for the design model. In particular, RPs are reusable, well-proven, architecture friendly and high-quality requirement models for recurrent problems, obtained as result of the experiences from development of real projects. These patterns are provided with the context of their usage, including forces, and they are designed to be customisable by modifying the linked business rules.

From the analysis of real world projects requirements, it has been possible to divide RPs into two main categories: domain specific and general purposes. The former are particularly suitable for specific domains like security, e-commerce, banking, etc. This category presents some similarities with the work of Bjørner [5], related to his studies to formally define the problem domain via a formal mathematical language. The latter are patterns, like data entry, searches, data analysis, and so on. These are typically extracted as a result of the process of reengineering patterns belonging to the previous category. Therefore, the previous category is a first-level application of the patterns present in this set. However, both categories are proper patterns since they provide a well-defined solution to recurrent problems, either domain specific or more general.

RPs can provide practitioners with:

- elegant solutions that not everybody would think of immediately;
- technology and programming language agnostic;
- architecturally compatible and consistent solutions that have been proved through successful implementation in other projects;

- well-proven solutions identified through the analysis of real projects;
- high level of flexibility;
- simple but effective solutions;
- reusable solutions;
- a framework for developing CASE tools;
- a set of superior solutions that can also be very useful for training purposes.

5. Advantages

The RPs adoption produced the following advantages:
time saving. These RPs allowed BA to save time and effort invested in modelling the requirements and therefore they were able to invest more time in the proper requirements analysis and less in the formal aspect of their modelling. This time saved is not only related to the initial production, but it is extended to the number of re-factoring iterations that BAs typically undertake. Often reviewing a model generates a ripple of a number of other models. For example, reviewing a package in the DOM necessitates the review of all UCs that, starting from the described entities, generate a number of services, the UI, the business rules, etc.
higher-level of quality. Analysis gathering discipline proved to be a complicated and particularly critical part of the SW development process. It is not always possible to think ahead about all the different aspects of a requirement (especially for the more complex ones) and as a result, a number of changes can occur that can produce serious consequences of the process outcome (e.g. requirement creep). Furthermore, it is not always possible or affordable to employ a BAs expert in very specific domains, like e.g. in IT security. Therefore, RPs are extremely convenient in these scenarios. Furthermore, RPs explore all scenarios and possible alternatives present in the analysed topic and therefore they do not leave any aspect unexplored, often they present a way of modelling the same requirement that not everybody would think of immediately, they are "implementation friendly" and consistence since they have been identified in previous projects, etc. Finally, each pattern, typically, includes other models like the DOM and the UI model that are often neglected because of a project's time and budget limitations. As proof, the issues tracking system (this project used the software Jira) showed that there was not a single log related to change requirements for the security system. They were all related to fixing and only 10% to the implementation of new services.
risk reduction. This advantage is a direct consequence of the overall quality enhancement described above. Furthermore, since the RPs are well-proven solutions identified through the analysis of real-projects, their feasibility and their ease of implementation are guaranteed. In addition, these patterns provide BAs with important tools for verifying the validity, accuracy and completeness of the requirements specified by users;
time and cost saving. These objectives are the logical consequence of a number of factors. First of all, BAs did not have to model a number of requirements since these were already provided by the patterns. UCs and scenarios can be labour-intensive to capture and document. ([10] and [11]). Furthermore, the requirement models present a high quality level and they are architecture-friendly. The requirements patterns can be raised to a further level by including design model and implementation. In fact, other outputs of the implementation of the security system are reusable design model related to the security RPs. Therefore, the selection of a RP has the potentiality to bring with it models belonging to the design and implementation phases. Finally, their presence allows managers to organize teams where not all business analysts need to be experienced.
standardisation of the business areas. A number of practitioners started realising that the large availability of out-of-the-box components evocated by the *.Net* and *J2EE* architecture has not happened. One of the explanatory factors can be found in the lack of business domains standardisation. This problem can be solved with the RPs, which provide a core with the description of the flows of actions, including the point where the behaviour can vary. Therefore, this should provide development teams with a standardisation that would allow them to produce well-defined and reusable software components. As proof, the development team is investigating the idea of releasing a few icomponents to the open source community.
learning. RPs provide junior BAs with an effective way of improving their technique. Furthermore, given their quality and elegance they allow the less experienced analysis to produce high quality outcome.

6. Related work

The RPs idea is in some ways related to previous works in this area. The most relevant works are:
parameterised UCs introduced by Cockburn [7], where two examples of patterns are discussed. One such pattern, the *"find whatever"*, represents the researching data function, and the second relates to a typical CRUD functionality.
"Patterns for Effective Use Cases" [8], in this case there are differences starting from the patterns notion, which is clearly illustrated by several quotes included in the book. E.g. they propose to consider patterns as

merely a sign of quality, and strategy. They do not consider pattern language as a complete strategy for writing requirements, but as a set of guidelines to support practitioners fill a gap in their knowledge, evaluate UCs quality, etc. Therefore, there is an important divergence from the idea presented in this paper.
"**Use Cases Patterns and Blueprints**" by G. Övergaard and K. Palmkvist [9]
Bjørner's study [5] where the author investigates specific domains (like the railways) with the aim of representing them via a formal mathematical language

The current IT body of knowledge embraces a number of patterns methodologies applied to other disciplines of the software development process, like analysis patterns and design patterns. Although these are extremely interesting, they are out of the scope of this paper.

Although several academic studies and empirical researches present some similarities with this approach, there are also a number of important differences. The most relevant ones shared by all other approaches are:

- some approaches do not consider UCs at all (e.g. [5], [8])
- approaches that focus on UCs are often not fully compliant with the corresponding standard ([7])
- most of the approaches focus on one artefact and do not expand to the wider concept of SW requirements. Either they focus on the functional requirements or on a sort of static view ([5]). Other important requirements artefacts, like DOM and UI, are simply ignored
- only one approach ([9]) tries to makes use of BR but not in a way that would promote reusability
- no single approach includes specific mechanisms for a convenient re-use and customisation of RPs.

Övergaard and Palmkvist ([9]) propose several UC patterns based on a high level of conceptuality that poses a number of problems for their re-use in real projects. E.g., as a matter of comparison, it is possible to analyse their version of the user authentication UC, called log-in. This is unexpectedly integrated with the logoff UC (the same UC encapsulates two completely different and logically opposite services). From the analysis of this UC it is possible to highlight that it is not considered the possibility of fraudulent security attacks, there is not a scenario aimed at locking a user account in case the maximum number of consecutively failed attempts to login has been reached, there are no further checks on the password data, etc. Therefore the reuse of their patterns it is not straightforward. It will likely require the production of a further and more detailed version of the proposed UCs. Finally, the UCs notation is adopted in an unconventional fashion highlighted by the unusual presence of two main scenarios.

7. Conclusion

The overall hypothesis is that the RPs strategy provides practitioners with an effective instrument to produce higher quality requirements analysis more efficiently. This, in turn, produces two major advantages:
project cost reduction: requirements are gathered more rapidly, there are fewer change requirements, etc;
risks reduction. This is achieved because the extracted requirements present a higher-level quality and because the saved time can be invested in more critical activities.

The latter advantage is particularly important since commercial surveys still indicate that the major number of software projects fail because of problems with the requirements stage.

The initial study and corresponding investigation showed a huge success during the requirement phases where UCs and the corresponding DOM were produced by copying patterns from a document. The whole set was produced in only twenty three man days and with virtually no change was requested during the whole process. Furthermore, architects could benefit straightaway from a whole set of requirements that allowed them to design the architecture and the system with no delays.

10. References

[1] R. Collard – "Test Design, Software Testing & Quality Engineering" – July 1999
[2] OMG UML 2 specification
[3] C. Alexander, "A Pattern Language", New York: Oxford University Press, 1977
[4] E. Gamma, R. Helm, R. Johnson, J. Vlissides – "Design patterns, Elements of Reusable Object-Oriented Software" – Addison Wesley, 1994
[5] D. Bjørner: A Cloverleaf of Software Engineering, IEEE SEFM'05
[6] C. Wohlin, K. Henningsson, M. Höst , "Empirical Research Methods in Software Engineering", 1998
[7] A. Cockburn, "Writing effective use cases", Addison-Wesley, September 2000
[8] S. Adolph, P. Bramble, A. Cockburn, A. Pols, in the book "Patterns for Effective Use Cases" Addison-Wesley, 08/2002
[9] G. Övergaard and K. Palmkvist – "Use Cases Patterns and Blueprints" – Addison Wesley – Nov/2004
[10] P.A. Gough, F.T. Fodemski, S.A. Higgings, and S.J. Ray "Scenarios – An Industrial Case Study and Hypermedia Enhancements" Proc. Second IEEE Symposium Requirements Engineering. IEEE Computer Society, pages 10-17. 1995
[11] T. Royer, "Using Scenario-Based Designs to Review User Interface Changes and Enhancements", proc. DIS95: Designing Interactive Systems, Ann Arbor. Pages 236-246 – 1995
[12] C. Alexander, "A Pattern Language", New York: Oxford University Press, 1977

Automatic Generation of Use Case Diagrams from English Specifications Document

Nathalie Rose T. Lim, Christobal T. Cayaba , Joseph Astrophel E. Rodil

***Abstract*— Users appreciate applications that allow human language as a medium of interaction. Inputs expressed in natural language reduce the amount of time needed by a user in learning and conforming to the required format of the application. Computer Automated Use Case Diagram Generator (CAUse) is a system that allows users to automatically generate use case diagrams from English document specifications that can be used for requirements management for business applications. Through the use of Natural Language Understanding concepts, the input specifications document is processed to derive potential actors, use cases, and relationships, thereby generating a use case diagram. The generated diagrams may be edited and saved in the provided diagram editor.**

***Index Terms*—requirements management, use case diagram, natural language understanding**

I. Introduction

Generating diagrams play an important role in software development. These are used to organize and model system behaviors, which are essential for requirements management and communication with end-users [14]. "Requirements management is a systematic approach to eliciting, organizing, and documenting the requirements of the system. It is a process that establishes and maintains agreement between the customer and project team on the changing requirements of the system" [9]. It is identified that the main cause of project failure is mismanaged user requirements. Without proper requirements management, the probability to meet the set objectives of system development teams decreases [9].

In requirements management, the specifications of requirements are given through textual descriptions of the nature or roles of the actors and discussions of the functions related to the actors. There is a style manual to be followed by the writers of such requirements; and this is written in simplified English in declarative form. These requirements are then drawn through the commonly used notation of use case diagrams of the Unified Modeling Language (UML).

Manuscript received February 19, 2007. This work was supported in part by De La Salle University-Manila (DLSU).

N. R. Lim is with the College of Computer Studies of De La Salle University-Manila, Philippines (phone: 632-5240402; fax: 632-5360278; e-mail: limn@dlsu.edu.ph).

C. Cayaba and J. Rodil hold a Bachelor of Science degrees in Computer Science specializing in Software Technology from De La Salle University-Manila (e-mail: chriscayaba@yahoo.com, j_rodil@yahoo.com).

Use case diagrams are used to present the functional aspect of the system. The main elements are the actors and the use cases. Actors are the entities performing the functions of the system. These actors are associated with roles that represent the actual business roles in the system while use cases refer to the business processes or the functions of the system being performed by the entities or actors [2]. Specifically, use case diagrams are used to obtain business processes such as requirements management.

Tool for Requirements Elicitation and Documentation (TRED) is an existing system that facilitates requirements management. It requires inputs entered into a form-based interface. In addition, a certain format of text inputs has to be followed. However, following a format will not allow users to describe and discuss the requirements freely; there will always be unnecessary restrictions in presenting the requirements. The next section discusses a system that generates use case diagrams from English requirements specifications document.

II. The CAUse System

Computer Automated Use Case Diagram Generator (CAUse) automatically generates use case diagrams from English document of business requirements specifications, which will be used for requirements management for business applications.

The system architecture of CAUse consists of three major components namely the Natural Language Analyzer, the Use Case Modeler and the Use Case Diagram Generator.

The Natural Language Analyzer accepts the input document then analyzes the content and generates the candidate actors, use cases, and relationships. These are then passed on to the Use Case Modeler for determining the actors, use cases, and relationships that will be used by the Use Case Diagram Generator to draw the use case diagram (see Figure 1).

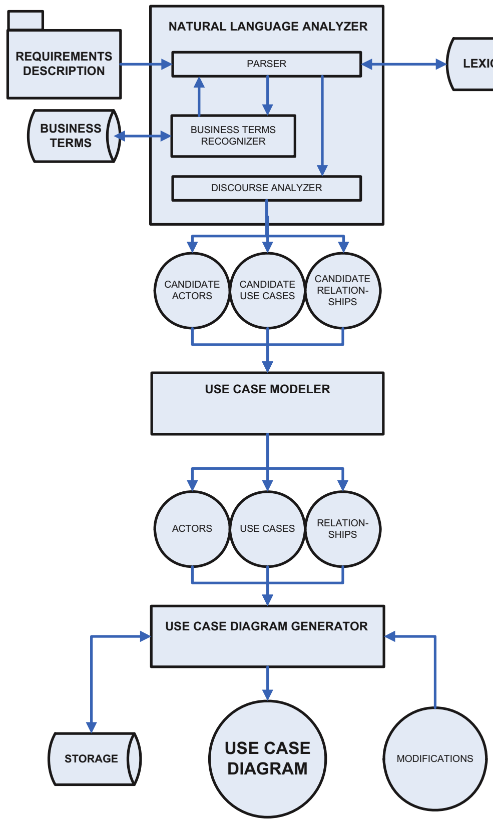


Fig. 1. System Architecture for CAUse

A. *Natural Language Analyzer*

The Natural Language Analyzer has two main processes: parsing and discourse analysis. An existing parser provided by the DLSU (sponsoring institution) was modified to tokenize and tag the part of speech of the words. It is assumed that the input is a business requirements document which is grammatically correct. Initially, the system checks if any of the words from the document matches with any terms from the Business Terms List (stored in a separate file). Then, the remaining words are tagged based on the lexicon used by the parser. The attributes (e.g., human, human action, and item) are used later to generate candidate actors and candidate use cases. After tagging, the discourse analyzer then resolves remaining ambiguities of the requirements specifications input by identifying which words are referred to by pronouns. The resulting information gathered from this phase will now be stored as Candidate Actors, Candidate Use Cases, and Candidate Relationships.

The problem encountered using pronouns is ambiguity as to what the particular pronoun represents. Given the following text, "Chris and Joseph are the entrepreneurs behind GokongCakes, a fast growing cake company. As part of the agreement, he collects daily inventory for analysis.", the pronoun "he" becomes ambiguous because the pronoun "he" could replace two possible antecedents: "Chris" or "Joseph".

In determining the object that is being referred to by a pronoun, the most recent noun that is appropriate for the reference is used. For instance, for pronouns "he" and "she", the preceding reference to a singular noun with an attribute of human is used as replacement. Thus, for ambiguities as described in the text above, the pronoun "he" refers to "Joseph", being the most recent and the most appropriate noun.

For pronouns like "they", the object being referred to may not necessarily be human. In such cases, checking the sentence structures also help in anaphoric resolution. In the following text, "The students borrowed books. They will return them to the library after reading.", "They" is replaced with "students" and "them" is replaced with "books". This is because the sentence structure is that in an active voice. Such types of sentences indicate that the subject (in this case, "They") acts as the doer of the sentence and, as such, a human should be the one performing the action (and not an item). On the other hand, if the text is, "The students borrowed books. They are returned by them to the library after reading.", the "They" refers to "books" and "them" refers to "students" because the sentence is in the passive voice.

B. *Use Case Modeler*

From the Candidate Actors, Candidate Use Cases and the Candidate Relationships generated by the Natural Language Analyzer, the Use Case Modeler processed these and constructs the use case model that is necessary for the generation of the use case diagram. Basic rules used by software engineers are applied to come up with the Actors, Use Cases and Relationships. These basic rules are discussed in Section III.

C. *Use Case Generator*

The Use Case Diagram Generator draws the use case diagram based on the Use Cases, Actors, and the Relationships produced by the Use Case Modeler. The generated diagram may be modified and/or saved for future modification. Modifications applied by the user are reflected on the display and can be stored.

III. UNDERSTANDING ENGLISH TEXT

The extraction process considers the part of speech and the attributes attached to the lexicons.

A. *Candidate Actors*

To extract candidate actors, the system looks for nouns in the sentences. The limitation in CAUse is that the actor should be a human because the nouns are checked if they are of the attribute human. Since a software/hardware component could also be an actor, a new attribute, say "software/hardware", could be made to recognize a candidate actor.

B. *Candidate Use Cases*

To extract use cases from a sentence in active voice, the system looks for verb phrases such that the form starts with a verb (may be followed by other succeeding words) up to a noun acting as its direct object. In the sentence, "The students returned the books to the library.", the use case will be "returned the books".

In a sentence written in the passive voice, the system looks for the word "by". The first noun, that is seen after "by", is determined to be the actor if the noun has an attribute of human. Otherwise, no use case is generated. If the noun is a human, then the system backtracks to check if the word next to the auxiliary verb is a verb from the lexicon. If the part of speech of the word is not a verb, there is also no use case.

C. *Candidate Relationships*

Relationships include association, generalization, extend, and include.

Association. Association is the relationship between an actor and its use case. If the sentence is in active voice, the candidate actor for this type of relationship is the noun prior to the verb. This noun should fall under the human category. The candidate use case is the verb phrase such that the form starts with a verb (may be followed by other succeeding words) up to a noun, which falls under the item category to connote that it is the direct object of the use case. Given the sentence, "The students returned the books to the library.", there is an association between the candidate actor "students" and the candidate use case "returned the books".
In a passive voiced sentence, the system looks for the word "by". Next, the noun after it, which falls under the human category, is considered as the candidate actor. The system then backtracks to check the auxiliary verb and the word next to it. The verb in the sentence is verified if it could be part of the use case (i.e., it should fall under the action category). If it does, it should be part of the candidate use case.

Generalization. Generalization implies similarities in attributes with some entity that exists between actors or between use cases. As for generalization between actors, the keywords "is a", "is", and "are" are considered. For instance, the sentence "Students are users" tells that a generalization relationship exists between "Students" and "users". The words before and after the keywords should both be human nouns before a generalization among candidate actors exist.
For generalization between use cases, given the text, "Dragging a use case is editing the diagram" or "To drag a use case is to edit the diagram", a generalization of use cases from "drag a use case" to "edit the diagram" is identified, since both phrases before the keyword are actions.

<<extend>>. To identify <<extend>> relationships, discourse analysis should be made because an <<extend>> relationship exists when the use cases involved are mentioned more than once in the document. Given the example: "In the enrolment system, the students can pay their tuition fees in cash or check. If they paid in cash, the cashier issues an official receipt. If they paid in check, the cashier issues a provisional receipt", it shows that students can either pay in cash or check and that different receipts are issued depending on the type of payment. To resolve this, discourse analysis should be made. However, this system only resolves pronouns for the discourse. In addition, a semantic analysis is also needed to determine that paying in cash and paying in check are variations of the use case pay. Through semantic analysis, it will be determined that each of the variations of the use case pay will extend a different use case. In the example above, "pay in cash" will have an extend relationship with the use case "issues an official receipt". On the other hand, "pay in check" will have an extend relationship with the use case "issues a provisional receipt". However, the system only implements the use of "respectively" for the semantic analysis. In the sentence: "Users can add and modify classes and calendars and save and edit diagrams and websites respectively", the word "respectively" results to use cases save diagrams and edit websites. A morphological analysis is also needed to determine that two or more use cases may mean the same thing (i.e., "pay in cash" means the same as "paid in cash").
The keywords "may", "can", "after", "before", "when", and "if" are considered in determining extend relationship.

<<include>>. Similar to detecting extend relationships, an include relationship exists only when the use cases involved are mentioned more than once. In the example: "The students can edit the diagram after students opening a diagram… The administrators must open a diagram before they can print the diagram", it specifies that students and administrators can edit and print the diagram but can only do so when they open a diagram. Thus, discourse and morphological analysis is needed. With the limitations similar to determining extend relationships, the system just considers the keyword "must" and repeated use cases in determining include relationships.

IV. Testing Results

The text "The library stores items. Staffs and students are users. Users borrow journals; students borrow books. They return the books to the library after reading." fed into CAUse resulted to the diagram in Figure 2.

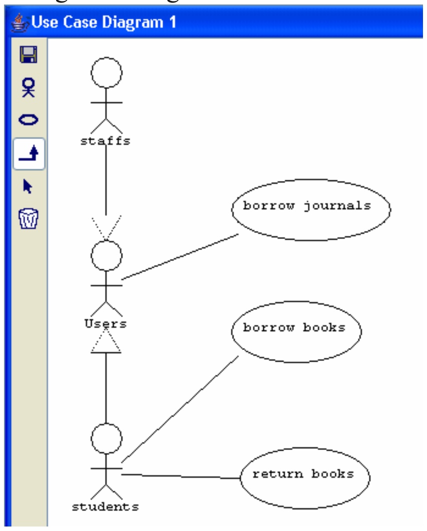


Fig. 2. Sample Use Case Diagram Generated by CAUse

CAUse was able to recognize that the library is not an actor. The extracted relationships were: generalization between the actors "staffs" and "users", generalization between actors "students" and "users", association between "users" and "borrow journals", association between "students" and "borrow books", association between "students" and "return books".

It should be noted that given the same text as an exercise, students who took up Software Engineering came up with a similar diagram. Refer to Figure 3.

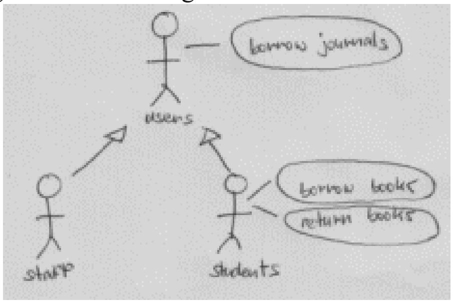


Fig. 3. Sample Use Case Diagram Created by Student

In addition to the exercise above, twenty-four other respondents consisting of computer science students and teachers were initially given six (6) sample inputs. They built expected solutions for each sample and these are then compared to the outputs of the system. The respondents are also allowed to edit the input to further evaluate the system in terms of correctness, consistency, usefulness and acceptability of the output based on the input. From a scale of 1 to 5, where 1 being strongly disagree and 5 being strongly agree, the respondents evaluated several diagrams generated by the system. Figures 4 – 9 show the diagrams that the respondents evaluated.

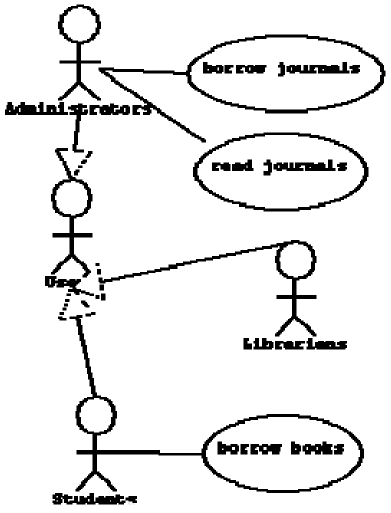


Fig. 4. Generalization relationship between actors for the sample input: "Students and administrators are users. The students borrow books from the library while the administrators borrow and read journals. The librarians are users."

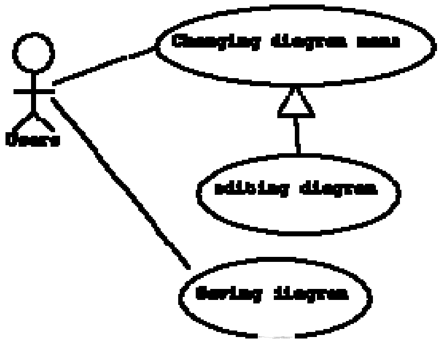


Fig. 5. Generalization relationship of use cases for the sample input: "Changing the diagram name is done by the users. Changing the diagram name is editing the diagram. Saving the diagram is also done by the users."

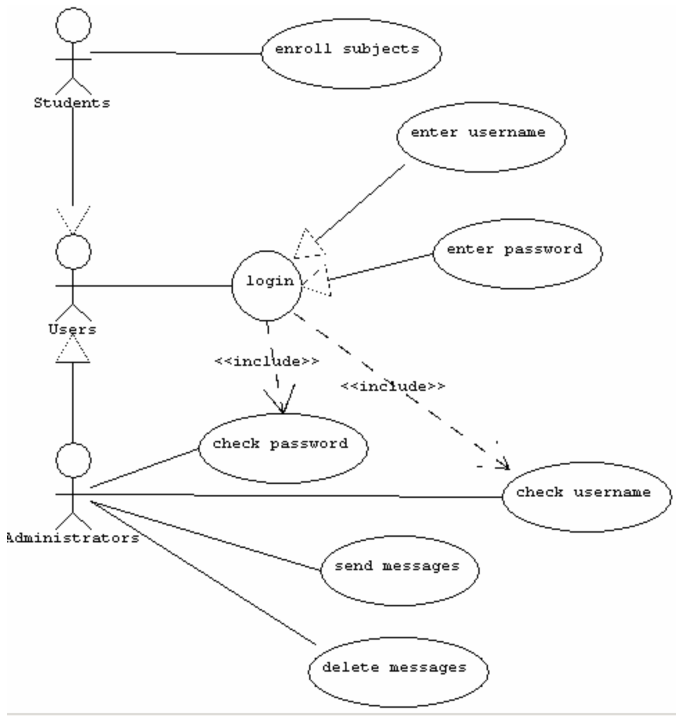


Fig. 6. <<include>> relationship given the sample: "Administrators and students are users. Users can login. To login is to enter username and to enter password. The system must check the password and check the username when users login. The administrators can send messages to other users. Administrators can also delete messages. Administrators can check the password and check the username. Students can enroll subjects."

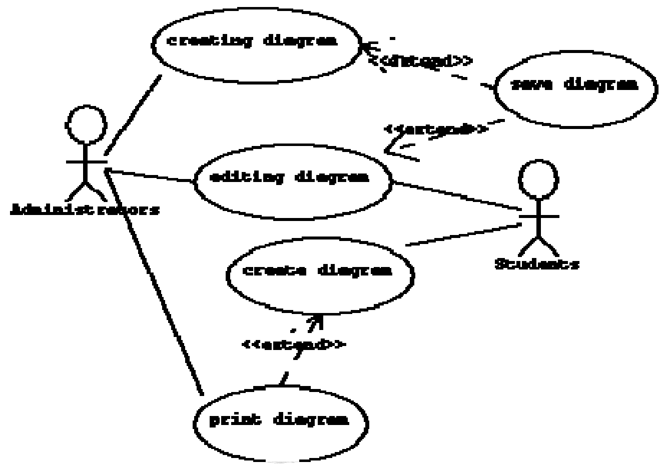


Fig. 7. <<extend>> relationship given the example: "Administrators may save the diagram when editing the diagram and creating the diagram. Students can also save the diagram after editing the diagram. Students can print diagram if they create diagram. Administrators can also print diagram."

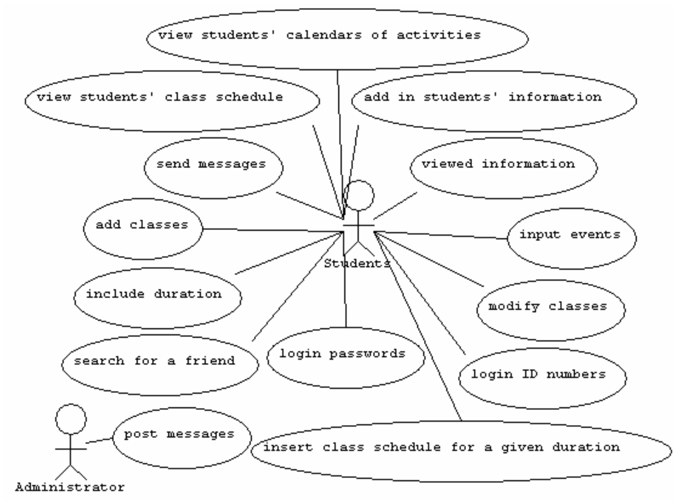


Fig. 8. Use Case Diagram given the sample: "The aim of the project is to specify the requirements and design of a university student online system. Students login through their ID numbers and passwords. Students should be able to view their calendars of activities, view their class schedule and send messages to other students. The system should be based on the Internet.
Students are required to add in their information. This information can be viewed by other students. The students can input events in the calendar and insert the class schedule for a given duration. The students can add and modify classes in their schedules. The students can also include the duration of the class. The students can also search for a friend in the system and send messages to them. Finally, a page for an administrator is provided. The system must have a bulletin board for the administrator to post the messages for all to see. These messages are seen in the home page of all students."

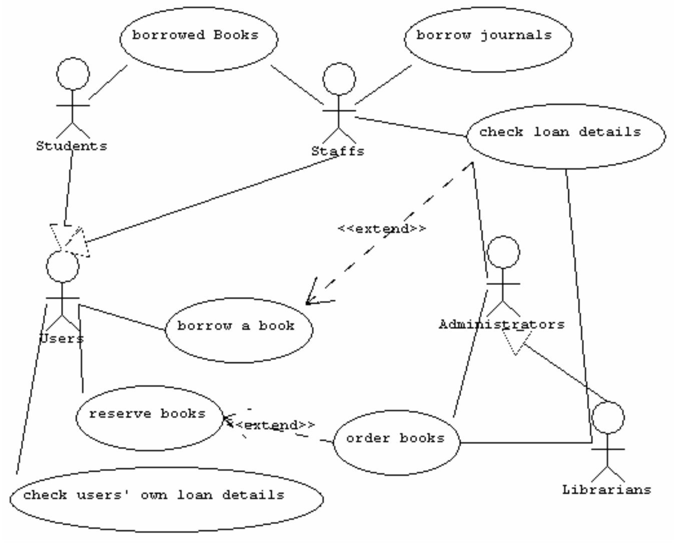


Fig. 9. Use Case Diagram given the sample: "The library stores various items that can be borrowed, including books and journals. Staffs and students are users. The librarian is not a user; the librarians act as the administrators.
Books can be borrowed by both staffs and students but only the staffs can borrow journals. They check loan details when users borrow a book. Administrators can check loan details. Users can check their own loan details at any time. Librarians are permitted to check the loan details of any user. Users of the library can reserve books that are currently out on loan. Librarians order books if the users reserve books. Administrators can order books."

SUMMARY OF TEST RESULT	
Figure 4	4.625
Figure 5	4.542
Figure 6	4.417
Figure7	4.458
Figure 8	4.542
Figure 9	4.000
Overall Rating for using the system to verify te correctness of their use case diagrams	4.458

Interestingly, the result showed that in case the students have forgotten how to create use case diagrams, they confirmed that the system was effective in making them remember and be familiarized on how to create diagrams again. Certain instances were noted where the respondents created incorrect use case diagrams, and through the system, they realized their mistakes and created the correct use case diagrams. For instance, students omitted some functionalities, which the system identified. Because of this, most of the respondents claimed that they were helped by the system generate the correct use case diagrams.
The respondents generalized the system as a useful and helpful tool especially to software engineering students. The system assisted students to come up with a more efficient answer since they can compare the use case diagram they

manually created with the use case diagram CAUse generated. If they did not use the system, they might have overlooked some important components of the use case diagrams in the exercises.

V. Conclusion and Future Work

CAUse would be beneficial to students taking up Software Engineering. The system could serve as a tool for creating the necessary use case diagram for requirements management or as a tool for learning by comparing manually crafted use case diagram with that generated by the system to check for correctness.

Currently, the sentence structures accepted by the parser is limited. Only simple sentences and compound sentences, whose independent clauses follow the same format as the accepted simple sentences, can be fed into the system. The user may choose to rephrase the statements or to use an existing part-of-speech tagger to generate the tags before feeding into CAUse.

A morphological analyzer is recommended to be integrated into the system in order to remove redundancies in the generated use case diagrams due to various morphological forms of words. Expanding the discourse analysis will also help in the optimization of use cases having <<extend>> and <<include>> relationships. Lastly, a deeper semantic analysis is also recommended in order to remove redundancies in the generated use case diagram that exist in synonymous use cases.

Despite the cited recommendations that could improve the system, CAUse can be used to generate use case diagrams that can be used for requirements management for business applications. With the diagram editor, the user could add, delete, and modify the use case diagrams that are initially generated by system.

References

[1] Boman, T. et al. 1996. “Requirements elicitation and documentation using TRED,” in Master's Thesis, UMNAD 159.96, Umeå University, May 1996. Available: http://www.cs.umu.se/~record/T-red/master.html.

[2] Chitnis, M., Tiware P., & Ananthamurthy L. 2004. “Creating Use Case Diagrams”. Available: http://www.developer.com/design/article.php/2109801.

[3] Heyward, R. 1999. “UML Use Case Diagrams: Tips and FAQ”. [online]. Available: http://www.andrew.cmu.edu/course/90-754/umlucdfaq.html (July 13, 2004).

[4] Larman, C. 2002. Applying UML and Patterns. Prentice Hall PTR Prentice-Hall, Inc. Upper Sadler River, NJ 07458.

[5] Leberknight, D. 2003. “Object-Oriented Programming and Design - UML Use Case Diagrams”. [online]. Available: www.softwarefederation.com/csci4448/ courseNotes/09_UMLUseCaseDiagrams.pdf (July 11, 2004).

[6] Le Vie, D. Jr. 2005. “Writing Software Requirements Specifications”. [online]. Available: www.raycomm.com/techwhirl/magazine/writing/softwarerequirementspecs.html. (February 23, 2005).

[7] Lillis, S. 1998. “Use Cases: Adopted by UML and OOSE but not by OMT1.” [online]. Available: http://www.csn.ul.ie/~mrmen/UseCaseReport/UseCaseFinalDocument.html (July 13, 2004).

[8] Mason, O. 2003. “QTag English Tagset.” [online]. Available: http://web.bham.ac.uk/O.Mason/software/tagger/tagset.html. (July 7, 2004).

[9] Oberg, R. 2003. “Applying Requirements Management with Use Cases”. [online]. Available: http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/apprmuc.pdf.

[10] Object Management Group. 2004. “Introduction to UML.” [online]. Available: http://www.omg.org/gettingstarted/what_is_uml.htm. (March 29, 2004).

[11] Object Management Group. 2004. “Unified Modeling Language (UML) 1.5.” [online]. Available: http://www.omg.org/technology/documents/formal/uml.htm (June 12, 2004).

[12] Probasco, L. and Leffingwell, D. 1999. “Combining Software Requirements Specifications with Use-Case Modeling.” [online]. Available: http://www.spc.ca/downloads/srs_usecase.doc. (February 23, 2005).

[13] Rumbaugh, J., Blaha, M., Premerlani W., Eddy F., Lorensen W. 1991. “Object-Oriented Modeling and Design”. New Jersey: Prentice Hall.

[14] Vinciguerra, R. 2004. “UML Diagram Types”. [online]. Available: http://www.vinci.org/uml/diags.html.

REM4j - A framework for measuring the reverse engineering capability of UML CASE tools

Steven Kearney and James F. Power
Dept. of Computer Science, National University of Ireland, Maynooth, Co. Kildare, Ireland.

Abstract

Reverse Engineering is becoming increasingly important in the software development world today as many organizations are battling to understand and maintain old legacy systems. Today's software engineers have inherited these legacy systems which they may know little about yet have to maintain, extend and improve. Currently there is no framework or strategy that an organisation can use to determine which UML CASE tool to use. This paper sets down such a framework, to allow organisations to base their tool choice on this reliable framework.

We present the REM4j tool, an automated tool, for benchmarking UML CASE tools, we then use REM4j to carry out one such evaluation with eleven UML CASE tools. This framework allows us to reach a conclusion as to which is the most accurate and reliable UML CASE tool.

1. Introduction

Many UML CASE tools provide the ability to reverse engineer UML diagrams from source code, and these diagrams can be essential to software maintainers in understanding the design of a system.

Since there are many UML CASE tools available, the question many organisations face is: *Which one suits our needs best?* To answer this question they will need to evaluate all the available tools, measure the results of this evaluation and rank the tools based on the evaluation.

This paper establishes a framework for benchmarking and evaluating UML CASE tools. We describe the construction of the framework, and its use on an oracle program, designed to expose inaccuracies in reverse engineering. We then examine the impact these inaccuracies have by running the UML tools over a suite of real-world programs, and examining the variance in reported metrics.

2. Background and Related Work

Often legacy systems have an originally convoluted design, obsolete documentation, and the original developers may have left the company. Software may have numerous patches and fixes applied over time. It can be an arduous task to understand a legacy system [11, 12]. *Re-engineering* is the examination of a subject system to reconstitute it in a new form and the subsequent implementation of the new form [2]. *Reverse Engineering* is the process of analyzing a subject system to create representations of the system in another form or at a higher level of abstraction [2, 16, 9, 1].

There has been much research carried out that investigates tools and techniques for reverse engineering. Notable examples include the RIGI toolset for reverse engineering [15], the Dali Workbench [8], CPPX [4] and Columbus/CAN [6]. More recently, reverse engineering tools have begun to use the diagrams of the Unified Modeling Language (UML) as a representation for reverse-engineered artifacts. One of the major challenges faced by designers of reverse engineering tools is the struggle to keep with continuously evolving UML versions, as well as version of the associated XML Metadata Interchange (XMI) [10].

Our approach exploits object-oriented software metrics to provide us with a means to collect information about the characteristics of a Java application [13, 14]. These characteristics are important since if we reverse engineer a Java application we would expect the characteristics exported in the XMI file to be an accurate reflection of the application.

While Cooper et al. studied the inaccuracies that occur from Forward Engineering vs. Reverse Engineering, they stopped short at evaluation the reliability of the tool to export XMI [3]. Jiang and Systä explored the differences in Exchange Formats between UML CASE tools and they investigated if UML CASE tools delivered on the OMG ideal of interchangeable XMI files, but they stopped short of automating the process[7].

Our approach, centered on the REM4j framework is unique in that it can automate the reverse engineering, metric capture and evaluation of metrics. We first calibrate the process on a specially-designed *oracle* program, and then use peer-evaluation to investigate the performance of tools on a suite of real-world Java programs.

3. Experimental Setup

REM4j (*Reverse Engineered Metrics 4 Java*) takes Java

Tool	Vendor	Version	Abbrv
ArgoUML	Tigris	0.22	AR
MagicDraw	Magicdraw	12.0	MD
Bouml	Bouml	2.17	BO
Metamill	Metamill	4.2	MM
Visual Paradigm	Visual Paradigm	3.1	VP
Jude	Change Vision	Prof 6.0	JU
Enterprise Architect	Sparx Systems	6.5	EA
UModel	Altova	2006 r.2	UM
ESS-Model	Ess-Model	2.2	ES
Ideogramic UML	Ideogramic	2.3.3	IC
Poseidon for UML	Gentleware	4.2	PO

Table 1. The 11 tools we have chosen for our study. The final column gives an abbreviation for each tool that we use in later tables.

source code as its input, and reverse engineers a class diagram from the code use a pre-recorded set of macros for a a particular UML CASE tool. The result is exported in XMI format and then run through a commercially available metric calculation engine, after which REM4j colllates all the results and saves them to a CSV file.

The first step in this experiment was to choose a set of UML CASE tools for the evaluation. We tried to gather as wide a range of tools as possible, with the only selection criterion being their ability to reverse engineer Java source code back to UML class diagrams, and to export the diagrams in XMI format. These tools are listed in Table 1.

Our REM4j framework is designed to provide a modular environment within which these UML tools can be run and their output data analysed. The framework needs to be able to open a UML CASE tool, import Java source code, reverse engineer it, export it to XMI, pipe the XMI into a metric calculation engine and gather and collate the results into one readable CSV file. REM4j uses a number of third-party applications, and serves to coordinate the interaction between these tools.

At the center of the framework is the *SDMetrics* tool [17], which is a powerful commercial application that is capable of analysing XMI and computing metrics based on that XMI. Automation is achieved using *Autohotkey*, a macro utility for Microsoft Windows that has the ability to record keystrokes and mouse clicks. Autohotkey provides the ability to write a macro for a particular CASE tool and then compile it to an executable. Finally, *Chart2D* is an open-source charting class library, and is used by REM4j to visualise its results.

REM4j takes two inputs when starting: the directories of the Java source files you would like to reverse engineer, and the AutoHotKey directory. If for example three source code directories are selected and then four UML tools are selected, the reverse engineering loop would execute twelve times. Each source code directory is reverse engineered and exported to XMI and then piped into the metric calculation engine tool. When the REM4j automation tool has finished executing it generates its results in both text and graphical formats.

4. Exploratory Analysis using an Oracle

We chose the term *Oracle* to describe a piece of Java source code for which all its characteristics, elements and attributes were known. The *Oracle* application was designed and written explicitly for this paper. In particular, all of the metric values were calculated in advance and the code was constructed to have as many different metric scenarios as was feasible.

The *Oracle* application was written with a 0-1-2 (ZOT) metric policy in place. For example, the *Oracle* application had at least one class with no *Public Methods*, at least one class with exactly one *Public Method* and at least one class with more than one *Public Method*.

Figure 1 shows a Class Diagram which illustrates the structure and design of the *Oracle* application. On this diagram, overlapping classes are inner classes, dotted lines represent an *implements* relationship and a solid line represents an *extends* relationship.

In the next three sections we will break down the metrics that this paper investigates into *Size Metrics* and *Inheritance Metrics*. The SDMetrics tool also reports on *Coupling Metrics* but, since these must be evaluated on a per-class basis, we have not considered them further here.

4.1. Size Metrics

Size metrics measure the size of design elements, this is simply a count of the elements that are contained within an application.

• **Number of Variables (NoV)** The NoV metric refers to sum of the number of variables in all classes regardless of type, visibility, changeability or scope. It does not count inherited variables, or variables that are members of an association [13].

As shown in the Class Diagram the *Oracle* application clearly has 12 variables or attributes. However 4 of the 11 tools produced a figure other than 12. Both Jude and Bouml had the lowest figure as they reported the *Oracle* application having only 7 variables, while Poseidon reported 8 and Ideogramic UML reported 9.

• **Number of Methods (NoM)** The NoM metric has a value that is the sum of the number of all methods in all classes regardless of type, visibility, changeability or scope. It does not count inherited methods, but it does count abstract methods [13].

The *Oracle* code contains exactly 23 Methods, so any derivation from this figure would suggest an inaccurate tool. All UML CASE tools with the exception of Bouml which reported 20, produced the correct figure of 23.

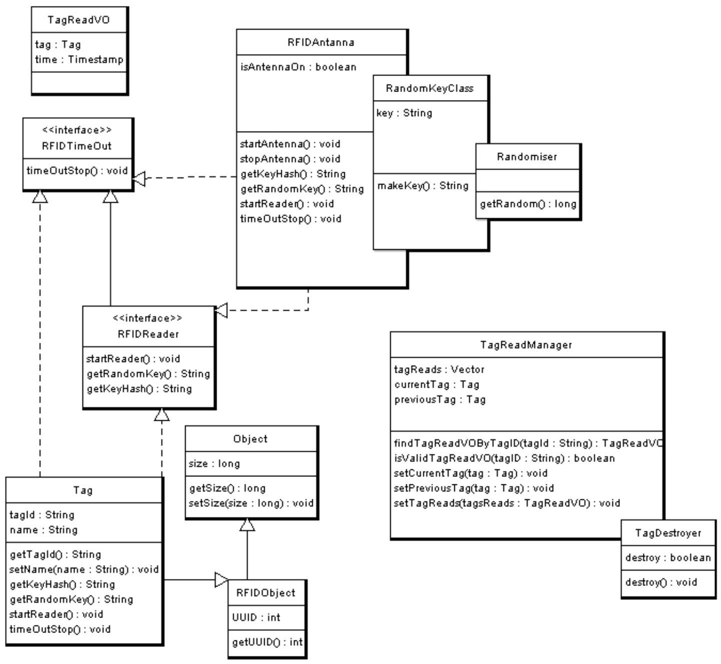


Figure 1. This Class Diagram shows the 11 classes of the *Oracle* application.

• **Number of Public Methods (NoPM)** The $NoPM$ metric has a value which is the sum of the number of methods in all classes that have public visibility [13].

The *Oracle* application was written with exactly 20 public methods, all of the tools agreed with this figure except one, Bouml, which reported 18. This is not surprising as Bouml only reported a total of 20 methods when there was actually 23 methods in the application.

• **Number of Setters (NoS)** The NoS metric counts any Method that begins with 'set'. The *Oracle* application contains 5 such methods, and all tools produced a NoS value of 5.

• **Number of Getters (NoG)** This metric counts any Method that begins with 'get', 'is' or 'has'. The *Oracle* application contains 9 such methods, and all tools produced a NoG value of 9.

• **Inner Classes (IC)** This metric will return the total number of inner classes nested within an application. The Oracle application contains the class *RFIDAntenna* which has an inner class nesting level of 2. The *RandomKey* class has a inner class nesting of 1, the *TagReadManager* also has a nesting of 1. These are summed to produce a value of 4. Ideogramic UML, Bouml, Enterprise Architect and ESSModel reported 0 inner classes, while MagicDraw UML reported 7. ArgoUML, Metamill, Poesideon, Visual Paradigm and Jude correctly reported 4.

• **Total Number of Classes (ToC)** This is a count of the total amount of classes that the UML CASE tool exported in its XMI document. The class diagram in Figure 1 clearly shows 11 classes however none of the UML CASE tools reported this figure, they all reported figures of between 9 and 53. Table 2 shows how Jude reported 42 classes above the actual of 11 classes.

This error has most likely occurred due to the fact that different tools treat imported packages differently, such as the java.util.String class, while not part of the *Oracle* application it is used by it. Some tools count classes like the java.util.String class in the ToC metric, Thus making the metric unreliable. The ToC metric will not be evaluated further.

Size Metrics Summary As we know the correct value for all the size metrics, we can state if the tools passed or failed.

As Table 2 shows, Argo UML, Metamill and Visual Paradigm were the only tools to be correct on all evaluated metrics. The table displays the actual metric value, if a tool reported an incorrect value, the difference between the reported and the correct value is displayed.

4.2. Inheritance Metrics

Inheritance Metrics deal with polymorphism, depth and width of the Inheritance tree, the number of ancestors or descendants of a class.

• **Interfaces Implemented (II)**

The total number of interfaces that are implemented within an application. The *Oracle* application contains two interfaces, both of which are implemented by *RFIDAntenna* and *Tag*. As both interfaces are implemented twice, the correct value for the II metric is 4.

	NoV	NoM	NoPM	NoS	NoG	IC	ToC
Actual	*12*	*23*	*20*	*5*	*9*	*4*	*11*
AR	0	0	0	0	0	0	+2
MD	0	0	0	0	0	+3	+4
BO	-5	-3	-2	0	0	-4	-5
MM	0	0	0	0	0	0	-3
VP	0	0	0	0	0	0	-2
JU	-5	0	0	0	0	0	+42
EA	0	0	0	0	0	-4	+1
UM	0	0	0	0	0	-4	+5
ES	0	0	0	0	0	-4	+4
IC	-3	0	0	0	0	-4	-2
PO	-4	0	0	0	0	0	+2

Table 2. Size Metrics Results: For each size metric on the top row and each UML tool in the left column, this table shows the difference between the actual value and the value calculated.

	II	NoC	IT	CLD	MI	VI
Actual	*4*	*2*	*3*	*3*	*5*	*2*
AR	-4	0	0	0	0	0
MD	0	0	0	0	0	0
BO	-4	0	0	0	0	0
MM	-4	0	0	0	0	0
VP	0	0	0	0	0	0
JU	-4	0	0	0	0	0
EA	0	+1	+1	+1	0	0
UM	0	0	0	0	0	0
ES	0	0	0	0	0	0
IC	-4	0	0	0	0	0
PO	0	0	0	0	0	0

Table 3. Inheritance Metrics Results: For each inheritance metric on the top row and each UML tool in the left column, this table shows the difference between the actual value and the value calculated.

Ideogramic, ArgoUML, Jude, Metamill and Bouml returned 0 as the metric value, all other tools agreed and returned 4.

• **Number of Children (NoC)** This measures the number of immediate subclasses subordinated to a class in the class hierarchy.

In the *Oracle* application, two classes, *Object* and *RFIDObject* are extended by another class, *RFIDObject* extends *Object* and *Tag* extends *RFIDObject*. The value of NoC for the *Oracle* application is 2. All of the UML CASE tools agreed that it was 2, with the one exception of Enterprise Architect, which stated that it was 3.

• **Inheritance Tree (IT)** This metric represents the sum of the ancestors or descendants of each class within the application. The *Oracle* application has two instances where a class has a inheritance depth of greater than 0, this is the *Tag* class which has a depth of 2 and the *RFIDObject* class which has a depth of 1. Hence the total IT value is 3.

With the exception of Enterprise Architect which reported 4, all tools agreed on 3 being the correct metric value.

• **Class to Leaf Depth (CLD)** With this metric we calculate the longest path from a class to a leaf in the inheritance hierarchy. For example the *Object* class has a depth of 2, the distance from its furthest leaf, the *Tag* class. Whereas the *RFIDObject* has a depth of 1 to reach its furthest leaf. These distances are then summed for each class in the application giving a total CLD value of 3.

All of the UML CASE tools agreed that the CLD was 3 with one exception again, the Enterprise Architect tool.

• **Methods Inherited (MI)** This metric is the sum of the number of methods inherited by each class. This is calculated as the sum of *Number of Methods* taken over all ancestor classes of the class [13, 17].

The *RFIDObject* class in the *Oracle* application extends the *Object* class, therefore it inherits the two methods in the *Object* class. The *Tag* class extends the *RFIDObject* class, so it inherits the one method in the *RFIDObject* class and it also inherits the two methods from the *Object* class. The MI metric value is thus 5, and all of the tools reported this value.

• **Variables Inherited (VI)** This is the sum of the number of variables each class in the application has inherited. It works in much the same manner as MI. The *Oracle* application produces a metric value of 3 and all the UML CASE tools agree on this figure.

Inheritance Metrics Summary As the table, 3 shows, the UML CASE tools were in agreement most of the time, with the exception of Enterprise Architect, which frequently over rated the metric value by 1.

5. Analysis of Real World Programs

While the results for the Oracle class diagram show some differences in the metric values over the UML tools, it is not clear whether these are significant. In particular, it is important to know how the differences reflected in the previous section impact the analysis of larger programs.

To this end, we have assembled a test suite of "real-world" Java programs, and used the REM4j framework to extract class diagrams and calculate metrics. In this section we examine the results of comparing the accuracy of the eleven UML tools.

5.1. Control Charts

When using real-world programs, we have no oracular value for the metrics, and so we use peer evaluation to rank the UML tools. That is, we assume the collective judgement produced by the tools to be a “correct” answer, and judge each tool in this context. To this end, we adopt the *control chart* or *Shewhart chart*, which is a chart with five horizontal lines running across it. We use the threshold values of Lanza and Marinescu [14], and define the *center line* as a middle line reflecting the mean value of the metric. The *upper* and *lower control limits* are then 1.5 standard deviations above or below this line, and define the boundaries of “trustworthy” results. A wider threshold is provided by the *upper* and *lower warning limits*, which are 1 standard deviation above or below the center line, and flag values that require further investigation.

5.2. Java Application Selection

A selection of Java applications was chosen for this evaluation. Some were drawn from the DaCapo benchmark suite [5] while other applications were chosen from sourceforge.net for diversity. The programs are:

antlr	generates a parser and lexical analyzer
eje	a simple editor for Java
fop	renders pages to a specified output e.g. PDF
hsqldb	a database written purely in Java
jameleon	an automated test framework written in Java
java2d	is a java 2d graphics package
jolden	a benchmark test application written in Java
junit	a well-known framework to write repeatable tests
pcj	a set of collection classes for primitive data types
pmd	analyzes Java source code for potential problems
xalan	an XSLT processor for transforming XML

We refer to these Java applications collectively as the *test suite* from now on.

5.3. Metric Capture

Table 4 summarises the pass/fail results for each tool when run over the test suite. The top row in Table 4 lists the 11 reverse engineering tools under study, and the leftmost column lists the 11 Java applications in the test suite. Each cell records either “P” for pass or “F” for fail, indicating whether or not the UML tool exported valid XMI for this benchmark program. For example, we can see that the Argo tool, represented by the column labelled “AR”, exported valid XMI for all benchmark programs other than `hsqldb` and `pcj`.

The bottom row of Table 4 summarises the results for each Reverse Engineering tool, by recording the number of benchmark programs that passed. From this column, we can see that 7 of the 11 tools failed for at least one benchmark program.

5.4. Results Per Tool

Each tool is run over 11 test programs and then evaluated with 12 metrics, giving a total of 132 potential metric values.

Figure 2 sums up our findings on the eleven reverse engineering tools. In this figure, deep blue represents metric values that are under the lower control limit while deep red shows metric values that are above the upper control limit. Metric values that are shaded light blue are in the lower warning zone while metrics that are shaded orange are in the upper warning zone. The white-coloured section represents the number of metric values within the warning limits. Thus, for each UML tool, the larger the white shading in its bar in Figure 2, the greater the level of reliability we ascribe to the tool. The different height of the bars in Figure 2 reflects the different pass/fail results as shown in Table 4.

• **Bouml** Of the 9 applications Bouml produced an output for, it failed to capture the IC and the II metric. There was a trend of Bouml underestimating size metrics, while overestimating inheritance metrics. The MI metric is the only

	AR	MD	BO	MM	VP	JU	EA	UM	ES	IC	PO
antlr	P	P	P	P	P	P	F	P	P	P	P
eje	P	P	P	P	P	P	P	P	P	P	P
fop	P	P	P	P	P	P	P	P	P	P	P
hsqldb	F	P	P	P	P	P	P	P	P	P	P
jameleon	P	P	P	P	P	P	P	P	P	P	P
java2d	P	P	P	P	P	P	P	P	P	P	P
jolden	P	P	P	P	P	P	P	F	P	P	P
junit	P	P	P	P	P	P	P	P	P	P	P
pcj	F	P	F	P	P	P	F	P	P	P	F
pmd	P	P	P	P	P	P	P	P	P	F	P
xalan	P	P	F	P	P	F	P	P	P	P	P
ΣP	9	11	9	11	11	10	9	10	11	10	10

Table 4. This table lists the test suite applications in the left column and the UML tools in the top row. Each cell records the production of valid XMI, indicating either “P" for pass or “F" for fail.

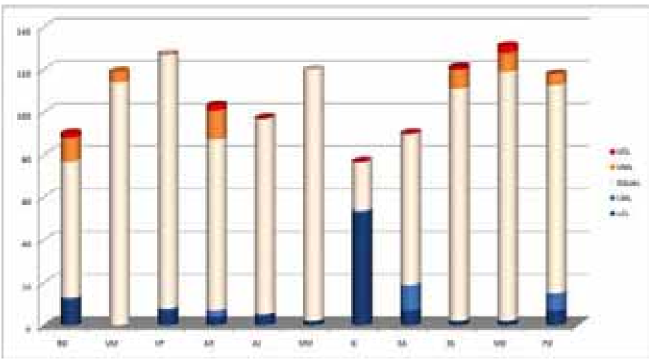


Figure 2. For each UML tool on the horizontal axis, this chart shows their success measured in terms of the metric values on the vertical axis

metric that Bouml produced that was completely accurate, with the CLD being slightly overestimated but acceptable.

• **UModel** UModel showed a trend of being correct a large proportion of the time. UModel failed to capture one metric NoG for one application. None of UModel's metric values were outliers, with only one metric value being in the warning zone.

• **Visual Paradigm** Visual Paradigm was accurate with size metrics but failed to capture metrics for 5 of the inheritance metrics and reported metric values below the lower control level for 3 inheritance metrics.

• **ArgoUML** ArgoUML has both under and over estimated many of the metrics, with a clear trend showing, the tool under-estimating the size metrics while over-estimating the inheritance metrics.

• **Metamill** Metamill failed to capture the II metric for all the applications, it also failed to capture the IC metric for one application. Out of 132 possible metric values, Metamill captured 120, or 91% of the metrics.

• **Enterprise Architect** Enterprise Architect failed to capture IC and II metrics for any of the applications. Investigating the results further shows that Enterprise Architect consistently under-estimates all the metrics.

• **ESSModel** ESSModel both over and under estimated metrics for a variety of different metrics, there was no noticeable trend.

• **Magicdraw** We can see that Magicdraw over and under reported the IC metric. The majority of the tools struggled to report this correctly, with many tools failing to capture it at all. In total Magicdraw correctly reported 117, or 89%, of the metric values.

• **Poseidon** Out of the 10 applications Poseidon did report metrics for, it failed to capture 2 metrics for one application, so in total Poseidon captured 119 metric values. For approximately half the metrics, Poseidon under reported them, with it only over reporting for the II metric.

• **Jude** Jude failed to capture the II metric but, of the remaining metrics, only two reported values in outside the Control Limits.

• **Ideogramic UML** Ideogramic failed to capture the II and IC metric. In the majority of metrics Ideogramic UML failed to capture a metric for each tool, in fact it only captured all the metrics values for a third of the metrics. In total it captured 23 metrics (17%) that were deemed to be correct.

6. Conclusions

Figure 2 shows Visual Paradigm to be the most reliable tool since it reported metric values that were accurate 90.15% of the time. It was closely followed by Metamill, Magicdraw and UModel. The worst performing tool was the Ideogramic UML tool, reporting just 17% of the metric values correctly.

Summary REM4j is an automation framework that evaluates a UML CASE tool's ability to reverse engineer and export valid XMI for class diagrams. It has been used to evaluate 11 UML tools using software metrics over a variety of input programs and clearly highlighted differences in the results obtained.

References

[1] L. A. Barowski and J. H. Cross II. Extraction and use of class dependency information for Java. In *9th Working Conf. on Reverse Engineering*, 2002.

[2] E. J. Chikofsky and J. H. C. II. Reverse engineering and design recovery: A taxonomy. *IEEE Softw.*, 7(1):13–17, 1990.

[3] D. Cooper, B. Khoo, B. R. von Konsky, and M. Robey. Java implementation verification using reverse engineering. In *27th Australasian Conf. on Computer Science*, pages 203–211, 2004.

[4] T. R. Dean, A. J. Malton, and R. Holt. Union schemas as a basis for a C++ extractor. In *8th Working Conf. on Reverse Engineering*, page 59, 2001.

[5] S. M. B. et al. The DaCapo benchmarks: Java benchmarking development and analysis. In *21st Conf. on Object-oriented programming systems, languages, and applications*, pages 169–190, 2006.

[6] R. Ferenc, F. Magyar, A. Beszédes, A. Kiss, and M. Tarkiainen. Columbus - tool for reverse engineering large object oriented software systems. In *7th Symposium on Programming Languages and Software Tools*, pages 16–27, 2001.

[7] J. Jiang and T. Systä. Exploring differences in exchange formats - tool support and case studies. In *7th European Conf. on Software Maintenance and Reengineering*, page 389, 2003.

[8] R. Kazman and S. J. Carriére. Playing detective: Reconstructing software architecture from available evidence. *Automated Software Eng.*, 6(2):107–138, 1999.

[9] M. Keschenau. Reverse engineering of UML specifications from Java programs. In *19th Conf. on Object-oriented programming systems, languages, and applications*, 2004.

[10] C. Kobryn. UML 2001: a standardization odyssey. *Commun. ACM*, 42(10):29–37, 1999.

[11] R. Kollmann and M. Gogolla. Re-documentation of Java with UML class diagrams. In *7th Reengineering Forum, Reengineering Week*, pages 41–48, 2000.

[12] R. Kollmann and M. Gogolla. Metric-based selective representation of UML diagrams. In *6th European Conf. on Software Maintenance and Reengineering*, page 89, 2002.

[13] M. Lorenz and J. Kidd. *Object-oriented Software Metrics*. Prentice Hall, 1994.

[14] R. Marinescu and M. Lanza. *Object-oriented Metrics in Practice*. Springer, 2006.

[15] M.-A. D. Storey, K. Wong, and H. A. Müller. Rigi: a visualization environment for reverse engineering. In *19th Intl. Conf. on Software Engineering*, pages 606–607, 1997.

[16] A. Sutton and J. I. Maletic. Mappings for accurately reverse engineering UML class models from C++. In *12th Working Conf. on Reverse Engineering*, pages 175–184, 2005.

[17] J. Wüst. SDMetrics, 2006. http://www.sdmetrics.com.

FlexUML: A UML Profile for Flexible Process Modeling

Ricardo Martinho
School of Technology and Management, Polytechnic Institute of Leiria, Portugal
rmartin@estg.ipleiria.pt

Dulce Domingos
Faculty of Sciences, University of Lisboa, Portugal
dulce@di.fc.ul.pt

João Varajão
Engineering Department, University of Trás-os-Montes e Alto Douro, Portugal
jvarajao@utad.pt

Abstract

Process modeling involves eliciting and capturing informal process descriptions into process models. A process model is expressed by using a suitable Process Modeling Language (PML). Flexibility in process modeling denotes the ability to modify only those parts of a process model that need to be changed and keeping other parts stable. The lack of flexibility has recently become a main challenge in Process Aware Information Systems (PAIS) research areas. However, latest research points out the fact of PAIS having the "wrong" rigidity, rather than suffering from the lack of flexibility. The Unified Modeling Language (UML) is often used as a general purpose PML. In this paper we propose the FlexUML profile: an extension to UML 2.0 activity diagrams (ADs) that enables modelers to design process models with the "right" flexibility.

Keywords: process modeling, flexibility, activity diagrams, UML profile

1. Introduction

Since the early 1990s, there has been a shift from data orientation to process orientation, triggering the development of Process Aware Information Systems (PAIS): software systems that manage and execute operational processes involving people, applications, and/or information sources on the basis of process models [4]. Process modeling involves eliciting and capturing informal process descriptions, and converting them into process models. A process model is expressed by using a suitable Process Modeling Language (PML), and is best developed in conjunction with the people who are participants in, or are affected by, the process. Therefore, it is important to them to be familiar with concepts expressed by the PML. This is generally accomplished by the use of meta-modeling, wherein a process modeler can specify the vocabulary and concepts used for process modeling.

One of these important concepts is *flexibility*. Process flexibility denotes the ability to modify only those parts of a process that need to be changed and keeping other parts stable [14]. The lack of flexibility as been pointed out in the last years as one of the main causes for Process Aware Information Systems (PAIS) low adoption [2]. The need for flexibility derives essentially from poor, rigid and user-unadaptable systems that underestimate human interaction, either in process execution, or in any other phase of the process lifecycle.

However, in the everyday business practice, most people do not want to have much flexibility, but would like to follow very simple rules to complete their tasks, making as little decisions as possible. In fact, latest research points out to the problem of PAIS having "wrong" rigidity, rather than suffering from the lack of flexibility [1]. In our case, we identify the need to express the "right" flexibility into process models, i.e., to express *which*, *where* and *how* flexible are the elements that compose a process model. As an example, let us consider a software company that uses a Unified Process [8] based model to manage and develop software projects. Several participants may contribute to refine the model. Nevertheless, there is a senior process modeler that designs its overall activities and related control flow, resources and data. For instance, she defines that a certain activity has a fixed goal of providing a fully specified UML use case model, accomplished by the execution of five main sub-activities. Therefore, she presents a preliminary arrangement for those sub-activities, but she also wishes to express that they can be changed by other participants (e.g. software architects), according to some restrictions. The PML must support concepts that will allow her to identify *which* elements composing those activities are to be flexible (e.g. sequencing control flow elements), *where* they can be changed (*type* or *instance* level), and *how* they can be changed (e.g. a *temporary* change in an activity resource).

The Unified Modeling Language (UML) [12] can be used as a PML (see, e.g., [6, 5]), having activity diagrams (ADs) as the base diagrams. Main advantages include its popularity, graphical notation, extensibility, well supported standard, tool support and integration facilities (XMI) [6]. It also provides a meta-model and a notation guide, defining concrete syntaxes for elements that can be used in process modeling. However, UML does not include means to express flexible aspects for those modeling elements.

In this paper we propose FlexUML: a UML profile that extends UML 2.0 ADs in order to provide a PML for the design of flexible process models. The profile is based on a reference meta-model that includes flexibility vocabulary and concepts, and provides the ability to combine them into a single AD modeling element.

Below we proceed as follows: in section 2 we briefly describe how UML 2.0 ADs can be used for process modeling. In section 3 we present generic flexibility concepts that will be mapped into the reference meta-model proposed in section 4. With this meta-model, we derive the FlexUML profile and provide an example of its application. Section 5 discusses related work and section 6 concludes the paper.

2. Process Modeling using UML

Complete process representation is generally achieved using several perspectives that integrate many forms of information. PMLs usually are able to represent one or more of these perspectives, including [3, 5]: 1) the *behavioral perspective*, representing when do process elements are performed (e.g. sequencing), as well as aspects of how they are performed through feedback loops, iteration, complex decision-making conditions, entry and exit criteria, and so forth; 2) the *functional perspective*, illustrating what flows of informational entities (e.g., data, artifacts, products), are relevant to process elements; 3) the *organizational perspective*, that represents where and who performs actions in the process; and 4) the *informational perspective*, that describes the informational entities produced or manipulated by the process.

ADs often serve as the base type of diagrams for process modeling with UML 2.0. We present in the next subsections a brief description of ADs elements for modeling process behavior, objects and object flow and organizational perspective (further details available in [12], sections 11 and 12).

2.1. Process behavior

Graphically, ADs are composed of nodes and edges. The edges connect the nodes in sequential order. Nodes represent either actions, activities, data objects, or control nodes. The fundamental unit of behavior specification in ADs is the `Action`. Actions access, transform and test data, and may require sequencing. Control nodes are used to structure control and object flow. These include decisions and merges to model contingency, and initial and final nodes for starting and ending flows. They also include forks and joins for creating and synchronizing concurrent subexecutions [12].

`Activities` represent the overall behavior of a system, and are composed of actions and/or other activities and related dependencies. In Figure 1 we present basic and advanced constructs (with notations) of UML 2.0 ADs, including actions, control nodes, activities, interruptible regions, expansion regions and exception handlers.

2.2. Objects and object flow

Objects and their types are used to represent physical entities like products or persons, information like data or documents, as well as logical concepts like product types or organizations. Figure 2 illustrates both object nodes and object flow (alternative) representations within UML 2.0 ADs.

In process modeling, it is important to define

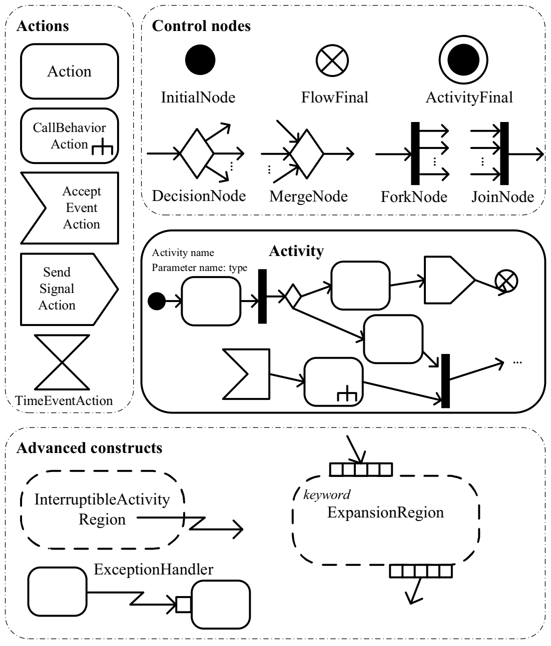


Figure 1: Basic and advanced behavioral constructs of UML ADs

the dependencies between objects and actions occurring in activities, in particular input and output relationships as well as object flow dependencies.

2.3. Organizational perspective

Representing the organizational perspective in UML 2.0 ADs is done by using *activity partitions*, which divide the set of nodes within an activity into different sections. Nodes can belong to none, one, or more partitions at the same time. Partitions can be visualized in two different ways (see Figure 3): 1) the partition name is written in parenthesis over the action name within the action symbol; or 2) using swimlanes drawn through the activity diagram, dividing it into different sections. The name of the partition is displayed on the top of the swimlane. Activity diagrams can also be partitioned multidimensionally, where each swimcell is an intersection

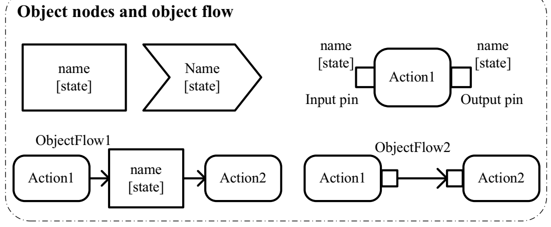


Figure 2: Object nodes and flow in UML ADs

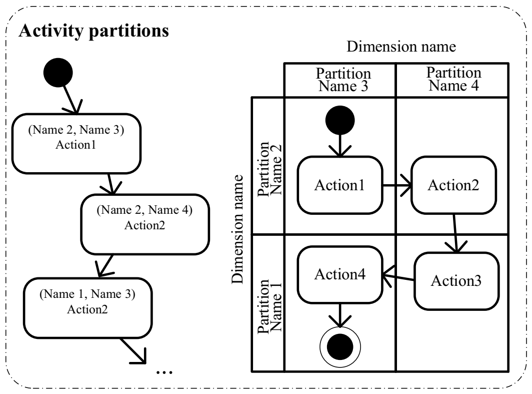


Figure 3: Activity partitions in UML ADs

of multiple partitions. This is often used to express both physical locations and roles simultaneously.

In the next section we provide foundations for process flexibility, along with a generally accepted taxonomy.

3. Flexibility in process modeling

Process flexibility denotes the ability to modify only those parts of a process that need to be changed and keeping other parts stable, i.e., the ability to change the process without completely replacing it [15].

Based upon a recent taxonomy proposed in [13], process modeling flexibility can be classified according to three orthogonal dimensions: 1) *abstraction level of change*, that distinguishes *where* changes are to be made, i.e., if at the *type* or *instance* levels (or both). Changing the process model (*type* level) implies changing the defined standard way of working, as it will affect all instances created there forward. However, change can occur only for certain instances of a process (*instance* level), in order to accommodate exceptional situations; 2) *subject of change*, representing *which* modeling elements are to be changed, and, consequently, which related perspective(s). For instance, changing an input parameter of an activity means changing both its functional and informational perspectives; and 3) *properties of change*, denoting *how* can a modeling element be changed. We consider four combinable properties of change: 1) the *extent* of change, denoting if change is only introduced to an already existing process model (*incremental* change), or if change abolishes the existing process model and creates a completely new one (*revolutionary* change). Often experts are required to do revolutionary changes to the whole or part of a process model. 2) the *duration* of change, that can represent *temporary* or *permanent* changes. Temporary changes are valid for a limited period of time, and permanent changes are valid until the next permanent change; 3) the *swiftness* of change, that expresses if changes are to be applied *immediately* to all process model instances (also the running ones), or *deferred* only to new instances of the changed process model; and 4) the *anticipation* of change, that identifies if the change is *planned* or *ad-hoc*. Ad-hoc changes are often made to tolerate exceptional situations, and planned changes are often part of a process redesign.

Based on these flexibility concepts, we present in the next section our proposed UML profile. This profile will provide the ability to support the referred flexibility concepts in AD process models.

4. UML Activity Diagrams as a flexible process modeling language

Since its inception, UML was intended to cover as many domains as possible. That is the reason why it provides extension mechanisms, such as stereotypes, tagged values and constraints [12]. These mechanisms allow designers to customize UML to a particular domain or purpose. A coherent set of such extensions constitutes a UML Profile. After identifying the flexibility concepts in the previous section, we turn, in the next subsections, to the very UML viewpoint focusing on the mapping of those concepts onto a UML meta-model, and deriving a UML profile accordingly.

4.1. The FlexUML Profile

The first step to create a UML profile is to define the meta-model that expresses the application domain. In our case, the application domain includes both existing UML 2.0 AD modeling elements and (new) flexibility modeling elements. In Figure 4a), flexibility-related elements are distinguished by a shaded class notation.

The meta-model structure is based on the *decorator* design pattern [7]. The structure is composed by a common abstract `Element`, from which both AD existing elements and flexibility elements specialize. The decorator pattern allows to attach additional responsibilities to an object dynamically, i.e., a process modeler can pick an AD element (such as an `ActivityNode`) and *decorate* it with *flexibility* responsibilities, represented by `FlexibleDecorator`'s specialized elements. These ele-

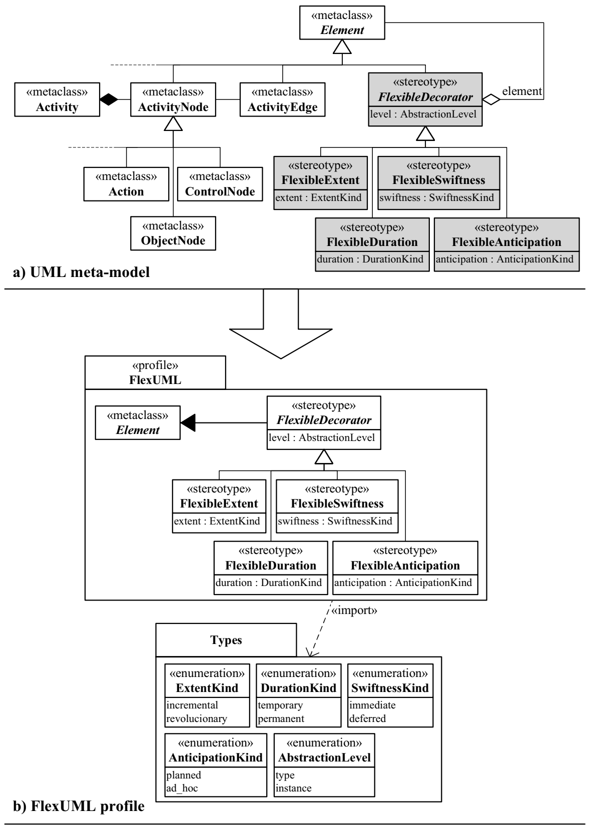


Figure 4: a) UML meta-model and b) related stereotypes introduced with the FlexUML profile

ments derive from the four properties of change and related attributes identified in the previous section. Besides those attributes, each `FlexibleDecorator` inherits a `level:AbstractionLevel` attribute that identifies the abstraction level of change it represents (*type* or *instance*).

Figure 4a) also illustrates which «`metaclass`» elements are eligible to be extended by the profile's «`stereotype`» elements. Stereotypes represent the new flexibility features added to the UML meta-model for extending the language. Figure 4b) shows a UML package representing the `FlexUML` «`profile`», derived from Figure 4a) meta-model. The arrow pointing from `FlexibleDecorator` to `Element` with a filled triangle arrowhead is a UML `Extension`. In our case, it indicates that properties of AD `Element` classes are extended through the use `FlexibleDecorator` stereotypes, giving the ability to add (and later remove) them to any `Element` class in a flexible way.

The `FlexUML` «`profile`» package has also an «`import`» dependency with the `Types` package, which is composed by «`enumeration`» types for the attributes of each specialized `FlexibleDecorator` stereotype.

Applying a profile means that it is allowed, but not necessarily required, to apply the stereotypes that are defined as part of the profile. Stereotype attributes (also known as *tag definitions*) assume *tagged values* when a stereotype is applied to a model element [12].

In the next subsection we present an example to better illustrate how we can use the FlexUML profile.

4.2. Applying FlexUML: an example

Our example reports to a software company that uses Unified Process-based models to manage and develop software projects. The company also uses a PAIS that interprets and executes operational process definitions derived from the UP-models. Process participants use UML 2.0 ADs as the standard PML, and all can contribute to refine process models according to their skills. However, there's a senior process modeler that not only designs the overall activities and related control flow, resources and data for the process model, but also defines which parts of the model and where and how they can be flexible.

Figure 5 shows a process model for the UP *Elaboration* phase. The gray-filled AD model elements represent the parts of the process that are flexible. FlexUML stereotypes are shown in guillemets and their tagged values in brackets, both above the name of the element to which they are applied.

The example has four (numbered) profile applications, each one referring to different types of AD elements. The 1) `UMLUseCaseModel` object node is represented with a «`FlexibleExtent`» stereotype, along with the `{instance, incremental}` tagged values, signaling that `instance`s of the object node can be `incremental`ly changed. The next 2) flexible element is an `ExpansionRegion` that contains three sequentially arranged actions, and to which are applied two flexibility stereotypes: «`FlexibleAnticipation`» and «`FlexibleSwiftness`». The related tagged values define that changes made to this element are to be carefully `planned` and `deferred` to new instances (not affecting the running ones). The 3) horizontal `JoinNode` can suffer incremental changes at the type level, and the 4) `Test prototype` action has a «`FlexibleDuration`» partition. This means that, for all `instance`s of this action, the assigned role of its performers (like the suggested `Tester` role) can be `temporarily` changed.

5. Related work

Foundations for modeling distinct process perspectives with UML can be found, for example, in [6, 5]. In [9] the authors propose dynamic task nets and the DYNAMITE PAIS to allow for changes and evolution in software process executions. In [11] the authors explore the possibility of also using UML as a formal language for operational (executable) process definitions, that can be interpreted and instantiated by a PAIS. However, neither of the above related works has concerns regarding the expression of selective flexibility concepts, nor present a flexibility meta-model for process modeling.

In [16] the authors shed some light in finding, through process mining, what process parts should be flexible. Recently we proposed in [10] a meta-model for the application of distinct flexible mechanisms onto UML modeling elements. Both these works are complementary to our proposed profile, as they can be useful in modeling flexibility and related mechanisms for each distinct AD modeling element.

6. Conclusion

In this paper we proposed the FlexUML profile for expressing flexibility in process modeling. We presented a reference meta-model that provides a mapping of flexibility concepts onto UML. The profile is derived from this meta-model, and consists of a set of stereotypes and related attributes that can be applied to each AD modeling element.

By using a decorator structure in our meta-model, flexibility responsibilities can be dynamically and differently combined with existing AD modeling elements. This means that instantiating the proposed meta-model (using e.g., a modeling tool) will provide process modelers the ability to design flexible process models, i.e., models that contain AD elements that can be changed by other process participants, under certain restrictions.

The intent of our proposed profile is not to identify specific changeable properties for each distinct AD element, as that would lead to a proliferation of different stereotypes and tag definitions. Moreover, working with many specific stereotypes would imply process modelers having advance knowledge of precisely *how* certain process parts can be changed. Instead, the FlexUML profile provides simple and usable generic stereotypes, leaving room for other more skillful participants to enforce the "right" flexibility within a process model.

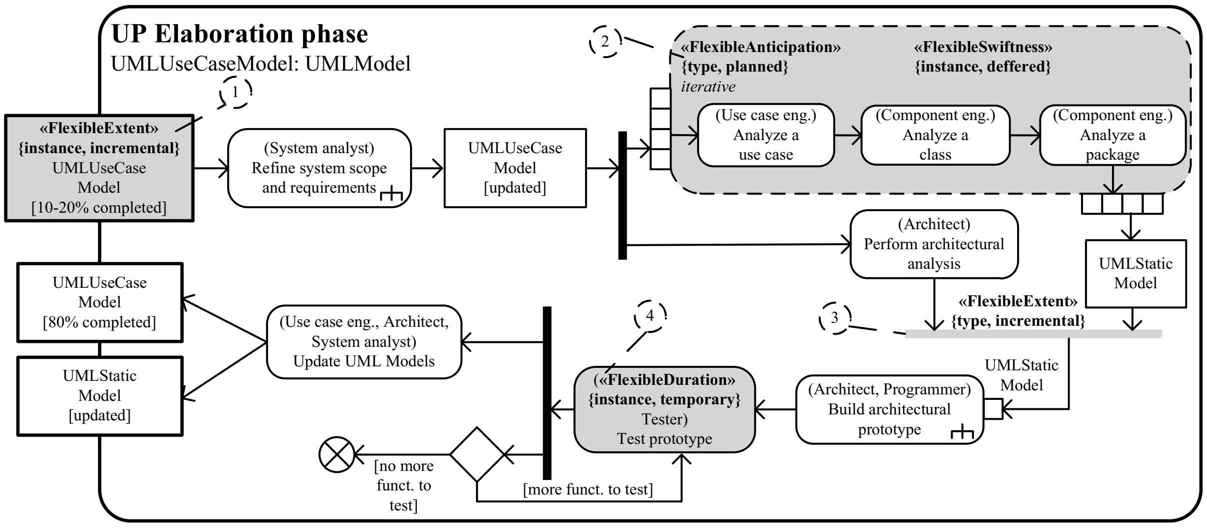

UP Elaboration phase
UMLUseCaseModel: UMLModel
1
2
«FlexibleAnticipation» {type, planned}
iterative
«FlexibleSwiftness» {instance, deffered}
«FlexibleExtent» {instance, incremental} UMLUseCase Model [10-20% completed]
(System analyst) Refine system scope and requirements
UMLUseCase Model [updated]
(Use case eng.) Analyze a use case
(Component eng.) Analyze a class
(Component eng.) Analyze a package
(Architect) Perform architectural analysis
UMLStatic Model
3
«FlexibleExtent» {type, incremental}
4
UMLUseCase Model [80% completed]
UMLStatic Model [updated]
(Use case eng., Architect, System analyst) Update UML Models
(«FlexibleDuration» {instance, temporary} Tester) Test prototype
(Architect, Programmer) Build architectural prototype
UMLStatic Model
[no more funct. to test]
[more funct. to test]

Figure 5: Example process model for the UP *Elaboration* phase

References

[1] I. Bider. Masking flexibility behind rigidity: Notes on how much flexibility people are willing to cope with. In J. Castro and E. Teniente, editors, *Proc. of the CaiSE'05 workshops*, volume 1, pages 7–8, Porto, Portugal, 2005. FEUP.

[2] G. Cugola. Tolerating deviations in process support systems via flexible enactment of process models. *IEEE Trans. Softw. Eng.*, 24(11):982–1001, 1998.

[3] B. Curtis, M. I. Kellner, and J. Over. Process modeling. *Commun. ACM*, 35(9):75–90, 1992.

[4] M. Dumas, W. van der Aalst, and A. H. M. ter Hofstede, editors. *Process-Aware Information Systems: Bridging people and software through process technology*. John Wiley & Sons, Inc., 2005.

[5] G. Engels, A. Förster, R. Heckel, and S. Thöne. Process modeling using UML. In *Process-Aware Information Systems*, chapter 5, pages 85–117. John Wiley & Sons, Inc., 2005.

[6] H. Eriksson and M. Penker. *Business Modeling with UML, Business Patterns at Work*. John Wiley & Sons, 2000.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[8] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[9] D. Jäger, A. Schleicher, and B. Westfechtel. Using UML for software process modeling. In *ESEC/FSE-7: Proc. of the 7th European Softw. Eng. Conf./ 7th ACM SIGSOFT Int'l Symposium on Foundations of Softw. Eng.*, pages 91–108, London, UK, 1999. Springer-Verlag.

[10] R. Martinho, D. Domingos, and J. Varajão. A Flexible Perspective for Software Processes - Supporting Flexibility in the Software Process Engineering Metamodel. In *Proc. of the 9th Int'l Conf. on Enterprise Information Systems (ICEIS'07)*, Funchal, Madeira - Portugal, June 2007.

[11] E. D. Nitto, L. Lavazza, M. Schiavoni, E. Tracanella, and M. Trombetta. Deriving executable process descriptions from UML. In *ICSE '02: Proc. of the 24th Int'l Conf. on Softw. Eng.*, pages 155–165, New York, NY, USA, 2002. ACM Press.

[12] OMG. Unified Modeling Language: Superstructure, version 2.0. Technical report, Object Management Group, 2005.

[13] G. Regev, P. Soffer, and R. Schmidt. Taxonomy of flexibility in business processes. Input to the 7th Workshop BPMDS'06, Website, June 2006. `http://lamswww.epfl.ch/conference/bpmds06/taxbpflex`.

[14] G. Regev and A. Wegmann. A regulation-based view on business process and supporting system flexibility. In J. Castro and E. Teniente, editors, *Proc. of the CaiSE'05 workshops*, volume 1, pages 91–98, Porto, Portugal, 2005.

[15] P. Soffer. On the notion of flexibility in business processes. In J. Castro and E. Teniente, editors, *Proc. of the CaiSE'05 workshops*, volume 1, pages 35–42, Porto, Portugal, 2005.

[16] W. M. P. van der Aalst, C. W. Günther, J. Recker, and M. U. Reichert. Using process mining to analyze and improve process flexibility (position paper). In *Proc. of the CAiSE'06 Workshops / 7th Int'l Workshop BPMDS'06*, pages 168–177, Namur, Luxembourg, June 2006.

A Formal Specification for Product Configuration in Software Product Lines

Huilin Ye and Yuqing Lin
School of Electrical Engineering and Computer Science
The University of Newcastle, Callaghan, NSW 2308, Australia
{huilin.ye, yuqing.lin}@newcastle.edu.au

Abstract

Feature modeling has been widely used in product line based software development. The major purpose of feature models is for member product configuration. Feature dependencies have very strong implications on the configurations in a product line. However, current existing approaches to feature modeling provide limited support for the specification of the dependency constraints. A formal specification using Z is developed to specify the dependencies in product lines and the constraints on the selection of variable features for product configuration. In addition simple dependency constraints, complex conflicting feature dependencies caused by transitive dependencies can be detected during the process of product configuration. As Z specifications provide natural transition from the specifications to implementations it will be easy to implement a modeling tool to support product configuration based on the Z specification.

1. Introduction

Feature oriented modeling approaches have been widely used in product line based software development. Software Product Line (SPL) is a collection of systems that share common characteristics as a family in an application domain. The features identified from a SPL are prominent and distinctive system requirements or characteristics that are visible to various stakeholders in the product family [1]. A member software product in a SPL is defined by a unique combination of legally selected features in the SPL. The process of selecting features for a member product in a SPL is called *product configuration.* The configuration is based on a feature model. A feature model for a SPL specifies features, their relationships, and the constraints of feature selection for product configuration.

Currently existing feature modeling methods usually use a tree structure to organize features. To configure a member product of a product line using a feature tree, if a parent node of the tree has one or more variable child features, called variants, a decision must be made on the choice of the variants. The parent feature representing a decision point together with its possible choices (variants) is defined as *variation point* [2]. The selection of variable features must satisfy constraints specified in feature models, otherwise the configured product may not be a valid one. The constraints are usually dependencies among the features. These dependencies are non-hierarchical in nature. Representing the non-hierarchical dependency relationships into a tree structure has left the current feature modeling methods with the possibility of omitting dependencies [3]. In a large complex SPL the number of variation points may range in thousands and each variation point may have several variable features [4]. Therefore the number of combination of the selection of variable features will be very large and the constraints for the selection will be very complex. Specifications of dependency constraints in feature models have not been addressed appropriately by the current existing approaches. As the specification of dependency constraints is crucial for validating product configurations an appropriate mechanism should be developed.

This paper proposes a formal specification for member product configuration in software product lines using a well known formal language Z. The main advantage of using Z in comparison with other languages, such as Object Constraint Language (OCL) used in [5] and First-Order Logic used in [6] is that Z can collectively represent concepts and the constraints of the concepts in a single structure, called schema, which makes the specification more intuitive and comprehensive. A set of Z schemas is used to describe a software product line and to validate the configured member products. A Z schema consists of two parts, a declaration part and a predicate part. The declaration parts can be used to formally define various kinds of concepts, such as features and variation points, used in a SPL. The predicate parts consist of a number of logic expressions to specify the constraints on the concepts.

The remainder of the paper will be organized as follows. Section 2 will present feature models, including feature relationships and their representations. Section 3 will discuss member product configuration and the implication of feature dependencies on the configuration. The details of the formal specification for product configuration in software product lines will be described in Section 4. Section 5 will discuss the benefits of the approach and conclude the paper.

2. Feature models

A feature model is a hierarchical tree structure consisting of two kinds of features, mandatory features and variable features. Each node of the tree represents a feature while each connection between two features represents the relationships of the two features. The configuration of a member product in a SPL will go through the feature tree to include all the mandatory features and select some of the variable features at each variation point. Variable features include optional, alternative, and multiple alternative features.

- **Optional feature**: If a feature may or may not be selected into member products it is called optional feature.
- **Alternative features**: Among a set of variable features one and only one feature must be chosen for member products. This set of features is called alternative features.
- **Multiple alternative features**: Among a set of features at lease one or more features must be chosen for member products. This set of features is called multiple alternative features.

The three kinds of variable features can be differentiated by a specified multiplicity as follows [7].

- 0 .. *: For a variation point, each of its variants may or may not be chosen for member products. This multiplicity can be used to represent optional features.
- 1: One and only one feature can be chosen from a set of variants. This can be used to represent alternative features.
- 1.. *: One or more features can be chosen from a set of variants. This can be used to represent multiple alternative features.

The multiplicity must be associated with each variation point in a feature tree to specify how many variants can be chosen from each variation point.

Three hierarchical feature relationships, Composition, Generalization/Specialization, and Variation Point, have been identified.

- Composition: When a feature is composed by a set of child features, there is a composition relationship from the parent to the children.
- Generalization/Specialization: If a parent feature is a generalized feature of its children, there is a generalization/specialization relationship between the parent and its children.
- Variation Point: For a given feature tree, if a parent feature node has at least one direct child who is a variable feature, there is a variation relationship between the parent and the child. The parent node together with its direct variable child features is called a variation point.

Figure 1 shows a simplified feature model for a `Car` product line. A set of UML-based notations is used to represent the relationships between features. As there is no standard UML notation for Variation Point a circle is used to represent a variation point. In this model `Car` is composed by `Control`, `Accessories`, `Engine`, and `Quality attributes`. They are all mandatory features. The features specified with a stereotype `<<variant>>` are variable features. `Manual` and `Automatic` are the specialized features of `Control`. `Quality attributes` is composed by `Operating cost` and `Reliability` etc. `Control`, `Accessories`, and `Quality attributes` represent variation points. `Control` has a multiplicity of `1` specifying that its two variants are alternative features. `Accessories` has two optional variants represented by the multiplicity `0..2`. `Quality attributes` has multiple alternative variants represented by the multiplicity `1..3`.

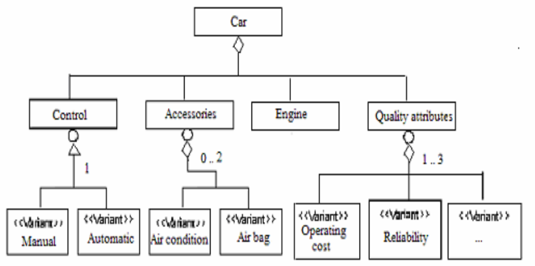


Figure 1. A feature model for a *Car* product line.

3. Product configuration and feature dependencies

The major purpose of developing a feature model is for member product configuration in a product line. Product configuration is a process of selecting variable features from each variation point of a feature model. Each configured product must include all the mandatory features and select different set of variable features based on the specific requirements of the product. Thus, how to select the variable features is crucial for the configuration. The constraints on the selection of the variable features will depend on the multiplicity specified for each variation point and the dependencies among the variable features.

As discussed earlier a multiplicity is associated with each variation point to specify the possible number of variable features that can be selected at this variation point. Three kinds of multiplicity are defined, 0 ..*, 1..*, and 1. For a multiplicity of 0 ..*, it will not be a problem when selecting any features from the set of variants of a variation point. . For a multiplicity of 1 ..*, it must be ensure that at least one feature will be selected at this variation point for any product configuration. For a multiplicity of 1 only one feature can be selected at this variation point for any product configuration.

In addition to the multiplicity constraints for selecting the variable features feature dependencies have much stronger implications on the selection of variable features [8]. Features are never stand alone in a product line and usually need to interact with, or use, other features to fulfill their tasks. How effective for a product line to derive configurable member products will greatly depend on the understanding of the dependencies between the variable features. Three feature dependency relationships have also been identified as follows:

- `Requires`: If a feature requires, or uses, another feature to fulfill its task, there is a `Requires` relationship between the two features.
- `Excludes`: If a feature conflicts with another feature, they cannot be chosen for the same product. There is a bi-directional `Excludes` relationship between the two features.
- `Impacts`: When a feature is selected for a certain product and the selection will have impact on another feature, it is called `impacts` relationship between the features.

The dependencies identified from a product line must be validated. For a complex product line involving a large number of features, some conflicting dependencies may be recorded without awareness of their existence [9]. The validation of dependencies is intended to discover these conflicting dependencies. The following rules are defined for the validation:

- `Excludes` relationship must be mutually exclusive.
- `Requires` and `Excludes` cannot be occur between the same pair of features.

When using the rules to validate the dependencies for a certain feature, only considering its direct dependants is not sufficient. All the indirect dependencies should be inspected as well. Additional dependencies may be derived via tracing the `Requires` dependencies and the derived dependencies may conflict with the existing dependencies. For example, if *Feature A* requires *Feature B* and *Feature B* requires *Feature C* and excludes *Feature E* then there are two derived dependencies, *A* requires *C* and *A* excludes *E.* These derived dependencies called transitive dependencies. Transitive dependencies can only be derived through direct `Requires` dependencies. When *Feature* A is selected, *Feature B* and *C* should also be selected and *Feature E* should be excluded. We extend this example in Figure 2. Starting from *A*, we trace the `Requires` dependencies until *Feature D* that has no `Requires` dependency with other features is found. The tracing path, from *A* to *B* to *C* to *D* forms a `Requires` *dependency chain.* If *A* is selected all the features on this chain should also be selected and all the features that have `Excludes` dependency with any of the features on the chain should be excluded. If the transitive dependencies are not inspected some conflicting dependencies may exist.

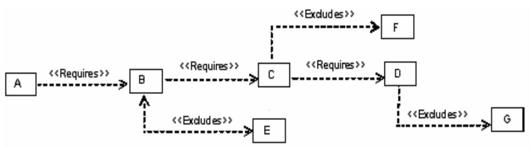


Figure 2 . An example of *Requires* dependency chain.

Feature dependencies are non-hierarchical in nature and cannot be represented appropriately in a hierarchically organized feature tree. A large number of connections of two dependent features crossing the branches of a tree will deteriorate the comprehensiveness and intuitiveness of the feature tree. A better representation of the dependencies should be developed. We use formal language Z to specify the dependencies in a product line. A set of Z operation schemas have also been developed to support product configuration. The detailed formal specification is presented in the next section.

4. Formal specifications

The formal specification has two parts, one for product line engineering and the other for application engineering. In the first part of the formal specification, we formally define a product line, the constraints on the product line. The second part of formal specification is used for product configuration. It provides a set of operation schemas to ensure that each selection of a variable feature is a valid selection to satisfy the required constraints. Consequently valid products can be configured based on the specification.

First we define a number of types.

[FEATURE-ID, VP-ID]

MultiplicityType ::= 0.. | 1..* | 1*

FeatureType ::= Mandatory | Variable

DependencyType ::= Requires | Excludes | Impacts

ImpactMessage::= $mesg_1|mesg_2|\ldots|mesg_m$

ErrorMessage::=MultiplicityError| $mesg_2|\ldots|emesg_m$

TruthValue ::= True|False

A feature can be modeled as a Z schema consisting of *identification number* and *feature type* (mandatory or variable).

```
──── FeatureInstance ────────────
│ id: FEATURE-ID
│ fType: FeatureType
└────────────────────────────────
```

The following global constraint ensures that every feature instance in a product line is unique.

∀f1, f2: FeatureInstance • f1.id = f2.id ⇔ f1 = f2

A schema defining a variation point consists of *identification number*, *parent feature*, a set of *variable child features*, and a *multiplicity* associated with the variation point.

```
──── VPInstance ─────────────────────────────────────
│ id: VP-ID
│ parent: FeatureInstance
│ variants: P FeatureInstance
│ multiplicity: MultiplicityType
├────────────────────────────────────────────────────
│ ∀var: FeatureInstance |var ∈variants • var.fType = Variable
└────────────────────────────────────────────────────
```

The *variants* defines a collection of variable child features associated with the *parent* feature at a variation point. The following global constraint ensures that every variation point instance is unique.

∀vp1, vp2: VPInstance • vp1.id = vp2.id ⇔ vp1 = vp2

The following schema defines a software product line. A product line consists of a set of features, variation points, and dependencies between the variable features. The relation *features* maps the features existing in a product line to the two different types (mandatory and variable). The *variationpoints* specifies a collection of variation points. The relation *dependencies* maps the dependency types to different sets of feature pairs that have different dependent relationships. The relation *ImptMesgs* maps a pair of features with `Impacts` relationship to an *impact message*. An impact message associated with a feature pair, such as *Feature A* and *B*, specifies the specific impact on Feature B when Feature A is selected.

```
──── ProductLine ─────────────────────────────────────────────
│ features: FeatureType ⇸ P FeatureInstance
│ variationpoints: P VPInstance
│ dependencies: DependentType ⇸ P (FeatureInstance X FeatureInstance)
│ ImptMesgs: (FeatureInstance X FeatureInstance) ⤔ ImpactMessage
├─────────────────────────────────────────────────────────────
│ ∀ft1, ft2: FeatureType | ft1, ft2 ∈dom (features ) ∧ ft1 ≠ ft2 •
│         features ft1 ∩ features ft2 = ∅
│ ∀vpins: VPInstance | vpins ∈variationpoints • vpins.parent ∈dom (features )
│         ∧ vpins.variants ⊆ dom (features)
│ ∀dpair: FeatureInstance X FeatureInstance | dpair ∈ ran (dependencies) •
│         first ( dpair) ≠ second (dpair) ∧
│         first ( dpair).fType = second (dpair).fType=Variable
│ ∀dt1, dt2: DependentType | dt1, dt2 ∈dom (dependencies) •
│         dt1= Requires ∧ dt2= Excludes   ⇒
│                 dependencies dt1 ∩ dependencies dt2 = ∅
└─────────────────────────────────────────────────────────────
```

The invariants assert the following

- Each feature instance cannot be a mandatory feature and a variable feature at the same time.
- The parent and variants of any variation point are the existing features in a product line.
- A feature cannot have dependent relationship with itself.
- Only the dependencies between the variable features should be recorded for configuration.
- Any pair of features should not have both the *Requires* and *Excludes* dependencies at the same time in a product line.

Based on the formal definition of a product line, member products within the product line can be configured. We use a state schema, called *ProductConfiguration*, to describe the state space of the product configuration. The *SelectedFeatures* are the features included in the configured product. The state invariant asserts that any selected feature for a member product belongs to the product line.

```
──── ProductConfiguration ──────────────────────────────
│ Ξ ProductLine
│ SelectedFeatures: PFeatureInstance
├───────────────────────────────────────────────────────
│ ∀fins: FeatureInstancee |fins ∈ SelectedFeatures •fins ∈ ran (features )
└───────────────────────────────────────────────────────
```

The initialization of a product configuration is specified by the following schema.

```
──── InitProductConfiguration ──────────────────
│ ProductConfiguration'
├───────────────────────────────────────────────
│ SelectedFeatures' = (features Mandatory) ∧
│ SelectedFeatures' ∩ (features  Variable) = ∅
└───────────────────────────────────────────────
```

The predicate part of *InitProductConfiguration* asserts the following conditions:

- Every mandatory feature must be included in any product in a product line.
- Initially, no variable feature has been selected.

After the initialization, all the mandatory features have been included in a product being configured. We then define a number functions for selecting variable features for the product. When selecting a variable feature several conditions should be satisfied. The first condition is the multiplicity specified for each variation point. The following schema is used to check multiplicity condition when a variable feature of a variation point is to be selected. The following function returns *True* if a variant can be selected at a variation point, i.e. in the situation that either multiplicity of the variation point is not 1 or no variant has been selected from the variation point if the multiplicity is one. When the multiplicity is not one, there is no limit on the maximum number of features that can be selected. While the multiplicity is one a variant can only be selected if there is no other variant has been selected.

```
── CheckMultiplicity  FeatureInstance → TruthValue ──
│ ΔProdutConfiguration
├──────────────────────────────────────────
│ ∀fins: FeatureInstance | CheckMultiplicityConstraint (fins) = True ⇔
│ (∀ vp: VPInstance, fins:FeatureInstance | fins ∈ vp.variants •
│         (vp.multiplicity = 1 ∧ vp.variants ∩ SelectedFeature = ∅)
│         ∨ (vp.multiplicity ≠ 1)
└──────────────────────────────────────────
```

In addition to the multiplicity constraint feature dependencies have great implications for product configuration. The following function specifies the constraints when selecting a variable feature that has *Excludes* dependency with another feature. If *Feature A* has `Excludes` relationship with *Feature B* when *A* is selected *B* should be excluded. We also need to remove *Feature B* from the selected feature set if it has already been included in the product.

```
── SelectExldFeature  FeatureInstance → TruthValue ──
│ ΔProductConfiguration
├──────────────────────────────────────────
│ ∀fins: FeatureInstance | SelectExldFeature (fins) = True ⇔
│         ∀ fpair: FeatureInstance X FeatureInstance |
│         fpair ∈ (dependencies Excludes) ∧ first(fpair) = fins •
│         SelectedFeature' = SelectedFeature ∪ {fins} \ {second(fpair)}
└──────────────────────────────────────────
```

The following function is used for selecting a feature that has *Impacts* dependency with another feature. The specific impact message associated with this relationship should be output.

```
── SelectImptFeature  FeatureInstance → TruthValue ──
│ Δ ProductConfiguration
│ message! : ImpactMessage
├──────────────────────────────────────────
│ ∀fins: FeatureInstance | SelectImptFeature (fins) = True ⇔
│         ∀ fpair: FeatureInstance X FeatureInstance |
│         fpair ∈ (dependencies Impacts) ∧ first (fpair) = fins •
│         SelectedFeature' = SelectedFeature ∪ {fins} ∧
│         Message! = ImptMesgs fpair
└──────────────────────────────────────────
```

When a feature to be selected has `Requires` dependency with another feature transitive dependencies must be taken into consideration. As discussed in Section 3 when selecting a feature that has `Requires` dependency with other features, we need to recursively to include all the features on its `Requires` dependency chain(s) and remove their excluded features. The following function recursively defines the function of selecting a feature with `Requires` dependencies.

```
── SelectReqFeature  FeatureInstance → TruthValue ──
│ Δ ProductConfiguration
├──────────────────────────────────────────
│   ¬∃ fpair0 : FeatureInstance X FeatureInstance |
│           fpair0 ∈ (dependencies Requires) ∧ first( fpair0 ) = fins •
│             SelectReqFeature (fins) = False
│    ∀ fpair: FeatureInstance X FeatureInstance |
│           fpair ∈ (dependencies Requires) ∧ first(fpair) = fins •
│             SelectReqFeature (fins) = True ∧
│             SelectedFeature' = SelectedFeature ∪ {fins, second(fpair)} ∧
│             (∀ fpair1: FeatureInstance X FeatureInstance |
│                fpair1 ∈ (dependencies Excludes) ∧
│                 second (fpair)= first (fpair1) •
│                  SelectedFeature' = SelectedFeature \ second (fpair1)) ∧
│            SelectReqFeature (second (fpair)))
└──────────────────────────────────────────
```

The following schema is defined for selecting features with no dependency to other features.

```
── SelectNoDependencyFeature  FeatureInstance → TruthValue ──
│ ΔProductConfiguration
├──────────────────────────────────────────
│ ∀fins: FeatureInstance | SelectNoDependencyFeature (fins) = True ⇔
│       ¬∃ fpair: FeatureInstance X FeatureInstance |
│       (fpair ∈ ran (dependencies )) ∧ first(fpair) = fins •
│       SelectedFeature' = SelectedFeature ∪ {fins}
└──────────────────────────────────────────
```

When selecting a certain feature it may have various kinds of dependencies with other features the *SelectFeature* operation uses *SelectExldFeature, SelectReqFeature, SelectImptFeature,* and *SelectNoDependencyFeature* functions defined before.

```
── SelectFeature ──
│ ΔProductConfiguration
│ selectFeature?: FeatureInstance
│ msg! : ErrorMessage
├──────────────────────────────────────────
│ CheckMultiplicity (selectFeature?) ≠ True ⇒
│        msg! = MultiplicityError
│ CheckMultiplicity (selectFeature?) = True ⇒
│        SelectReqFeature(selectFeature?) ∧
│        SelectExldFeature(selectFeature?) ∧
│        SelectImptFeature(selectFeature?) ∧
│        SelectNoDependencyFeature(selectFeature?)
└──────────────────────────────────────────
```

5. Discussion and conclusions

Feature models are used for member product configuration in product line based software development. Feature dependencies are the major constraints for the configuration. However, the current existing approaches to feature modeling provide limited support for the specification of the dependency constraints. Consequently contemporary tools often offer little or no support for the validation of feature models and the configured product.

Some related works have been done. The Adapted Object Constraint Language (A-OCL) was used to check consistency of feature models [5]. A set of constructs were developed to specify the constraints applicable to a certain feature or several features. When the feature dependencies in a product line are complex it is difficult to represent the dependencies in individual OCL constructs as there is no way to represent the relations between the constructs. For example, the transitive dependencies and the possible problems caused by the transitive dependencies discussed in Section 3 are difficult to be specified by the OCL constructs. Mannion employed First-Order Logic expressions to validate feature models [6]. Different kinds of dependencies have not been distinguished in the approach. It is difficult to formulate the product line logic expressions for complex dependency relationships among features. Such complex expressions are also difficult to understand. Other formal models using similar logic programming have been reported in [10-12].

The formal specification presented in this paper, in comparison with the abovementioned formal models, provides a generalized approach for product line engineering and application engineering (product configuration) and has the following advantages:

- It supports product line evolution. The defined schemas will detect any conflicting dependencies introduced by adding new features into a product line. It will be difficult for the logic expression based approaches to support product evolution as there is no facility to check any inconsistent dependency after a product line is updated.
- It validates the product configurations. The operation schemas defined for product configurations can inspect whether a selection of a certain feature is valid or not. The inspection can also detect the complex dependency inconsistencies caused by transitive dependencies.
- As Z specifications provide natural transition from the specifications to implementations it will be easy to implement a modeling tool based on the specification.

A modeling tool for product line engineering and product configuration will be developed in the near future. A large case study will also be conducted based on the developed modeling tool.

References

[1] Lee, K., Kang, K., and Lee, J., Concepts and Guidelines of Feature Modeling for Product Line Software Engineering, In *Proceedings of 7th International Conference of Software Reuse*, *LNCS*, 2319, pp.62-67, 2002.

[2] Riebisch, M., BÖllert, K., Streitferdt, D., and Philippow, I., Extending Feature Diagrams with UML Multiplicities. In *Proceedings of the Sixth Conference on Integrated Design and Process Technology*, Pasadena, CA, June 2002.

[3] Hein, A., Schlick, M., and Vinga-Martins, R. Applying Feature Model in Industry Setting. In *Software Product Lines – Experience and Research Directions*, pp. 47-70. Kluwer Academic Publishers, Boston, 2000.

[4] Bosch, J., Software Product Families in Nokia. *In Proceedings of 9th International Software Product Line Conference*, *LNCS*, Volume 3714, pp. 2-6, 2005.

[5] Streitferdt, D., Riebisch, M., and Philippow, I., Details of Formalized Relations in Feature Models Using OCL. In *Proceedings of the 10th IEEE Symposium and Workshops on Engineering of Computer-Based Systems(ECBS)*, Huntsville Alabama, USA, 2003, pp. 297-304.

[6] Mannion, M., Using First-Order Logic for Product Line Model Validation. In *Proceedings of 7th International Software Product Line Conference*, *LNCS,* Volume 2379, pp.176-187, 2002.

[7] Ye, H. and Liu H., "An Approach to Modeling Feature Variability and Dependencies in Software Product Lines." *IEE Proceedings – Software*, **152**(3), pp. 101-109, 2005.

[8] Ferber, S., Haag, J. and Savolainen, J., Feature Interaction and Dependencies: Modelling Features for Reengineering a Legacy Product Line. In *Proceedings of Second International Conference on Software Product Line*, pp. 235-256, 2002.

[9] Ye, H. and Sharmin, A. "Modelling Feature Variability and Dependency in Two Views." In Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering, Taipei, Taiwan, pp. 661-664, July 2005.

[10] Neema, S., Sztipanovits, J., and Karsai, G., Constraints-Based Design Space Exploration and Model Synthesis. EMSOFT 2003, *LNCS,* 2855, pp.290-305, 2003.

[11] Batory, D., Feature Models, Grammars, and Propositional Formulas. SPLC2005, *LNCS,* 3714 , pp.7-20, 2005.

[12] Benavides, D., Trinidad, P., and Ruiz-Cortes, A., Automated Reasoning on Feature Models. *Conference on Advanced Information Systems Engineering (CAISE)*, July 2005.

Designing a Platform-Independent Use-Case for a Composite Application using a Reference Architecture

Helge Hofmeister* and Guido Wirtz
Otto-Friedrich-University Bamberg, Distributed and Mobile Systems Group
Feldkirchenstr. 21, 96052 Bamberg, Germany
hofmeister@ecoware.de, guido.wirtz@wiai.uni-bamberg.de

Abstract

This paper describes the design of an industry-scale use-case for a composite application. The described design is based on former work of the group which basically investigated the reference architecture and development methodology for composite applications [8, 9]. The paper puts its focus on the platform independent design of the composite application from the business-process based requirements down to the design of the required services. Additionally a meta-model is provided that describes the hierarchy of services that need to be designed in order to meet business requirements in a more flexible way.

Keywords: SOA, composition, reference architecture

1. Introduction

Service-oriented architecture (SOA) is currently a buzz-word hitting the industry. Following the discussions in, e.g. [4] or [14], service orientation might be more or less just a relaunched distributed object approach. Although being recognized in the software industry most customer-industries, however, are still questioning the benefit of this approach.
We decided to evaluate the business value of SOA in a large-scale project for an IT service provider in the area of chemical industry. One key part of this (ongoing) evaluation is to apply the service-oriented paradigm to a real-life industry-scale use case. Service-orientation should proof that it brings value into a solution landscape with various legacy applications partially supporting business processes.
Our previous research on SOA in the same context resulted in the definition of a reference architecture [8, 9] as well as a proposal for a methodology [10] that can be used to design service-oriented applications - so-called composite applications - according to the reference architecture. This paper discusses our findings in applying this methodology to the design of the real-life use-case from industry and validates our results.
We refer to [8–10] for an in-depth discussion of related work regarding our overall approach. The work on service-design as described in [6] and [11] as well as the work of the SDO [3] specification in terms of data standardization have influenced the design we present in this paper. Moreover, the transformation of event-driven process chains (EPC) into WS-BPEL [2] executable processes as described in [5] is important for the case study since it allows for the business-process centered approach we follow.
The rest of the paper is organized as follows. After giving a sketch of our approach in section 2, section 3 discusses the design case study in detail. The paper concludes with an outlook on future work in section 4.

2. A Sketch of the Approach

The design of the pilot was done using a development methodology as well as using a reference architecture mutually corresponding. In order to meet both, a clean design and the industry-scale reality of predetermined tools of the corporate environment, we decided to split the design into three major sections: The platform-independent model (PIM), the platform-specific model (PSM) and a landscape architecture[1]. The aim is to design the composite application based on the tool-independent reference architecture. Using a generic mapping of the architecture to a specific set of tools and products, the PIM is specified with regards to the actual products in a later phase. The PSM and the concrete realization are then vivid demonstrations of the real-life applicability of the elaborated concepts. The actual tool-independent application of the concepts to real-life business requirements proofs the applicability of the design principles. Finally, the landscape architecture is the topology of deployed products and servers.

3. The Case Study

The use-case of the pilot project was chosen by a dedicated internal working group. Lacking a formal methodology, the group analyzed on a business expert level several business processes and how steps that are included in these processes

*Helge Hofmeister is an external PhD student with the Distributed and Mobile Systems Group at University of Bamberg.

[1]We still investigate the suitability of the model-driven architecture (MDA) [7] to our approach.

might be used in other processes as well. Hence, the advantage of service-oriented architecture that was best understood by the business analysts was reusability across system boarders. The business process finally chosen was the so-called ”agreement management” business process. This process describes the procedure how the company reacts on customer demands by estimating the request and providing an offer to the customer.

3.1. BP-based Functional Requirements

The business process is used as a functional requirements specification for the composite application (see Figure 1 for an extract modelled as an EPC). It involves control and data flow as well as the organizational/system flow. The process describes a collaborative sequence where different experts from single delivery units work out descriptions of the offer as well as cost estimations. In the subsequent phase, the account management (AM) uses this information to generate an offer that might lead to a contract with the customer.
Up to now, this process is supported by various applications: Sales and distribution (SD) as well as project management (PM) use an ERP system. Additionally, there is a change-management database that is employed in the different ITIL processes of the company [1]. This database is used to store service-level agreements, service descriptions and assets that are required for performing the services. The account management today uses a CRM application for managing offers and contracts. The aim of the case study is to automate this process using the existing system landscape by building a composite application unifying also the user access to the systems.

3.1.1. Enterprise Service Matching

The business process describing the requirements is used as the top-level orchestration of the involved processes. Following this approach, EPC functions (aka. Business tasks; green boxes) are so-called enterprise services. In order to allow the concrete implementation in terms of a service orchestration, the enterprise service themselves might need to be aggregated services of lower levels. In order to determine whether such an aggregation is required, the design methodology starts with an ”enterprise service matching”-phase. This phase describes the determination of native services being exposed by applications (application services) that might fit to the described function. Since our pilot project is the first experience with implementing composites, there are no matching enterprise services.

3.1.2. Data Interaction Analysis

In order to set-up new enterprise services matching the business functions, a data-interaction analysis based on the functional requirements is performed next. This step mainly analyzes which data are required by the single functions as their respective input and output. Having identified these data, the required data structures may be deduced from the enterprise's canonical data model.

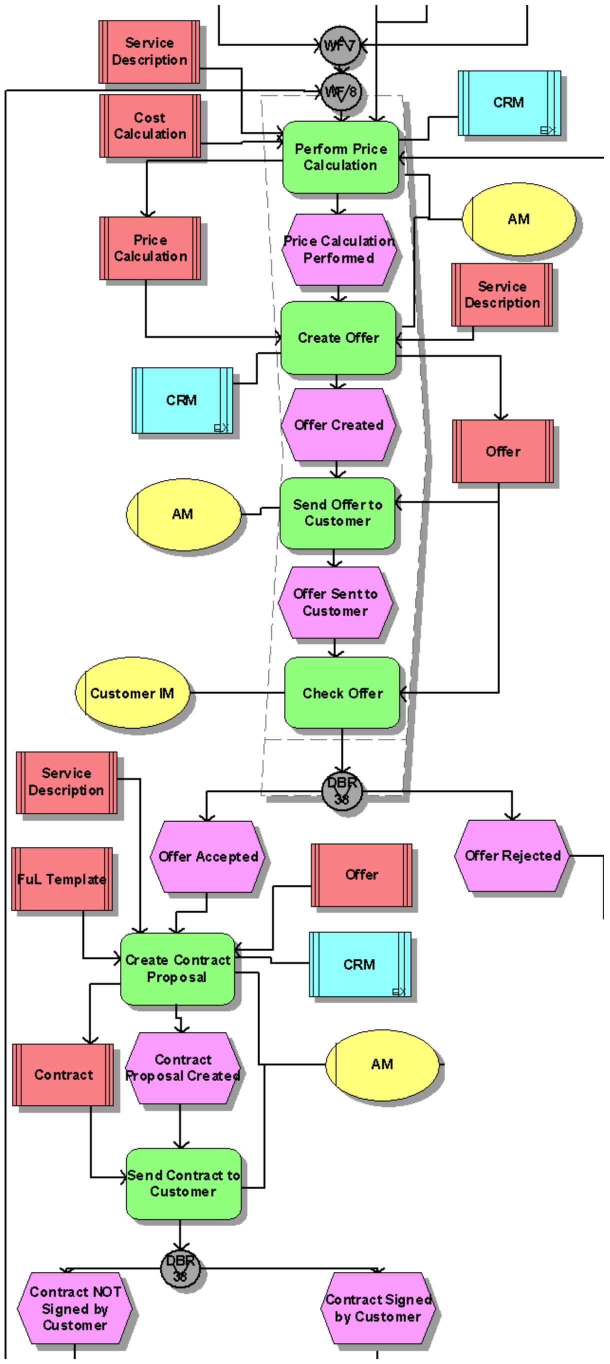


Figure 1. EPC for Functional Requirements

3.1.3. Transactional Property Identification

Knowing the data and the respective data access performed by the to-be enterprise services, some transactional properties of the process flow can be identified. Meeting the characteristics of business processes being long-running transactions, no ACID transactions involving multiple enterprise services are identifiable. Though, global transactions with relaxed ACID properties can be identified. In the business process snippet in Figure 1, the set of enterprise services performing a global transaction are marked by an underlying arrow (”Perform Price Calculation” till ”Check Offer”). In the given example the functions are grouped as a single global transaction since the state reached after checking the offer might require a new cycle of the offer phase that would replace the previously created offer. It is notable that all global transactions identified in the business process occur in process branches being encapsulated by a workflow pattern (8 in [13]) and a data-based routing pattern (38 in [12]).

3.1.4. Service Composition

This phase is the main phase of the PIM because it describes functionality that is independent of any target architecture. Nevertheless is this the phase creating the design of the enterprise service's implementation. As composite applications are mainly orchestrations of functionality being already existent in legacy applications, the main aspect of our design phase is the description of how application functionality that is accessible by the means of ”application services” is aggregated by the service coordination layer. The design of the service composition involves all layers of the underlying reference architecture, though. This is because the service coordination layer's design describes also the requirements for lower layers. These are mainly requirements for the data-exchange and transformation layer and the data repository. Often, the coordination describes an additional requirement for the design of new application services. This is because application functionality may not always be defined by means of services[2]. The design method and the example service design are described in the next section.

3.2. Enterprise Service Design

The design of the service coordination is based on three requirements: The description of the business functions that results in the enterprise services, the data flow and the corresponding data model that are computed by the business functions as well as the organizational perspective describing how humans and application systems are considered to interact in order to perform a certain business task. Notable about these three items is that the actual service designed is completely determined by the requirements expressed in the business process. Even the application systems that need to act as agents for the service provider part of the application services, are identified by that description. The approach incorporates these aspects since business experts turn out to often know precisely which legacy systems are capable of providing the required functionality - and these determinations are usually to be kept in order to allow for an agile deployment of composite applications.

[2]”means of services” refers not to the characteristics of services of using a common protocol. Furthermore does this refer to the encapsulation of functionality into coarse-grained and stateless units of work - services.

The starting point of the service coordination design is the actual business task. As an earlier step may already have identified existent enterprise services (”perfect matches”), the service coordination might be obsolete. Whenever no enterprise services have been matched, the initial decision of the coordination's design is whether the business task could be mapped to either an application system or a user interaction[3]. This is a design decision that has to consider two influence factors: The possibility of a single service agent to implement the functionality completely and the potential reusability of the enterprise service. Whenever parts of the required functionality can be mapped to existent application services, the services shall be reused and the enterprise service therefore will be composed of several application services.

Usually the coordination involves both reusing application services and designing new application services. During the design phase of the case study it turned out that the design of new application services is an informal procedure that relies on a subjective design of the solution architect. This is the case since the to-be application services need to fulfill immediately the requirements of the business task. Hence, the first step of the application service design is to model application services that support the requirements. This approach results in a process flow at the coordination layer that orchestrates several application services that compute data. Which data is actually computed results out of the data interaction analysis of the business process and the identified data model.

Since the identified data determines the context of both, the business process as well as the coordination process, the reference architecture relies on a data repository the context is kept in. Even if the context data is either retrieved from legacy applications or is created by human users during the process cycle, data access is not directly modeled as calling application services. The data repository is included in the coordination design as a separate entity (cf. [8, 9]).

The design steps described so far lead to two classes of services: enterprise services (that are by definition the tasks of the business process) and the steps of the coordination layer. The latter are referred to as ”coordination services”. Coordination services are introduced in order to allow for the design of non-existent application services. Coordination services might be entity-centered or task-centered. Entity-services are used to read or manipulate data kept by the legacy applications. Entity-centered services utilize application services in order to make data available to the context. Task-specific coordination services finally provide the functionality that operates on this context. The meta-model of our service hierarchy is shown in Figure 2.

Including the data repository into the reference architecture enables stateless services that are better reusable as the data repository respectively the entity-services manage the state of the composite while all participating services do not need to keep a state. In order to allow for reusable services, stateless

[3]Even if the reference architecture handles user-interactions like every other legacy functionality, it turned out to be helpful to describe application services and user interactions separately.

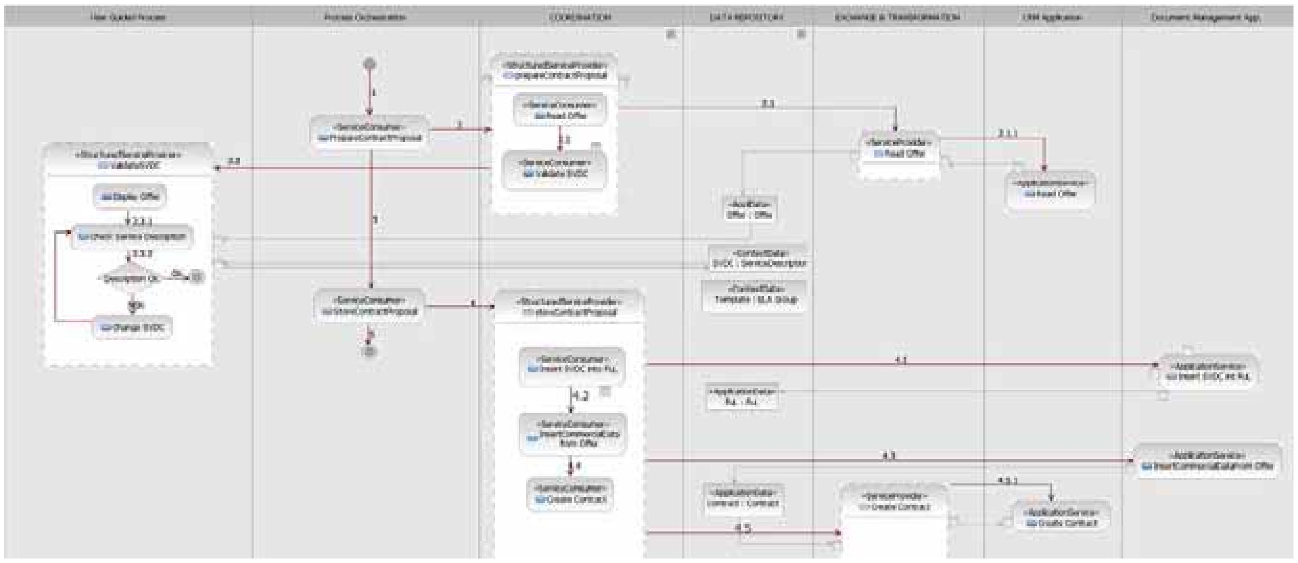


Figure 3. Create Contract Proposal Design

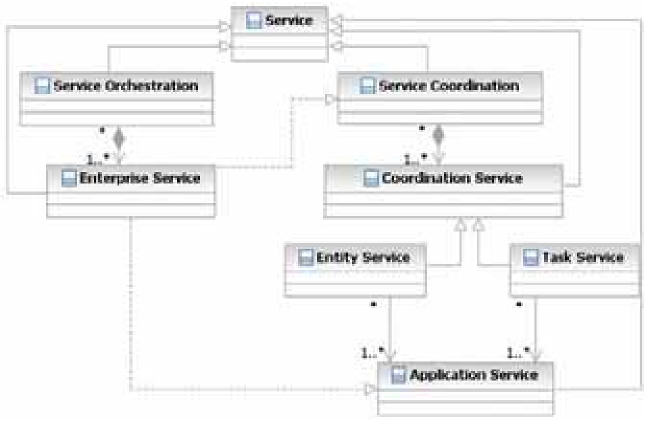


Figure 2. Service Hierarchy Meta-Model

services are not sufficient, though. Services also need to be employable in any arbitrary context. Thus, the services need to be loosely coupled. One aspect of that design principle is that services do not need to be aware of each other. One way to partly achieve this goal is to set-up service registries for a brokered communication among services. Services are completely unaware of other services whenever their dependence to these services is managed by a dedicated instance. This is the reason why the service coordination aggregates the single services without passing the control to other services.

Following the principles descibed so far, the platform-independent design of the entity services is as exemplified in Figure 3. The control-flow is numbered and marked in red. In the design shown, the CRM application is used as application service provider for the entity-specific coordination services *Read Offer* and *Store Contract*. A newly developed application for document management is required in order to support the task-specific services that manage the data manipulation (*Insert SVDC into FuL*[4] and *Insert Commercial Data from Offer*).

The next step of the service design is to categorize the identified task-services and the activity that is required to be performed in order to support the actual enterprise service. The modeled activities pose initial candidates for the services' operations. According to the underlying task or entity, the service is then completed by adding additional operations and eventually modifying the operation candidates extracted from the enterprise service model. The coordination services that can be identified to be supported by the CRM application system are shown in Figure 4. The operation candidates *Read Offer* and *Create Contract* were modified to the operations *callUpOffer* and *bookContract*, respectively.

In order to finish the design of the coordination services, the data model needs to be analyzed. This analysis reveals the data that are required to perform the operations of the services. Two messages are assigned to each operation - an input and an output message. In the presented use-case, the messages for the operations of the CRM-related services are exemplified by showing the message structure for the *bookContract* operation of the *ContractService* entity-specific service. As shown in Figure 5, the attributes of the service-inbound messages consist either of the necessary attributes to create a contract or a contract object itself.

Knowing the interfaces of the coordination services, the final step of the PIM is finishing the interface description of the enterprise services themselves. Since the data manipulation is performed by the included coordination services and the data is kept within the data repository, the enterprise service does not require input or output parameters at all. The business functions are clustered with regards to their business semantics into services while the single business tasks be-

[4]"FuL" stands for "Funktions- und Leistungsbeschreibung" and is a fixed term used in the company for "service description".

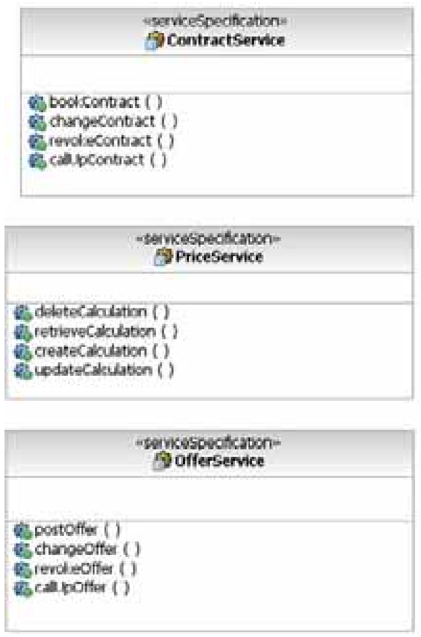


Figure 4. CRM Coordination Services

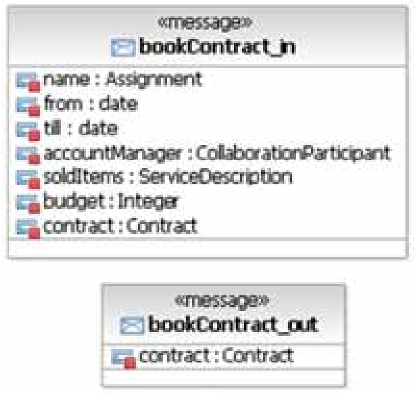


Figure 5. *bookContract* **Operation Messages**

come the operations of these services. All subsequent phases of the methodology depend on actual platform-dependent constraints (e.g how application services could be realized - both technically as well as functional). Hence, the later phases are considered designing the PSM which is out of the scope of this paper (cf. [8, 9]).

4. Conclusion and Future Work

We have presented the first application of our architectural concepts to a real-life business process. The case study shows that the concepts are applicable in real-life. Both, the methodology and the architecture are helpful when it comes to implementing the business process. In addition to it's guidance during the design phase, the concepts showed that they quite naturally lead to a distinct description of coordination services. This distinction is both, in line with accepted best-practices for application design as well as with more recent findings regarding service-design [6]. Currently we are involved with assigning the identified coordination services to the identified backend systems. So far it showed that only little functionality is remotely accessible yet but implementing it in a remote-accessible fashion seems achievable and efficient. The inherent heterogeneity issues identified up to now can be handled by the exchange and transformation layer. In parallel we are also busy to start the detailed implementation of the use-case. On the conceptual side we are validating the pattern language described in [8] in order to check for formal support of our approach. Formalizing the findings, we are currently working on a design metrics that will help to identify business cases that are suitable for the service-oriented paradigm and that will support the design decisions even more than the methodology, patterns and architecture presented here accomplish already.

References

[1] British Government Office of Government Commerce: ITIL IT-Infrastructure Library.

[2] *OASIS WSBPEL Technical Committee: Web Services Business Process Execution Language 2.0*, 31.02.2007.

[3] J. Beatty, H. Blohm, C. Boutard, S. Brodsky, M. Carey, and J.-J. Dubray. Service data objects for java specification. Technical report, BEA and SAP and IBM, 2005.

[4] K. P. Birman. Like it or not, web services are distributed objects. *Commun. ACM*, 47(12):60–62, 2004.

[5] J. Dehnert and W. M. P. van der Aalst. Bridging the gap between business models and workflow specifications. *Int. J. Cooperative Inf. Syst.*, 13(3):289–332, 2004.

[6] T. Erl. *Servcie-Oriented Architecture*. Prentice Hall, Inc., Upper Saddle River, NJ USA, 2005.

[7] D. S. Frankl. *Enterprise Patterns and MDA*. Prentice Hall, Inc., Addison-Wesley, 2004.

[8] H. Hofmeister and G. Wirtz. Approaching a methodology for designing composite applications integrating legacy applications using an architectural framework. In *Proc. GI-FG Treffen EMISA, Hamburg, LNI, Vol. 95, Springer*, 2006.

[9] H. Hofmeister and G. Wirtz. A Pattern Taxonomy for Business Process Integration Oriented Application Integration. In *Proc. 18th Intern. Conf. on Software Engineering and Knowledge Engineering, San Francisco Bay, USA*, 2006.

[10] H. Hofmeister and G. Wirtz. Using patterns to design composite applications. In *Intern. Conf. on Enterprise Information Systems and Web Technologies 2007, Orlando, FL; USA*, 2007.

[11] H. Reijers. A cohesion metric for the definition of activities in a workflow process. In *CaiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Desing (EMMSAD, 03). Velden, Austria.*, 2003.

[12] N. Russell, A. H. M. ter Hofstede, D. Edmond, and W. M. P. van der Aalst. Workflow data patterns: Identification, representation and tool support. pages 353–368, 2005.

[13] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[14] W. Vogels. Web services are not distributed objects. *IEEE Internet Computing*, 7:59–66, November/December 2003.

Synchronization of UML Based Refactoring with Graph Transformation

Y. Köşker and A. B. Bener, *Member, IEEE*

***Abstract*—Software engineers and researchers have been increasingly use refactoring in object oriented programming. There are different refactoring approaches in the literature. Refactoring of source code is the starting point of the researches and it is applied to different programming languages such as Small Talk, C, Java, etc. Refactoring is also applied to different software artifacts including the Unified Modeling Language (UML) class diagrams. Since, most of today's java development editors provide the synchronization of source code with the UML class diagrams, there are some open problems for the synchronization after refactoring is applied, such as the loss of inheritance links between two classes after renaming one of the classes. In this research we mainly focused on the issue of synchronization of the UML diagrams. We simulated the problem domain to examine Rename and Extract Interface. As a solution we have proposed a graph algorithm to maintain the inheritance links between classes after refactoring is applied. We have seen that, the graphical representation and transformation techniques fit our problem domain very well and it can be used as a solution for this domain.**

***Index Terms*—Refactoring, graph transformation, Unified Modeling Language, UML.**

I. Introduction

Since cost is the most important key factor for a software project to succeed, and the maintenance costs are contributed to a big part of the project costs, new technologies decrease the maintenance cost by providing efficient design and implementation patterns. As the software is enhanced, modified, and adapted to new requirements, the code becomes more complex and drifts away from its original design [11], hence lowering the quality of the software. Because of this, the major part of the total software development cost is devoted to software maintenance. The need for techniques that reduce software complexity by incrementally improving the internal software quality brings us to restructuring or specifically refactoring. The term *restructuring* is first introduced by Chikofsky and Cross [3] and defined as "*the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior (functionality and semantics).*" and continued as *"it does not normally involve modifications because of new requirements. However, it may lead to better observations of the subject system that suggest changes that would improve aspects of the system.*" Refactoring can be thought as the object-oriented counterpart of restructuring. The term *refactoring*, first introduced by Opdyke in his PhD thesis [6] and defined as "*the process of changing an [object-oriented] software system in such a way that it does not alter the external behavior of the code, yet improves its internal structure*" [7]. The earliest significant work on refactoring was the suite of C++ refactorings developed by William Opdyke [4]. It did however form the basis for the development of the Smalltalk Refactory Browser [6].

Refactoring increases its usage areas and it has significant importance for improving the design of existing code, especially with the advent of methodologies such as Extreme Programming [5] that involves little design and various iterations through the software development lifecycle.

It is important that the behavior preservation capability of a program must be satisfied. In other words, for the same input, the program must give the same output. However, some behavior aspects are mainly important for some software project types. For *real-time systems*, an essential aspect of the behavior is the execution time of certain (sequences of) operations. For *embedded systems*, memory constraints and power consumption are also essential aspects of the behavior [4]. Although refactoring have been applied primarily at the level of source code, the same techniques can also be used for a wide variety of different object oriented software artifacts: database schema; (UML) design models; software architectures; software requirements; executable code; test suites and many more-reference.

Unified Modeling Language (UML) can be used for finding smells which is described as "certain structure in code that suggest the possibility of refactoring" [9]. It also enables refactoring tasks to be done in large granularity. UML refactorings can be preferred instead of source code for some reasons [9]:

1. Many people are visually oriented and they like to be able to visualize the classes and their relationships

2. Being able to directly manipulate code at a higher level of

granularity (i.e. methods, variables, and classes rather than characters) can make refactoring more efficient. The efficiency increase depends on the ability to quickly grab and move the lines of codes in chunks that constitutes methods, classes, etc.

3. Being able to visualize code, specifically the content of classes and the relationships between them, can help in detecting smells.

The research in this paper is about the UML refactoring and the basic problems encountered with UML refactoring. The remainder of this paper is structured as follows: Section 2 defines the Unified Modeling Language (UML) and gives information about the previous related works. Section 3 gives a motivating example. Section 4 talks about the problem domain and the proposed solution. Finally, section 5 and 6 evaluates the proposed solution, draws some conclusions and talks about the future work.

II. RELATED WORK

UML is a language which gives a graphical representation of software projects in order to visualize and better understand them- reference. Today's programming tools provide an interface to build UML diagrams rather than building it from scratch. Also, they synchronize the code and UML diagrams according to the changes that have done. UML provides many different types of diagrams; however Class and Sequence diagrams are especially useful for the refactoring-reference. Class diagrams give a static view of the system (what classes make up the system, their contents, and their relationships), while sequence diagrams give a dynamic view of a specific sequence.

There has been much interest in refactoring recently, but little work has been done on tool support for refactoring or on demonstrating that a refactoring does indeed preserve program behavior [12]. Refactoring with UML is a relatively new has research area [9]. The detection of code smells and UML refactoring is discussed by many researchers- [9,10] However, they don't consider reconstruction and behavior preservation of the UML diagrams after refactoring is applied.. They use a UML tool called Together which bases it's class diagram directly on code, and allows you to manipulate the code by directly manipulating the diagram [9]. There are other researches on UML refactoring, but it is only based on the Java programming language [12].

Mens et.al. have done various researches about refactoring with graph transformations [1,2,4,8,11]. Although, their work does not focus on synchronization of UML diagrams through graphical representation, the outcomes of these researches are the milestones for refactoring. An interesting practical application of refactoring is used to detect inconsistencies in UML diagrams when the UML metamodel itself evolves which is achieved by the formalism on the level of type graphs [8]. However, the type graphs they are using are restricted to only one level, so it doesn't meet our problem domain when *Extract Interface* type refactoring is applied.

III. MOTIVATING EXAMPLE

A motivating example about the computer hierarchy is given. The codes are implemented using Java language in an object-oriented manner and Oracle JDeveloper 10g is used as the Java editor. Java is selected since it is usage areas are growing and the JDeveloper editor is selected because it is an open-source free editor which makes is possible to add some additional features as it will be discussed in the future work part. The code is kept as simple as possible to make it easier to understand. On the other hand, it covers most of the interesting constructs of the object-oriented software paradigm.

A. Computer Hierarchy Example

There are six classes in the example namely *Pc, Hp, Dell, CanPlay, CanRecord, MException* which are located in the *project* package. *Pc* is the abstract class where *Dell* and *Hp* are extended from. *CanPlay* and *CanRecord* are the interfaces where *Dell* and *Hp* are implemented from. *MException* is extended from the *Exception* class which is defined in *java.lang* class. Below is the some sample java codes. Figure 1 shows the UML Class Diagram of the example java implementation.

```
public  abstract class Pc{
   int screenSize = 14;
   public int getScreenSize(){
      return screenSize;
   }
   public abstract int getAge();
}
public class Hp extends Pc implements CanPlay,
CanRecord{
   int age = 0;
   public int getAge(){
      return age;
   }
   public void Record(){
      System.out.println("Recording..");
   }
   public void Play(){
      System.out.println("Playing..");
   }
   public int CalculateCost(int basecost, int discount) throws
MException{
      int result = 0;
      try{
         result = basecost / discount;
      }
```

```
        catch(Exception e){
           throw new MException();
        }
        finally{
           System.out.println("Test");
        }
        return result;
     }
  }
  public interface CanPlay{
     public void Play();
  }
  public interface CanRecord{
     public void Record();
  }
  public class MException extends Exception{
     public MException(){
        super("I catch error");
     }
  }
```

B. *Selected Refactoring*

1) Rename:

This is the simplest refactoring type but generally the most used one. Since, the renaming ability is needed any time; this feature enables users to make this change in automatic manner.

2) Extract Interface:

This refactoring type is selected especially for the purpose of the examination of UML and source code consistency after the refactoring is applied.

Here is the class diagram after the two selected refactorings are applied to the class diagram. The source code is changed according to the change made by the refactoring applied to the UML diagram. The Extract Interface refactoring is applied to *CanRecord* class and the interface is created with the name *ExtractedInterface*.
Moreover, the Rename refactoring is applied to *Pc* class and it is renamed to *Computer*. Figure 2 shows the synchronized class diagram after the refactoring.

Since, there exists problems with the synchronization which will be discussed detailed in section 4; the expected synchronized class diagram is given after applying the manual change parts.

IV. PROBLEM DOMAIN AND PROPOSED SOLUTION

Construction of UML diagrams from source code and generation of source code from UML diagram is supported by today's editor tools such as Eclipse, JDeveloper. Moreover, the synchronization of the source code with UML class diagrams and vice versa is supported. Refactoring of UML class diagrams is an emerging research topic and heavily inspired by refactoring of program code written in object-oriented implementation languages [10]. Only few tools are currently available to support UML refactoring. Since UML refactoring is a growing research area and selected as the refactoring method in our approach, we have investigated the existence of fully automated synchronization.

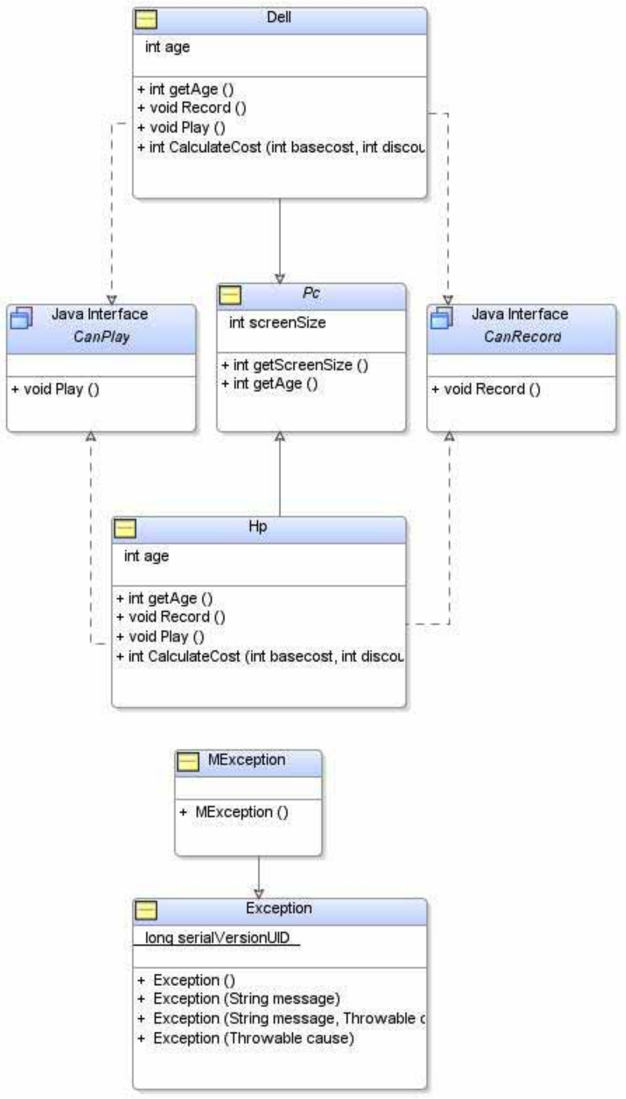


Fig. 1. Class Diagram of the Computer Hierarchy example.

A. *Problem Definition*

After the two selected refactoring types are applied, one can easily see that the synchronization of the UML diagrams is not supported. The inheritance links between classes are lost and one has to add the class diagram of the inherited classes manually. For small software projects or for the ones which are not object-oriented, this may not be a major problem. However, for complicated and granular applications manual updates would be costly and error prone.

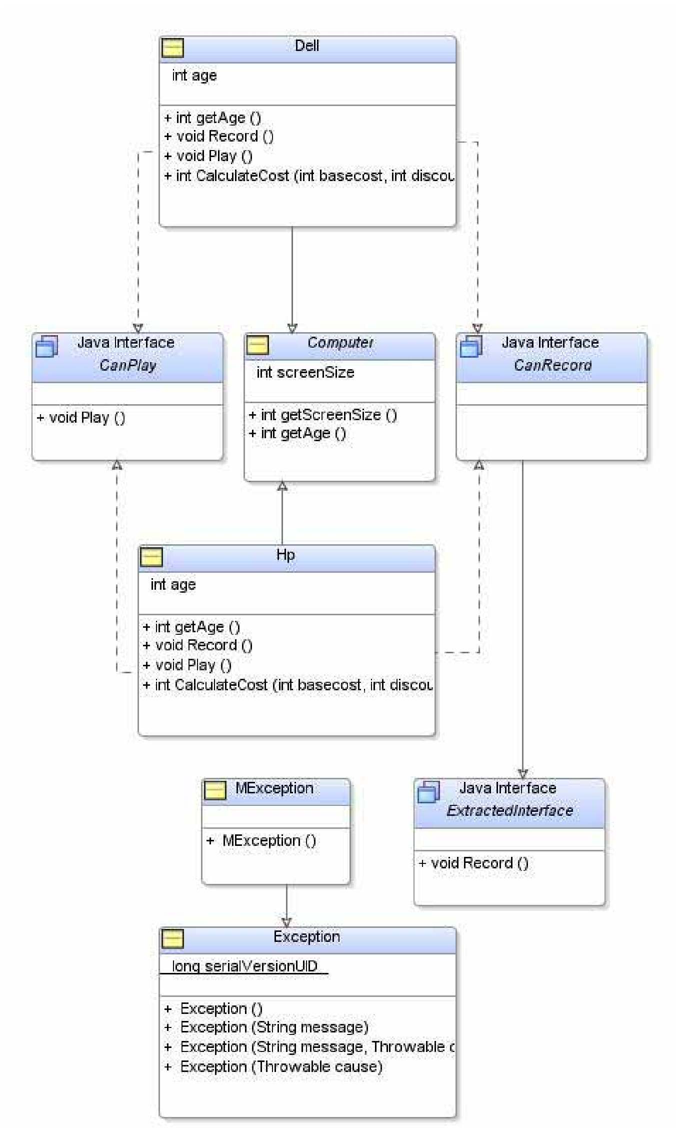


Fig. 2. Class Diagram of the Computer Hierarchy example after applying the Extract Interface and Rename type refactorings.

B. *Synchronization of Refactoring by Graph Approach*

There is a direct correspondence between refactorings and graph transformations. Programs (or other kinds of software artifacts) can be expressed as graphs, refactorings correspond to graph production rules, the application of a refactoring corresponds to a graph transformation, refactoring pre and postconditions can be expressed as application pre and postconditions [13], [15].

Software entities (such as classes, variables, methods and method parameters) are represented by *nodes* whose label is a pair consisting of a name and a node type [1]. Relationships between software entities (such as membership, inheritance, method lookup, variable accesses and method calls) are represented by *edges* between the corresponding nodes [2].

Table 1 is taken from [11] and summarizes some formal properties of graph transformation that may be used to address important issues in refactoring.

Refactoring	Graph transformation
software artifact	graph
refactoring	graph production
composite refactoring	composition of graph productions
refactoring application	graph transformation
refactoring precondition	application precondition
refactoring postcondition	application postcondition
(in)dependence between refactorings in a sequence	parallel or sequential (in)dependence
conflict betweeen refactorings applied in parallel to the same software artifact	confluency and critical pair analysis

Table. 1. Correspondence between Refactoring and Graph Transformation given by [11].

However, there is no work done on synchronization over UML class diagrams with graph transformation. Our proposed solution involves using graph transformation for the synchronization of UML class diagrams.

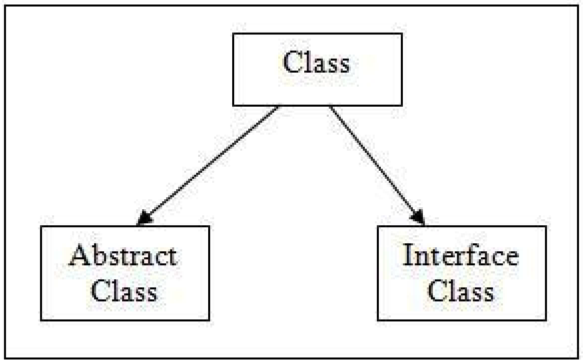


Fig. 3. The Graph representation for the java classes

When a class extends from another class or implements another class, it can be turned into a graphical representation that is shown in Figure 3. The rectangles here stand for the classes and the edges stand for the inheritance. In order not to lose that link in the UML class diagram after refactoring, we use a simple Graph Algorithm.

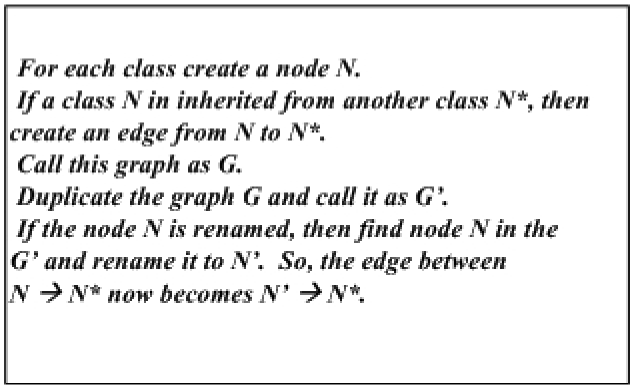
C. *Graph Algorithm*

The behavior is preserved after the refactoring is applied. On the other hand, preserving the graphical structure does not successfully work. In order to provide this feature, we give a simple algorithm here.

1) *Rename:*

After renaming a class, we expect that nothing will change in the UML class diagram except the name of the class. Hence, we expect that the links (edges for the graphical representation) that denote inheritance remains unchanged, but they do not.

So, the algorithm that will be used in order to turn graph G to G' by renaming the node from N to N' is shown below- Figure 4. Figure 5 shows the graph representation of the before and after Rename type refactoring is applied.



For each class create a node N.
If a class N in inherited from another class N*, then create an edge from N to N*.
Call this graph as G.
Duplicate the graph G and call it as G'.
If the node N is renamed, then find node N in the G' and rename it to N'. So, the edge between N → N* now becomes N' → N*.

Fig. 4. The Graph Algorithm for the Rename Type refactoring

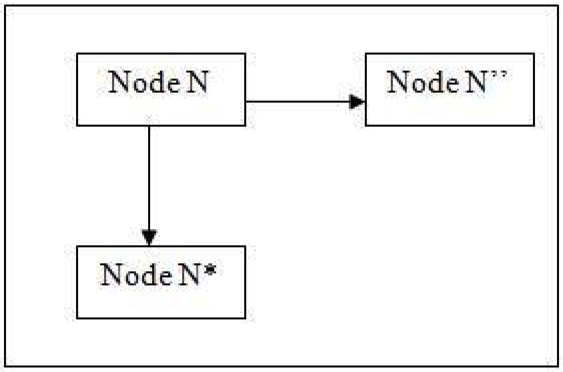


Fig. 5.a. The Graph representation before refactorings are applied

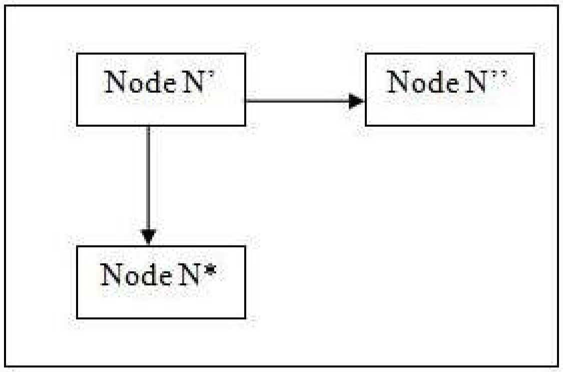
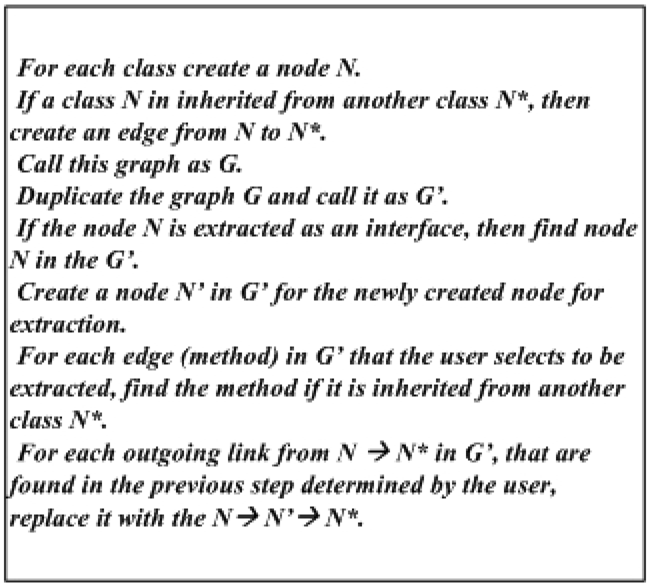


Fig. 5.b. The Graph representation after Rename Type refactoring



For each class create a node N.
If a class N in inherited from another class N*, then create an edge from N to N*.
Call this graph as G.
Duplicate the graph G and call it as G'.
If the node N is extracted as an interface, then find node N in the G'.
Create a node N' in G' for the newly created node for extraction.
For each edge (method) in G' that the user selects to be extracted, find the method if it is inherited from another class N*.
For each outgoing link from N → N* in G', that are found in the previous step determined by the user, replace it with the N→ N'→ N*.

Fig. 6. The Graph Algorithm for the Extract Interface Type refactoring

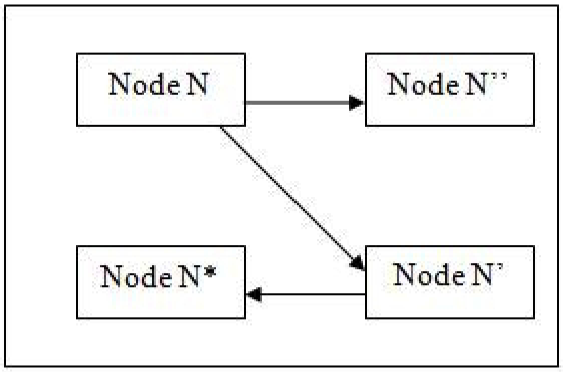


Fig. 7. The Graph representation after Extract Interface Type refactoring

2) Extract Interface:

Extract Interface type refactoring is more complicated than *Rename*. When the user selects this type of refactoring over a class, then it is expected that the newly extracted interface will be added to the UML class diagram and necessary links will be updated accordingly.

So, the algorithm that will be used in order to turn graph G to G' by extracting the node from N to N' is shown above- Figure 6. Figure 5.a and 7 show the graph representations of the before and after Extract Interface type refactorings are applied, respectively. Note that the edge (method) which is inherited from the N'' isn't changed since it's assumed that the user doesn't want to extract it to the newly created node N' (class). However, the user wants to extract the method for the class N*.

V. Conclusion

In this paper, we have examined Refactoring which we believe it is one of the most important challenges of Software Engineering development methodology. Refactoring based on UML class diagrams has been our starting point. Our main motivation was the lack of synchronization ability of today's Java Editors (Oracle JDeveloper 10g is used as the Java Editor). We concentrate on two refactoring types namely *Rename* and *Extract Interface*. In order to preserve the inheritance of classes for UML class diagrams after refactoring, we propose a graph algorithm. The idea behind the graph representation is based on representing the classes as nodes and the inherited methods or variables as edges. We develop an efficient algorithm by the graph representation before the refactoring is applied and transform it to a new graph which represents the program after it is refactored.

VI. Future Work

As a future work, we would like to investigate on other refactoring types such as Pull Members Up, Pull Members Down, and Extract Superclass. Also, we would like to improve the algorithm and apply it to an open source Java Editor in order to make the system more robust. Another future direction would be to examine the problem domain for other programming languages (rather than Java) and for other programming editors (rather than JDeveloper). A performance tuning for the graph algorithm can also be done and the algorithm can be extended to other types of refactoring.

Acknowledgment

This research is supported in part by Bogazici University research fund under grant number BAP-06HA104.

References

[1] T. Mens, S. Demeyer, and D. Janssens, "Formalising Behaviour Preserving Program Transformations", Proceedings of Graph Transformation: First International Conference, ICGT 2002, Barcelona, Spain, October 7-12, 2002.

[2] T. Mens, N. V. Eetvelde, D. Janssens, and S. Demeyer, "Formalising Refactorings with Graph Transformations", Journal of Software Maintenance and Evolution, 2004, pp.1001–1025.

[3] W. F. Opdyke, "Refactoring: A Program Restructuring Aid in Designing Object-Oriented Application Frameworks". PhD thesis, University of Illinois at Urbana-Champaign, 1992.

[4] T. Mens and A. van Deursen, "Refactoring: Emerging Trends and Open Problems", Proceedings of First International Workshop on REFactoring: Achievements, Challenges, Effects (REFACE), University of Waterloo, 2003.

[5] K. Beck, *Extreme Programming*. Addison Wesley Longman, Massachusetts , 2000.

[6] M. Fowler, *Refactoring: Improving the Design of Existing Programs*. Addison-Wesley, NY, 1999.

[7] E. J. Chikofsky and J. H. Cross, "Reverse engineering and design recovery: A taxonomy", *IEEE Software*, 7(1):13–17, 1990.

[8] T. Mens, "Conditional Graph Rewriting as a Domain-Independent Formalism for Software Evolution", Proceedings of Applications of Graph Transformations with Industrial Relevance: International Workshop, AGTIVE'99, Kerkrade, The Netherlands, September 1999.

[9] D. Astels, "Refactoring with UML". Proc. 3rd Int'l Conf. eXtreme Programming and Flexible Processes in Software Engineering, 2002, pp. 67-70.

[10] S. Marković and T. Baar, "Refactoring OCL Annotated UML Class Diagrams", Proceedings of Model Driven Engineering Languages and Systems, 8th International Conference, MoDELS 2005, Montego Bay, Jamaica, October 2-7, 2005, Vol. 3713, 2005, pp. 280-294.

[11] T. Mens and T. Tourwe, "A Survey of Software Refactoring", *IEEE Trans. Software Engineering*, vol. 30, no. 2, February 2004.

[12] O'. M. Cinneide and P. Nixon, "Composite Refactorings for Java Programs", Proceedings of the Workshop on Formal Techniques for Java Programs, European Conference on Object-Oriented Programming, 2000.

[13] R. Heckel, "Algebraic Graph Transformations with Application Conditions", MS thesis, TU Berlin, 1995.

[14] D. Roberts, J. Brant, and R.E. Johnson, "A Refactoring Tool for Smalltalk," Theory and Practice of Object Systems, vol. 3, no. 4, pp. 253-263, John Wiley & Sons, NY, 1997.

[15] A. Habel, R. Heckel, and G. Ta¨ntzer, "Graph Grammars with Negative Application Conditions," Fundamenta Informaticae, vol. 26, no.3, pp. 287-313, June 1996.

Y. Köşker is a Computer Engineering graduate student at Bogaziçi University. She has her undergraduate degree in Computer Engineering department at Boğaziçi University in 2005. Her research interests are Defect Prediction and Cost/Effort Estimation. She is a BI consultant now and has 3 years of working experience. Contact her at Boğaziçi University, Istanbul, Turkey, yasemin.kosker@cmpe.boun.edu.tr

A. B. Bener is a faculty member in the Department of Computer Engineering at Bogazici University. Her research interests include Web services, security, e-commerce, and m-commerce applications and software engineering. Bener has a PhD in information systems from the London School of Economics. She is a member of the IEEE, the IEEE Computer Society, and the ACM. Contact her at Boğaziçi University, Istanbul, Turkey, bener@boun.edu.tr

Using Formal Composition of Use Cases in Requirements Engineering

Rabeb Mizouni
ECE Department
Concordia University
Montreal, Canada
mizouni@ece.concordia.ca

Aziz Salah
Computer Science Department
UQAM University
Montreal, Canada
aziz.salah@uqam.ca

Rachida Dssouli
CIISE Institute
Concordia University
Montreal, Canada
dssouli@ciise.concordia.ca

Abstract

Use Case techniques are widely used to capture software requirements. The current tendency is to keep such use case models as understandable and simple as possible. This simplicity is a barrier to make use cases more accurate and may lead to incorrect and inconsistent system specifications. A formal and expressive model may help the modeler to express her/his needs in a more intuitive way. In this paper we propose an approach for generating an overall system specification using use cases. Each use case represents a partial system behavior described as an extended finite automaton. We develop an automated and incremental approach which aims at merging use cases. We define imperative expressions that specify the semantics of the composition to perform. In each increment, the specification is augmented by the set of use cases generated by composition. The approach is illustrated by an e-Purchasing system case study.

1 Introduction

Requirements engineering is a critical phase in system development process. It is the phase where the needs of customers have to be unambiguously speciÞed in order to make easier the process of getting the right system. It has been proved that system behavioral models are very important to system requirements validation. However, such models are hard to construct and need expertise. Use case-based approach facilitates the process of generating the intended behavioral model by describing different system functionalities separately. Consequently, each use case may depict the behavioral model of a system-level functionality. The composition of these use cases should conform to the system behavioral model. This model can further be used as a speciÞcation of the system for checking of model compliance to properties and correctness.

Despite the consensus on their usefulness, use case based approaches suffer from a lack of formality. This formality would have facilitated the development of automated composition approaches and tools that help building reliable system behavioral models. In order to meet this challenge, a use case based approach has to state the level of mechanization of its composition methods, and consequently, Þnd the appropriate formalization level of use case representation model.

Our notion of a formal use case is based on its representation as an extended Þnite state automaton. Once the analyst has described the different system-level functionalities by delivering a certain number of use case extended automata (UCEAs), she/he can state imperative composition expressions which describe how to compose use cases using operators. The evaluation of a composition expression is a new behavior which results from inserting one UCEA, called referred use case, into the other one, called base use case, according to the template of the composition expression operator. The new UCEA is, therefore, added to the original UCEAs initially speciÞed and can further be used in next increments of the system speciÞcation synthesis.

The contribution of this paper is mainly extending our previous work in [10] by using a variant of the extended Þnite automaton model, called UCEA, in which states are extended by values of UCEA variables, and transitions are guarded by pre-conditions and have assignments to deÞne the UCEA variables. We propose new composition operators which are more adapted to compose UCEAs. Introducing variables allows to fold many transitions into a single transition of the UCEA. Consequently, UCEAs are more concise and scalable models.

The paper is structured as follows. In section 2, we present the formal deÞntions of the UCEA model and the label matching based composition of two UCEA. In section 3 the composition operators and expression syntax are presented. Section 4 describes the composition method. Our approach is illustrated through the presentation of the e-Purchasing system case study. Discussions on related works are given in Section 5. Finally, Section 6 concludes the paper and shows current directions of future works.

2 Formal Use Case Model

A use case is used to describe a functionality or a part of the system behavior regarding a certain concern. An extended Þnite state automaton model is a formal, suitable and intuitive model for describing the behavior of a use case. The transitions connecting states in the extended Þnite state automaton play the role of the use case actions and represent, on one hand, the control ßow. On the other hand, the variables of the extended Þnite state automaton represent the data ßow.

A UCEA is a 7-tuple $(S, s^o, S^f, L, V, I, E)$ such that: S is the set of states, s^o is the initial state, $S^f \subseteq S$ is the set of Þnal states, L is the set of labels, V is the set of variables, $I \subseteq V$ is the set of input variables, and Þnally $E \subseteq S \times C \times L \times A \times S$ is the transition relation such that: C groups the set of preconditions on variables and A is the set of variable assignments. The precondition of a transition has to be true before the transition is enabled. Its variable assignments play the role of a post-condition.

The set of input variables represents the variables that can be initiated when inserting the UCEA into another one. For a transition $(s, c, l, a, s') \in E$, we also write it $s \overset{c,l,a}{\to} s' \in E$. We deÞne c as a list of $(v_i \# v)$ where $v_i \in V, v \in dom(v_i)$, and $\#$ is a binary relation. In contrast, a is deÞned as a set of assignments that can be either $(v_i := v_j)$, $(v_i := v_i\ op\ const)$, or $(v_i := const)$, where v_i and v_j are variables. Once the system is in s, if the precondition c is true, the transition $s \xrightarrow{c,l,a} s'$ is Þred, and then the variables are updated by the execution of the assignments in a. By default, the value of any variable in V is equal to $null$. $null$ denotes the fact that at this stage, the variable is not initiated yet to any value.

In order to compose UCEA, we advocate an automated and formal method. For this purpose, we deÞne the notions of *extension point* and *label matching composition*. *Extension point* represents a location, either a state or a transition, which is used to identify where another use case can be inserted. The *label matching composition* is a generic composition method of two UCEAs. Such method will be used later in order to build UCEAs of composition expressions. We deÞne the label matching based composition of two UCEAs $U_1 = (S_1, s^o{}_1, S^f_1, L_1, V_1, I_1, E_1)$ and $U_2 = (S_2, s^o{}_2, S^f_2, L_2, V_2, I_2, E_2)$ where $V_1 \cap V_2 = \emptyset$ as an UCEA $U = (S, s^0, S^f, L, V, I, E)$ where $S \subseteq S_1 \times S_2$ such that all the states of S are reachable from s^o in the graph of the resulting composed UCEA, $s^o = (s^o{}_1, s^o{}_2) \in S$, $S^f \subseteq (S^f{}_1 \times S_2) \cup (S_1 \times S^f{}_2)$, $L \subseteq (L_1 \cup L_2)$, $V = (V_1 \cup V_2)$, $I \subseteq (I_1 \cup I_2)$, $E \subseteq S \times C \times L \times A \times S$ where E and S are inferred by the following rules:

$$\frac{(s_1, s_2) \in S; s_1 \xrightarrow{c,l,a} s'_1 \in E_1; s_2 \xrightarrow{c,l,a} s'_2 \in E_2}{(s'_1, s'_2) \in E; (s_1, s_2) \xrightarrow{c,l,a} (s'_1, s'_2) \in S} \quad (1)$$

$$\frac{(s_1, s_2) \in S; s_1 \xrightarrow{c,l,a} s'_1 \in E_1; s_2 \xrightarrow{c,l,a} s'_2 \notin E_2}{(s'_1, s_2) \in E; (s_1, s_2) \xrightarrow{c,l,a} (s'_1, s_2) \in S} \quad (2)$$

$$\frac{(s_1, s_2) \in S; s_1 \xrightarrow{c,l,a} s'_1 \notin E_1; s_2 \xrightarrow{c,l,a} s'_2 \in E_2}{(s_1, s'_2) \in E; (s_1, s_2) \xrightarrow{c,l,a} (s_1, s'_2) \in S} \quad (3)$$

Rule (1) states that when the labels, the preconditions and the assignments of two transitions of the two UCEAs are the same, these two transitions are merged into a single one in the composed UCEA. Otherwise, as stated in Rules (2) and (3), each transition is represented separately in the composed UCEA. This deÞnition is an extension of the definition of label matching in [10] where we have introduced variables. We can extend this deÞnition to the case of n UCEAs.

3 UCEA composition Expression

3.1 Composition Operators

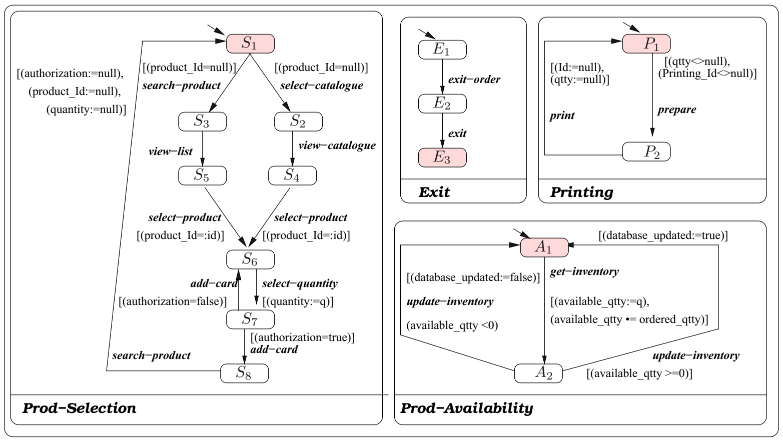
Operators deÞne templates for the composition of UCEAs and allow the derivation of a new behavior from two existing ones. We propose three composition operators: $Include$, $Extend\_with$, and $Alternative$.

In the case of $Include$ operator, the resulting UCEA is composed of the behavior of the base UCEA where we insert at the extension point the behavior of the referred UCEA. The behavior of the base UCEA would resume from the extension point. With this operator, some traces of the base use case might be modiÞed. They represent the set of traces that cross the extension point, where the traces of the referred use case are inserted.

The $Extend\_with$ operator is similar to $Include$ because in both of them after the execution of the referred UCEA at the extension point, the behavior of the UCEA of the expression resumes in the base UCEA. However, the UCEA of an expression using $Extend\_with$ makes it optional the execution the referred UCEA at the extension point while it is mandatory with $Include$.

The $Alternative$ operator can also be called $Branching$. The UCEA of an expression using the $Alternative$ operator is composed of the behavior of the base UCEA and the behavior of the referred UCEA as an alternative behavior in the extension point.

In this paper, we are presenting our approach in the case of these three operators, however, our approach is not limited to them and the same methodology can be applied in order to design other composition operators.



S_1
[(authorization:=null), (product_Id:=null), (quantity:=null)]
[(product_Id=null)] search−product
[(product_Id=null)] select−catalogue
S_3
S_2
view−list
view−catalogue
S_5
S_4
select−product [(product_Id=:id)]
select−product [(product_Id=:id)]
S_6
add−card [(authorization=false)]
select−quantity [(quantity:=q)]
S_7
[(authorization=true)] add−card
search−product
S_8
Prod−Selection
E_1
exit−order
E_2
exit
E_3
Exit
P_1
[(Id:=null), (qtty:=null)]
print
[(qtty<>null), (Printing_Id<>null)]
prepare
P_2
Printing
A_1
[(database_updated:=true)]
[(database_updated:=false)]
update−inventory
(available_qtty <0)
get−inventory
[(available_qtty:=q), (available_qtty •= ordered_qtty)]
update−inventory
[(available_qtty >=0)]
A_2
Prod−Availability

Figure 1. Use Cases of the e-Purchasing System

3.2 Example : UCEAs of an e-Purchasing System

In order to illustrate our approach, we will be using a set of UCEAs of an e-Purchasing system which allows online orders. The requirements of the e-Purchasing system cover a wide range of use cases. We are restricting our case study to use cases of the purchaser side only. e-Purchasing requires a number of activities to be performed. First the customer has to select a product. She/he either consults the catalog list or makes a search by the name of the product in available catalogs. After checking the availability of the selected product, he can print a quote. He also can cancel the order. Figure 3 depicts some UCEAs. For each of these UCEAs, we deÞne the set of variables and we specify the set of input variables in table 1.

Use Case	Variables	Input Variables
Prod_Selection	$\{prod\_Id, quantity, authorization\}$	$\emptyset$
Prod_Availability	$\{Id, available\_quantity, \}$ $database\_updated, ordered\_quantity\}$	$\{Id, ordered\_quantity\}$
Printing	$\{qtty, Printing\_Id\}$	$\{qtty, Printing\_Id\}$
Exit	$\emptyset$	$\emptyset$

Table 1. Variables of the e-Purchasing Use Cases

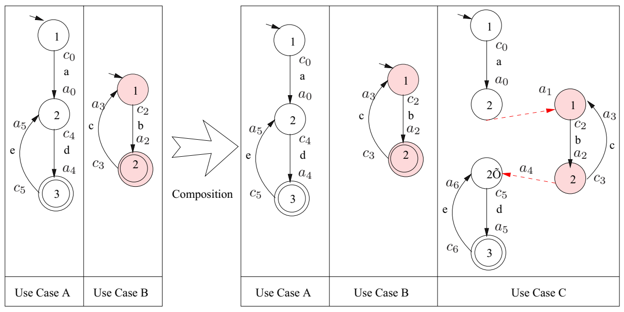
In the Þrst use case, $prod\_Id$ deÞnes the identiÞer of the product the customer has chosen. The variable $quantity$ stores the quantity of the product the customer is ordering. Finally, $authorization$ is a Boolean variable which keeps track whether the quantity asked by the customer is available in the inventory or not. $Prod\_availability$ UCEA has four variables: Id represents the identiÞer of the product for which availability would be checked, $available\_quantity$ returns the quantity that Þgures out in the database, $ordered\_quantity$ indicates the quantity needed, and Þnally $database\_updated$ indicates if the quantity in the database has been updated by the new available quantity. The $Printing$ use case has two input variables, $qtty$ and $Printing\_Id$, that indicate the identiÞer and quantity of the product asked by the user, respectively.

3.3 UCEA Composition Expressions Definition

An UCEA composition expression speciÞes the way a new behavior (a UCEA) is synthesized by composing two existing UCEAs. A composition expression is formed from Þve elements: two UCAEs, an extension point, and an input and output sets of assignments of variables. The two UCEAs of an expression have different roles: a base use case role and a referred use case role. The UCEA expression speciÞes where a referred use case is inserted in the base one as an extension point. Since we consider that variables are deÞned locally in the UCEAs, it is necessary to deÞne a mapping between the variables of the base and the referred use cases. This is speciÞed within the UCEA expression in the $Input\_Var\_Assign$ and the $Output\_Var\_Assign$ Þelds. LetÕs consider the case of this expression:

$$\begin{aligned} Z := \quad & Include(Prod\_Selection, Prod\_Availability) IN\ (S_7) \\ & [(Id := prod\_Id), (ordered\_quantity := quantity)] \\ & [(authorization := database\_update)] \end{aligned} \tag{4}$$

Z represents the UCEA that will be generated from the valuation of the expression. $Prod\_Selection$ is the base UCEA and $Prod\_availability$ is the referred one. $IN\ (S_7)$ represents state where the composition will be performed. When inserting, we have to assign to the input variables of the referred UCEA Id and $ordered\_quantity$ the values of the variables $prod\_Id$ and $quantity$ of the base UCEA, respectively. At the end of the referred UCEA, the variable



(a) SpeciÞcation Current Increment (b) SpeciÞcation Next Increment

$$C := \quad Include(A, B)\ IN\ (2)\ [a_1]\ [a_4]$$

Figure 2. System Specification Synthesis: Approach Description

$authorization$ of the base UCEA is assigned to the value of the variable $database\_update$ of the referred UCEA.

3.4 Approach Overview

When evaluating a composition expression, we aim at generating a new behavior (represented as a UCEA) where the behavior of the referred use case is merged in the behavior of the base use case in the extension point. In fact, informally, it is a cut and paste operation where the referred UCEA is copied and pasted in the extension point according to the semantics of the operator expressed in the composition expression. To avoid undesired scenarios, complete traces of the referred use case are inserted in the extension point of the base use case and insertion of a part of the trace is not allowed.

LetÕs consider the example in Figure 2. The modeler starts by specifying two use cases A and B, as well as a UCEA expression C. C is a use case where the behavior of the UCEA B is included in the behavior of the UCEA A in state 2 with the input variable assignment a_1 and the output variable assignment a_4. The UCEA C that we aim at generating automatically is represented in Figure 2 (b). The use cases A, B, and C represent the new increment of the system speciÞcation. They represent the set of use cases that the modeler can use in order to specify new behaviors.

4 UCEA Composition Approach

The insertion of a UCEA into another is performed with respect to the semantics of the composition operator of the expression. In our approach, this insertion is achieved through the label matching-based composition. We developed state based patterns generated from the base and the referred UCEA respectively that shows the semantics of the composition the analyst speciÞed. They reßect the semantics of the composition operator in the extension point. We deÞne a pattern for each of the three operators we previously mentioned. We illustrate these patterns through the e-purchasing UCEAs. In addition to the composition expression 4, we deÞne the UCEA Y such that:

$$\begin{aligned} Y := \quad & Extend\_with(Prod\_Selection, Printing)\ BEFORE \\ & (s_6, (authorization = true), add\_card, true, s_7) \\ & [(Printing\_Id := prod\_Id), (qtty := quantity)]\ [] \end{aligned} \tag{5}$$

It speciÞes that the UCEA Y is obtained by inserting $Printing$ in $Prod\_Selection$ UCEA. $Printing$ is optional since the composition operator used is $Extend\_with$. We also deÞne the UCEA W obtained by inserting $Exit$ in $Prod\_Selection$ UCEA. ItÕs an alternative to the execution of the $Prod\_Selection$ UCEA at the extension point. The composition expression is as follows:

$$\begin{aligned} W := \quad & Alternative(Prod\_Selection, Exit)\ AFTER \\ & (S_6, true, select\_quantity, (quantity := q), S_7) \\ & []\ [] \end{aligned} \tag{6}$$

4.1 State-Based Patterns

To show our composition approach, we propose to generate the state-based patterns needed to derive the UCEA Z, Y and W. Figure 3(a) illustrates an example of a synthesized state based pattern for the use case $Prod\_Selection$ using the expression of Z (Expression 4). Two states were added, q and qÕ. Two transitions labeled with$begin$ and end are added in order to indicate the starting point and the ending point of the referred UCEA. These transitions have to be decorated with the variables assignments the analyst asked for. The input assignment indicates the assignment to perform before starting the referred UCEA. Hence, it plays the role of postcondition of the transition we added labeled with $begin$. In the same way, the output assignment indicates the assignment to perform before resuming back to the base UCEA. Hence, it plays the role of postcondition of the transition labeled with end. Figure 3(b,c) shows examples of the state base patterns in the case of $Extend\_with$ operator and $Alternative$ operator respectively. We note that in the case of $Alternative$, the state base pattern shows the fact that there is no resumption to the base UCEA after starting the referred one, which is not the case of the two other state-based patterns.

In the same way, since the referred UCEA has to be inserted in the UCEA using the label matching, a state based pattern for the referred UCEA has to be deÞned. It is independent from the extension point and the composition operator. Figure 4 illustrates an example of a synthesized state

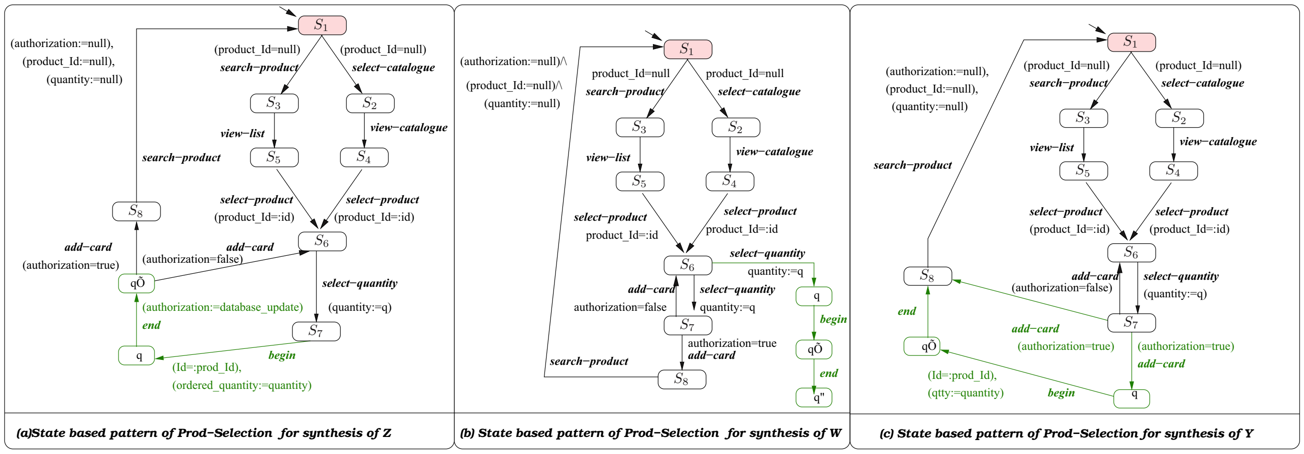


Figure 3. Illustration of State-Based Patterns for Base UCAs

based pattern for UCEA $Printing$ as the referred UCEA in the expression of Y. We note that two transitions labeled with $begin$ and end are also added here.

4.2 UCEA Generation

When applying the composition approach to the state-based patterns of the base and referred UCEAs, we derive a new UCEA that contains transitions labeled with $begin$ and end. These transitions were not originally speciÞed. They were added to construct the state-based patterns of the different operators and to avoid the insertion of unexpected scenarios. The removal of these transitions labels consists of a backward propagation of the postconditions (the assignments) of $begin$ and end labeled transitions. This propagation concerns the ingoing transitions toward the source states of the transitions $begin$ and end. The postconditions of these transitions (the ingoing transitions of the outgoing state of $begin$ and end) will be augmented by the postcondition of $begin$ and end respectively. Then, these transitions are considered as ϵ-transitions and further removed using the algorithm of ϵ removal in [9].

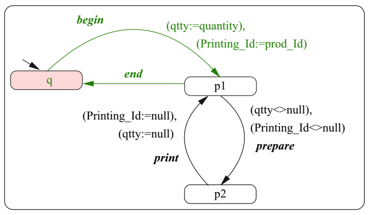


Figure 4. State-Based Pattern for referred UCEA

As a last step of our composition approach, the Þnal states of the obtained UCEA of an expression have to be deÞned. They depend on the composition operator of the expression and the extension point where the referred use case has been inserted. LetÕs take as an example the case of $Include$ composition operator where the extension point is different from the Þnal states of the base UCEA, then the Þnal states of UCEA of the expression are the composite states containing one of the Þnal states of the base UCEA.

Figure 5 shows the newly generated use case Y. It can be further used in the description of other UCEA expressions in the next increment of the speciÞcation synthesis.

5 Related Work

Because use cases are used in an early state of system development, they need validation. This fact motivated the development of several approach that generated state-based model from use cases[3, 7, 8, 13, 14, 15]. During the composition of use cases into state-based system, the challenge is to identify states at the scenario level that serve as extension points between use cases. There exist two types of state characterization: trace-based [7, 8], and variable (or label) state-based characterization [14, 12, 2]. On the other hand, some classical composition operators exist on the literature such as sequential and alternative concatenation [1] that are used to compose use cases. They consider use cases as entities. In this paper, we added another level of granularity by allowing the explicit speciÞcation of an extension point as a transition or a state of the base UCEA.

On the other hand, many notations [3, 1, 6] have described use cases with different degrees of expressiveness and formality. Glinz [5] uses statecharts to model scenarios. The integration of scenarios is performed in a way to retrieve the relationship between scenarios by keeping their internal structure unchanged, and to detect inconsistencies. The approach proposed carries only the composition of disjoint scenarios with elementary constructors. As an exten-

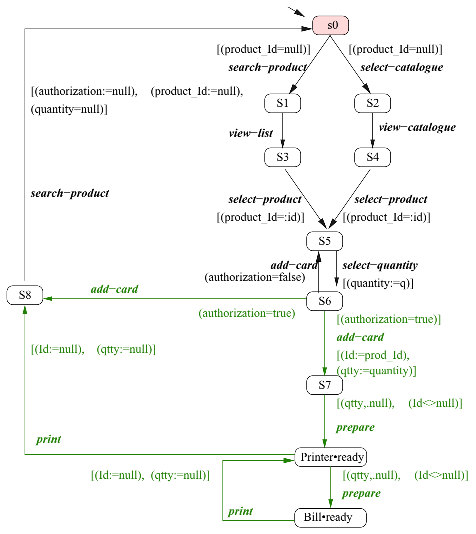


Figure 5. The synthesized UCEA Y

sion of this work, Ryser [11] introduces a new kind of chart and notation to model dependencies among scenarios. The advantage of this approach is the fact of capturing clearly these inter-scenarios dependencies. Bordeleau et al. [4] have proposed integration patterns for scenario dependencies, detected using UCMs. A state-based speciÞcation per use case is generated for each component and integrated to reßect the scenarios dependencies. The whole process is done manually and relies on the creativity of the analyst to connect together the different statecharts in the right way. Our approach has the advantage of being fully automated: the analyst will only specify the set of composition expressions in order to generate a UCEA representing the intended system behavior.

6 Conclusion

In this paper, we present a formal and incremental approach for composing use cases expressed as extended Þnite automata. Our main contributions are : Þrst, our composition approach which may be distinguished by avoiding to keep use cases as solid blocs. It also allows to incrementally build the targeted use case using a ßexible method that promotes separation of concerns. Second, the introduction of variables in the description of use cases in the form of UCEA provides a concise, expressive, and scalable use case model. Our mechanism of composition has the advantage of generating the exact intended behavior without inducing any unexpected scenario. As a future work, we aim at extending the system speciÞcation by distinguishing between local variables to the use case and global variables to the speciÞcation. Such distinction may ease the process of writing composition expressions.

References

[1] *Message Seuqnce Chart (MSC). ITU Communcation Standardization Sector (ITU-T. Z120 Recommendation for MSC-2000*, 2000.

[2] R. Alur, K. Etessami, and M. Yannakakis. Inference of Message Sequence Charts. In *22nd International Conference on Software Engineering*, pages 304Ð313, 2000.

[3] D. Amyot, D. Cho, X. He, and Y. He. Generating scenarios from use case map speciÞcations. In *Third International Conference on Quality Software, Dallas*, November 2003.

[4] F. Bordeleau and J.-P. Corriveau. On the Need for ÓState Machine ImplementationÓ Design Patterns. In*Proceedings of ICSE 2002 Workshop on Scenarios and state machines: models, algorithms, and tools*, May 2002.

[5] M. Glinz. An integrated formal model of scenarios based on statecharts. In *In proceeding of the fifth european Software Engineering Conference*, pages 254Ð271. Springer Verlag, 1995.

[6] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231Ð274, June 1987.

[7] D. Harel and H. Kugler. Synthesizing state-based object systems from LSC speciÞcations. *Lecture Notes in Computer Science*, 2088:1Ð??, 2001.

[8] K. Koskimies, T. Mannisto, T. Systa, and J. Tuomi. SCED: A tool for dynamic modelling of object systems. Technical Report A-1996-4, 1996.

[9] P. Linz. *An introduction to formal languages and automata*. D. C. Heath and Company, Lexington, MA, USA, 1990.

[10] R. Mizouni, A. Salah, S. Kolahi, and R. Dssouli. Composition of use cases using synchronization and model checking. In E. Najm, J.-F. Pradat-Peyre, and V. Donzeau-Gouge, editors, *FORTE*, volume 4229 of *Lecture Notes in Computer Science*, pages 292Ð306. Springer, 2006.

[11] J. Ryser and M. Glinz. Dependency Charts as a Means to Model Inter-Scenario Dependencies. In *In G. Engels, A. Oberweis and A. Zndorf (eds.): Modellierung 2001. GI-Workshop, Bad Lippspringe, Germany. GI-Edition - Lecture Notes in Informatics*, volume P-1, pages 71Ð80, 2001.

[12] A. Salah, R. Dssouli, and G. Lapalme. Implicit integration of scenarios into a reduced timed automata. *Information and Software Technology*, 45, Issue 11:715Ð725, 2003.

[13] S. Somé, R. Dssouli, and J. Vaucher. From scenarios to timed automata: Building speciÞcations from users requirements, 1995.

[14] S. Uchitel, J. Kramer, and J. Magee. Synthesis of behavorial models from scenarios. In *IEEE Transactions on Software Engineering*, volume 29, February 2003.

[15] J. Whittle and J. Schumann. Statechart synthesis from scenarios: An air trafÞc control case study. In *International Conference on Software Engineering (ICSE2002)*.

Real-Time Trust Management in Agent Based Online Auction Systems[*]

Rinkesh Patel, Haiping Xu and Ankit Goel
Computer and Information Science Department
University of Massachusetts Dartmouth
North Dartmouth, MA 02747
{g_rpatel, hxu, agoel}@umassd.edu

Abstract. *Agent based online auctions have not yet become popular because they are not trustable. One of the major concerns in agent based online auctions is the shilling behavior problem, which makes winners have to pay more than what they should pay for auctioned items. In this paper, we propose a real-time trust management module for agent based online auction systems using role-based access control mechanisms. As one of the key components of the trust management module, a security agent can actively monitor online auctions in order to detect abnormal bidding behaviors in real-time. To illustrate the feasibility of our approach, we implemented a prototype real-time trust management module for agent-based online auction systems, and demonstrated how shill agents could be efficiently detected.*

1. Introduction

One of the most popular electronic commerce activities in recent years has been the use of online auction systems. Among the various auction types, the English auction has emerged as the preferred form for online auction systems (e.g., eBay) due to its characteristics of multiple bids and ascending bidding price [1, 2]. As the number of users and products increases, more time is required for a user to search and bid for an auctioned item. To cope with this problem, agent based online markets have come into play. An agent based online auction system is a multi-agent system [3] that comprises software agents to handle tedious tasks on behalf of human users. Each agent is autonomous and capable of taking actions to fulfill its goal. Thus, in an agent based online auction system, an agent can represent a user to search and bid for a product based on the constraints defined by the user.

However, with the rapid rise in the number of users, fraudulent behaviors in online auctions become more and more severe. The British Sunday Times recently revealed that shill biddings were very common on eBay [4]. A shill bidding is an act of bidding against other bidders in order to raise the auction price, so a winner has to pay more than what he should pay for an auctioned item [2]. In a trustworthy online auction system, buyers must trust sellers to provide the services they advertise, and not indulge in shill bidding; while sellers must trust buyers to be capable of paying for goods or services, and be authorized to make purchases on behalf of an organization. Trust in the sellers' competence and honesty will influence a buyer's decision on choosing sellers. In addition, users also must trust an auction house for not disclosing their personal information. Thus, there is a pressing need for a trust management system to maintain trust among users as well as with the online auction system.

In this paper, we propose a real-time trust management model to establish trust for agent based online auction systems. In our proposed model, a security agent is responsible for keeping track of each transaction and detecting unusual activities, such as shill biddings; while an authorization module can update a user's role and access permissions dynamically. Due to real-time actions against any abnormal auction activities, our trust management model can effectively maintain trust for agent based online auction systems.

The rest of this paper is organized as follows. Section 2 discusses about related work. Section 3 describes agent based online auction systems. Section 4 introduces a real-time trust management module integrated with a security agent. Section 5 presents an example to show how shill agents can be detected in real-time. Section 6 provides conclusions and our future work.

2. Related Work

There are two main strands of work to which our research is related, i.e., work on agent-based online auction system and work on trust management in e-commerce. Ito and his colleagues proposed *BiddingBot* as a multi-agent system that supports co-operative bidding [5]. In their approach, bidding decisions are actually made by users rather agents. Ogston and Vassiliadis proposed a peer-to-peer agent-based auction system for continuous double auctions [6]. They found that peer-to-peer auctions are able to display price convergence behavior similar to that of centralized auctions. In Collins and his colleagues' work, a multi-agent system for contract negotiation was

* This material is based upon work supported by the Chancellor's Research Fund and UMass Joseph P. Healey Endowment Grants.

presented [7]. The system can be used as a testbed for online auctions; however, it may have problems with secrecy of bids, non-repudiation, and manipulation of bids. Although the above efforts are useful in justifying the feasibility of agent-based approach for online auctions, there are no attempts so far to provide security mechanisms to prevent an agent-based online auction system from being abused. Therefore, it is still hard to convince users to adopt the existing agent-based approaches for practical usage.

On the other hand, most of the previous work related to trust management in e-commerce tried to secure online transactions, and establish trust among users by proposing different trust models [8, 9, 10]. Trust management using reputation models are based on a user's prior history and feedback from other users. For example, the reputation based trust model used by eBay has a very simple rating scheme for users. As one of the major drawbacks of this approach, it is possible for a user to provide counterfeit ratings for other users with a dummy account. Zacharia and Maes implemented a social mechanism of reputation management in Kasbah, in which a central system keeps track of users' explicit ratings, and uses these ratings to compute a person's overall reputation in a directed graph [10]. However, it is not clear how the agents may collect the ratings in an open agent-based environment.

Our work is closely related to a trust management model proposed by Herzberg and his colleagues [11], which was later extended by Mouri and his colleagues for consideration of changes in user's internal state [12]. In their proposed models, a trust management system consists of a trust establishment module and a role-based access control (RBAC) module [13]. However, their models are either "stateless" in nature, or use state information only when a user starts a new session. Thus, their approaches can not ensure trust among users in real-time.

In this paper, we propose a real-time trust management model for agent based online auction systems. Our proposed model can be used to establish and maintain trust among agents based on both agents' history information and real-time state information. To monitor and detect any undesired behaviors such as shilling behaviors in an agent based online auction system, a security agent is designed and implemented. In addition, we isolate various security related policies in different modules, so the policies can be updated dynamically.

3. Agent Based Online Auction System

An agent based online auction system is a multi-agent system that facilitates online auction activities on behalf of human users to make users' life much easier. We have developed a prototype agent based online auction system using the JADE agent development framework [14]. Figure 1 shows a client-server architecture of our agent based online auction system, which consists of various types of software agents, such as search agent, bidding agent, and auction agent. In particular, a security agent is introduced to provide security mechanisms for detection of undesired bidding behaviors.

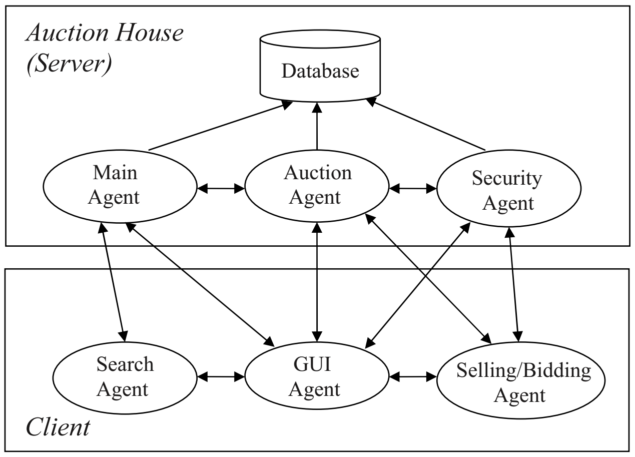


Figure 1. Architecture of agent based auction systems

The agent based online auction system is managed by an auction house administrator and used by various sellers and buyers. The auction house is implemented at the sever side with three major types of agents, namely the main agent, the auction agent, and the security agent. The main agent works as a controller for the auction house, and is responsible for creating new accounts for users, creating auction agents, and also responding to queries for items or auctions from agents at the client side. For each new auction, a corresponding auction agent is created to handle its auction related activities such as posting bids. While an auction is running, an agent representing a user can put bids on auctioned items; meanwhile, the corresponding auction agent is responsible for updating bidding activities for all involved agents. At the end of an auction, the auction agent notifies the winner of the auction, and passes the control back to the main agent. As a major component for security, the security agent monitors all online auction transactions performed by bidding agents.

The agents that work on behalf of human users are implemented at the client side, which involves three major types of agents, namely the search agent, the selling/ bidding agent, and the GUI agent. A GUI agent receives commands from a user, and updates the user interface when messages are sent and received. A search agent can automatically search and join an auction on behalf of a user. Finally, a selling/bidding agent is responsible for initiating auctions or automatically placing bids on behalf of a user according to user defined bidding strategies. Note that a user can be a seller and a bidder at the same time.

In the agent based online auction system, a user can configure a bidding agent by providing auction related information, such as the type of items they are interested in, maximum value for that item, and bidding strategies for how to put bids during an auction. A configured bidding agent will run autonomously, and make decisions on behalf of the user during the bidding process.

4. Trust Management for Online Auction Systems

4.1 Shilling Behaviors

A shill bidding is a deliberate activity of placing bids in order to artificially raise the price of an auctioned item. Although shilling behaviors are prohibited in most of the online auction houses, e.g., eBay, it is very easy for malicious users to disguise themselves and put shill bids.

As most of the auction houses allow users to create new accounts using false information, a seller can create a new dummy account and pretend to be a valid bidder to bid on his own auction for shilling purpose. A shill user may also get help from his friends, immediate employees, and relatives to put fake bids using their auction accounts. When normal buyers realize that they have to pay extra for an auctioned item due to shilling activities, the credibility of the online auction house will surely be affected. To maintain trust among users as well as with the auction house, it is necessary to provide security mechanisms to detect shilling behaviors in real-time, and restricts further abnormal activities done by shill bidders.

Shilling behaviors could be much more severe in an agent based online auction systems because detection of shill bidders can be more difficult than in ordinary online auction systems, where auction activities are continuously monitored by human users. Furthermore, shill bidders may take advantages of the agent technology to introduce more shilling activities that are hard to detect. The major goal of this paper is to propose a real-time trust management module that can detect shilling behaviors and takes appropriate actions accordingly in a timely manner for agent based online auction systems.

4.2 An Overview

Figure 2 is an overview of our proposed trust management module in an agent based online auction system. From the figure, we can see that a human user can configure an agent to initiate an auction as a seller or put bids on an auctioned item as a buyer. Before an agent starts to work, it must go through a trust management module for security purpose. The agent needs to send a

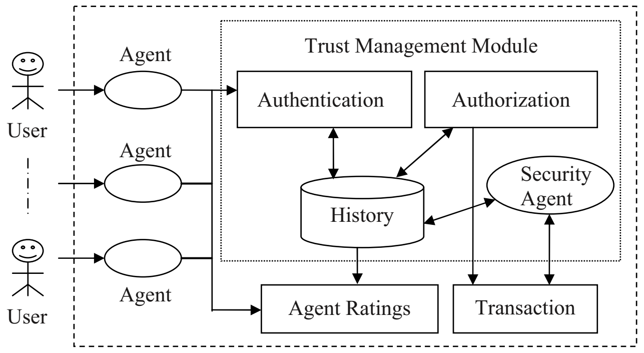


Figure 2. Trust management module

digital certificate or user credentials to the trust management module for authentication and authorization. Once the user configured agent is authenticated and authorized, it will be allowed to place requests for auction related activities. During the auction process, a configured agent can check current status or ratings of other configured agents in order to make proper decisions on choosing the right auction. Meanwhile, a security agent is designed for monitoring auction transactions for any suspicious bidding behaviors.

4.3 Trust Management Module

The trust management module (TMM) defined in Figure 2 is a key component in an agent based online auction system for trust maintenance, which can be further refined as shown in Figure 3. From the figure, we can see that the trust management module consists of a number of sub-modules such as authentication, authorization, state and history modules. As one of the major features of our TMM module, the security agent works closely with other modules of the TMM to maintain trust among agents in real-time. The authorization module, the access control module, and the security agent have their own policy rules defined by the auction administrator. Each set of policy rules are modularized in a corresponding database that can be updated dynamically without shutting down the agent-based online auction system.

Both the history module and the state module are parts of the TMM that are used to store and maintain the activities performed by user configured agents. When a user configured agent provides its digital certificate to the authentication module, the authentication module checks the certificate against previously stored information in the history module. If the authentication process is passed, the agent receives its initial pass, and is ready to make requests to perform auction activities. However, to make a request, the agent must also go through the authorization module, which consists of two major procedures, namely the role assignment and the access control. The role assignment process assigns a role to the configured agent dynamically by applying role assignment policies, called *RA Policies* based on gathered information related to the corresponding user. The access control process grants or restricts the access to auction related activities for the user configured agent based on access control policies, called *AC Policies*. The access control mechanism also determines how frequently the security agent should monitor a configured agent's auction transaction activities. After being authorized, the configured agent can start to make requests for auction related activities with certain permissions. Meanwhile, the security agent continuously monitors auction related activities in the auction system according to security agent policies called *SA Policies*. Once the security agent detects any shilling behaviors, the security agent determines the severe level of the shilling behaviors, and updates the current state information of the

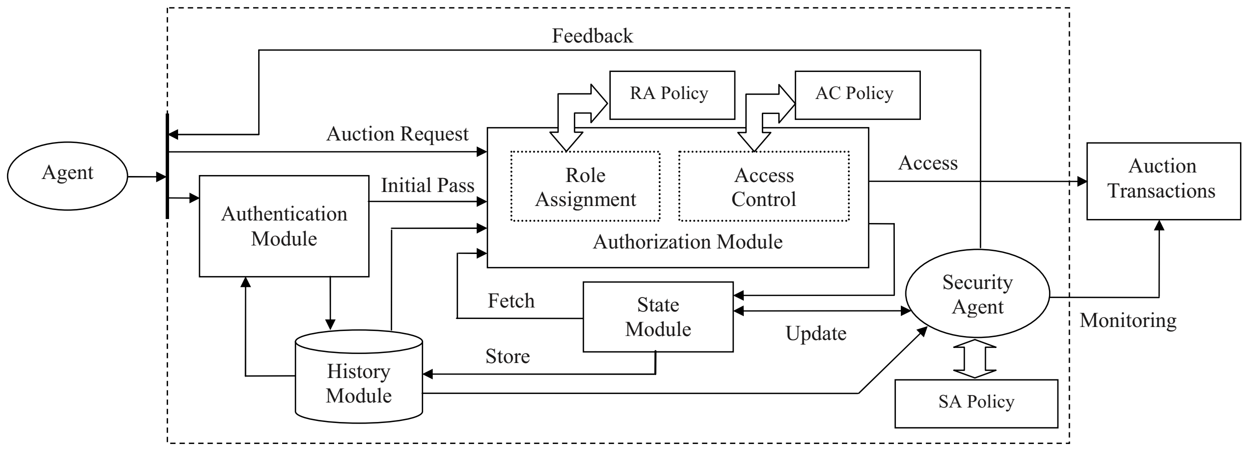


Figure 3. Refinement of the trust management module

shill bidder. Furthermore, the security agent notifies all participating configured agents about the shilling behavior of the shill bidder in the corresponding auction.

4.4 History Module and State Module

The history module stores information about users' previous auction activities over a certain period of time. Examples of such information include previously assigned roles, access information, shilling behaviors, and feedback information. After each successful transaction of a configured agent, the information in the history module is updated, and is ready to be accessed by the security agent and the trust management module for decision making.

The state module stores information related to the configured agents and their current activities, which includes currently assigned agent roles, granted resource access information, and possible shilling behaviors. The state module information is used along with the history module information to determine a configured agent's next dynamic role assignment by the role assignment module.

The information stored in the state module can be updated by both of the security agent and the authorization module. Current state information of the configured agent is the vital information used in a role assignment process for an agent's next bidding activity. After each successful transaction, information stored in the state module is saved into the history module for future use.

4.5 Authorization Module

In our proposed agent-based auction system, all requests made by an agent are controlled by the authorization module (Figure 3). In other words, in order to perform any auction related activities, an agent must first get an appropriate role and access permissions from the authorization module. We now describe in more details for the two major components in the authorization module, i.e., the role assignment module and the access control module, as follows.

Role Assignment Dynamic role assignment is performed according to predefined *RA Policies* stored in a role assignment database. The needed information for the computation includes the following: (1) the configured agent's history information, number of positive and negative feedbacks, and feedback status from the history module; (2) the user's current role and shilling behavior information from the state module. According to the *RA* policies, an agent can be assigned to one of the following five types of roles: *most trusted*, *trusted*, *average*, *untrusted*, and *most untrusted* for both sellers and buyers. As an example of role assignment rules, the following policy written in Prolog defines the conditions for assigning the *most trusted* buyer (*mtb*) role to an agent.

```
%If the current role is mtb or tb, and the
agent's reputation score is high enough.
conditions_for_mtb(HIST,CUR_ROL,SHILL_STATUS,POS_
FB,NEG_FB):- HIST>=0.8,
(is_identical_to(CUR_ROL,mtb);
is_identical_to(CUR_ROL,tb)),
(is_identical_to(SHILL_STATUS,clean);
is_identical_to(SHILL_STATUS,probable)),
POS_FB>=1000, NEG_FB=<(0.1*POS_FB).
```

According to the above rule, an *mtb* role is assigned to a bidding agent when the agent satisfies requirements such as having more than 1000 positive feedbacks, having less than 100 negative responses, not doing shilling in the last transaction, and taking a role of either *most trusted* buyer (*mtb*) or *trusted* buyer (*tb*) currently.

Access Control The access control module grants or denies an agent the access to resources requested by the agent. It may also restrict a bidding agent to perform certain auction activities for a period of time, if the agent has any shilling behaviors in its previous history.

A newly registered agent, which starts by getting a role of *average* buyer, is assumed to be trustable, so it shall have the privilege to perform auction activities. During the auction time, if an agent's role is downgraded (e.g., from a role of *average* buyer to a role of *untrusted* buyer), it signifies that undesired activities have been done by the agent. In this case, the access control module may give warnings to the agent or restrict the agent to perform further activities for a certain period of time. If an agent is restricted to participate in any auction related activities for a certain period of time, the access control module sets the penalty status as *active* for the agent, and will deny all requests by that agent. The following is an example of *AC Policy* in Prolog that defines how different penalties can be applied and how different security levels will be set according to different situations of role changes.

```
% When a user's role has been downgraded
(is_identical_to(CUR_ROLE,tb),is_identical_to(LAS
T_ROLE,mtb))->
(penalty_assess(PENALTY,FIRST_TIME,oneday),assign
ed_value(SEC_STATUS,level3));
(is_identical_to(CUR_ROLE,avgb),is_identical_to(L
AST_ROLE,tb))->
(penalty_assess(PENALTY,FIRST_TIME,oneweek),assig
ned_value(SEC_STATUS,level2));
(is_identical_to(CUR_ROLE,ub),is_identical_to(LAS
T_ROLE,avgb))->
(penalty_assess(PENALTY,FIRST_TIME,twoweeks),assi
gned_value(SEC_STATUS,level2));
(is_identical_to(CUR_ROLE,mub),is_identical_to(LA
ST_ROLE,ub))->
(penalty_assess(PENALTY,FIRST_TIME,onemonth),assi
gned_value(SEC_STATUS,level1));
```

Note that the security level assigned (from 1 to 4, with level 1 being the highest security level) will be used by the security agent to determine the way the bidding agent should be monitored for abnormal behaviors when the bidding agent's auction activities are resumed.

A configured agent, whose role has been downgraded due to its past undesired behaviors, may gain back trust by refraining itself from performing undesirable activities after the restricted time period expires. When an agent has shown sufficient evidence for trustworthiness, the role assignment module may upgrade the agent's role according to predefined *RA Polices*. In addition, to prevent further undesired bidding behaviors, for those agents with high security level, the security agent will monitor them more closely and thoroughly for any activities performed by them when their bidding activities are resumed.

4.6 Security Agent and Detection Rules

To make online auction system trustworthy and to ensure the bidding process reliable, we should prevent and minimize undesired bidding behaviors. The security agent is designed for the purpose of monitoring bidding agents for their activities, and detecting shilling behaviors based on shill patterns and security policies. Since it is not feasible to monitor every activity of each agent in details, we decrease the load of the security agent by defining different security levels such that the depth of checking is directly proportional to the level of distrust in the user. For example, a bidding agent with security level of 1 will receive the most careful monitoring.

The following *SA Policy* is an example of detection rules that defines the way of monitoring a bidding agent with security level 2.

```
% Invoked if the security status is level 2
security_level_2(SHILL_STATUS,SHILL_PROB,DIFF_IN_
LOC,CONC_BID,WL_RATIO,PRESENT_INITIAL_STYLE):-
proximity_of_ip(TEMP1,DIFF_IN_LOC),
concurrent_bid_check(TEMP2,CONC_BID,WL_RATIO),
initial_bid_style(TEMP3,PRESENT_INITIAL_STYLE),
SHILL_PROB is TEMP1+TEMP2+TEMP3,
status_evaluation(SHILL_STATUS,SHILL_PROB).
```

To detect shilling, the security agent is configured to perform different types of security checks. At the lowest level (level 4), only the distance in locations of a buyer and a seller are checked according to their IP addresses. At level 3, we check if a buyer is participating in concurrent auctions with identical auctioned items. Note that concurrent shilling, where a bidding agent places bids on an auction item with higher auction price rather than on the auctioned item with lower auction price, is a strong indication of shilling behaviors. At level 2, the security agent analyses the bidding style of a buyer against common shill patterns. In many cases, it has been found that a shilling agent does aggressive biddings at the beginning, and stops bidding towards the end of the auction to avoid winning the auction. Finally, at the highest security level (level 1), the security agent performs all above checks coupled with an analysis of the bidding agent's history. The security agent derives a shill factor by applying different security rules on the agent's current and previous behaviors. If the shill factor is high enough, the agent's bidding status will be set as *shilling*, and the state module will be updated. The updated information stored in the state module will be used by the role assignment module when the shill bidder makes a new bidding request. Furthermore, as an alert, the security agent will inform all participating agents about the detected abnormal behavior. In a severe situation, when an agent's shilling behaviors are committed based on strong evidence, the security agent will force the auction to be closed and notify all involved users about such decision.

5. An Example

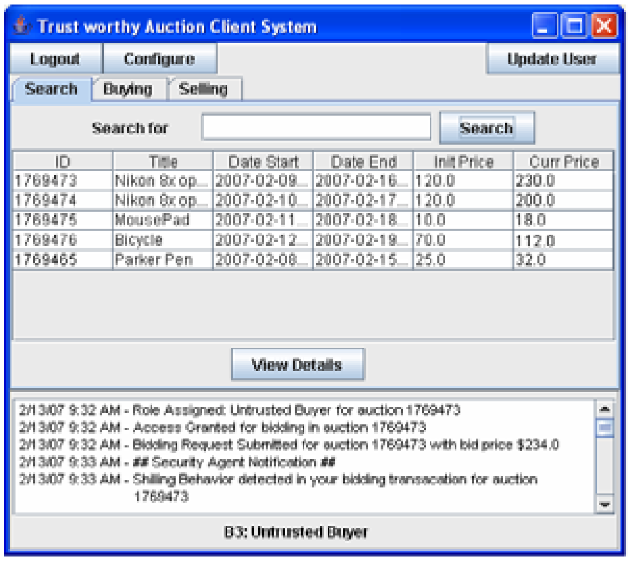
Our approach can be illustrated by an example of online auctions that involve shilling behaviors. In our example, we consider two concurrent online auctions – we call them *Auction 1* and *Auction 2*, which are initiated by seller *S*1 and *S*2, respectively. The auctioned items are "Nikon 8x Optical and 4x Digital Zoom Camera," which are identical in both of the two auctions. There are five bidding agents *B*1 to *B*5 that may put bids on either of the auctions. Agent *B*1, *B*2 and *B*4 are configured to work as normal bidders. But for agent *B*3, we set up a bad feedback history for the agent initially. Consequently, the role assignment module downgrades *B*3's role from *average* buyer to *untrusted* buyer when *B*3 makes a bidding request, and the access control module sets agent *B*3's security status to level 1. Furthermore, we configure agent *B*3 with the following bidding strategy: it tries to drive up the auction price of the item listed by seller *S*2 aggressively at the beginning, but stops to put bids on that item when the auction price reaches a certain value. In addition, agent *B*5 is configured with a strategy called *Preferred Seller Strategy*, which instructs agent *B*5 to put bids on the item listed by seller *S*1 rather than *S*2 for most of the time. Table 1 lists the role assignment and some of the access rights for each bidding agent.

Table 1. State information of bidding agents

Bidding Agent	Previous Role	Role Assignment	Access Control
B1	most trusted buyer	most trusted buyer	no actions Sec_status: level 4
B2	trusted buyer	trusted buyer	no actions sec_status: level 3
B3	average buyer	untrusted buyer	warning sec_status: level 1
B4	average buyer	average buyer	no actions sec_status: level 2
B5	average buyer	average buyer	no actions sec_status: level 2

While both *Auction 1* and *Auction 2* are running, the security agent monitors each bidding agent according to its security level. Since the bidding agent *B*1, *B*2 and *B*4 show their normal bidding behaviors, the security agent sets their bidding status as *normal*. On the other hand, since agent *B*5 puts bids on both of the auctions, and the security agent detects that *B*5 sometimes bids on one of auctions with higher auction price. By further analyzing *B*5's bidding behaviors, the security agent has found that *B*5 bids on the item listed by seller *S*1 for most of the time, and puts bids on the item after the reserve price has reached. This indicates that agent *B*5 does not attempt to drive up the price because it has no intention to avoid winning the auction. Thus the security agent concludes that agent *B*5's bidding status is *normal*.

Since *B*3's security status has been set to level 1, the security agent analyzes *B*3's bidding activities thoroughly and finds that *B*3's bidding behavior matches the concurrent shilling pattern, where an agent places bids on the auctioned item with higher auction price rather than on the auctioned item with lower price, and also tries to avoid winning an auction by stopping bidding when the price reaches the reserve price. Furthermore, the security agent analyzes *B*3's current and past bidding transactions as well as the number of wins in auctions listed by both seller *S*1 and *S*2. The security agent notices that *B*3's win-loss ratios on auctions listed by seller *S*1 and *S*2 are close to 0. Based on the above knowledge, the security agent assigns *B*3's bidding status as *shilling*. The security agent then notifies all participating agents, and updates the state module information for agent *B*3. Figure 4 shows the user interface for agent *B*3 with a notification from the security agent.



ID	Title	Date Start	Date End	Init Price	Curr Price
1769473	Nikon 8x op...	2007-02-09...	2007-02-16...	120.0	230.0
1769474	Nikon 8x op...	2007-02-10...	2007-02-17...	120.0	200.0
1769475	MousePad	2007-02-11...	2007-02-18...	10.0	18.0
1769476	Bicycle	2007-02-12...	2007-02-19...	70.0	112.0
1769465	Parker Pen	2007-02-08...	2007-02-15...	25.0	32.0

```
2/13/07 9:32 AM - Role Assigned: Untrusted Buyer for auction 1769473
2/13/07 9:32 AM - Access Granted for bidding in auction 1769473
2/13/07 9:32 AM - Bidding Request Submitted for auction 1769473 with bid price $234.0
2/13/07 9:33 AM - ## Security Agent Notification ##
2/13/07 9:33 AM - Shilling Behavior detected in your bidding transacation for auction
                  1769473
```

B3: Untrusted Buyer

Figure 4 User interface for bidding agent *B*3

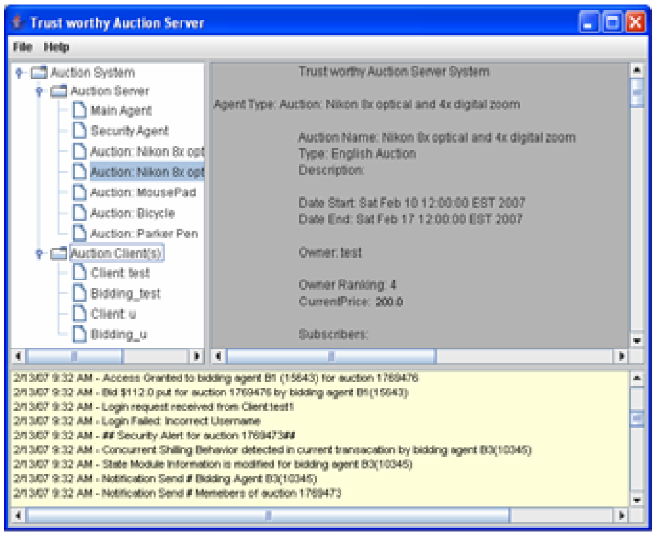
When agent *B*3 places a new bidding request, the role assignment module assigns *B*3 a role of *most untrusted* buyer, and as a penalty, the access control module restricts *B*3 from putting bids for a week. Table 2 shows the updated state information of each agent after the analysis is done by the security agent.

Table 2. Updated state information of bidding agents

Bidding Agent	Bidding Status	Role Assignment	Access Control
B1	normal	most trusted buyer	no actions sec_status: level 4
B2	normal	trusted buyer	no actions sec_status: level 3
B3	shilling	most untrusted buyer	one week penalty sec_status: level 1
B4	normal	average buyer	no actions sec_status: level 2
B5	normal	average buyer	no actions sec_status: level 2

In Figure 5, we show a user interface for an auction house administrator to view all auction related activities performed by bidding agents, as well as any actions taken by the security agent.

In our prototype agent-based online auction system, actions against a shill bidder are taken in real-time by updating the agent's role assignment and restricting the agent's access to auction related activities. To ensure a more accurate detection of shill bidders, the security agent also requires evidence such as user's IP address, ratings, user feedbacks and current and past trading histories. In addition, expert experience and considerations of practical situations are vital for us to set up effective policy rules for shill detection. With more and more expert knowledge on shill patterns [15, 16], our approach can be very effective in shill detection for practical agent-based online auction systems.



```
Trust worthy Auction Server System

Agent Type: Auction: Nikon 8x optical and 4x digital zoom

          Auction Name: Nikon 8x optical and 4x digital zoom
          Type: English Auction
          Description:

          Date Start: Sat Feb 10 12:00:00 EST 2007
          Date End: Sat Feb 17 12:00:00 EST 2007

          Owner: test

          Owner Ranking: 4
          CurrentPrice: 200.0

          Subscribers:
```

```
2/13/07 9:32 AM - Access Granted to bidding agent B1 (15643) for auction 1769476
2/13/07 9:32 AM - Bid $112.0 put for auction 1769476 by bidding agent B1(15643)
2/13/07 9:32 AM - Login request received from Client:test1
2/13/07 9:32 AM - Login Failed: Incorrect Username
2/13/07 9:32 AM - ## Security Alert for auction 1769473##
2/13/07 9:32 AM - Concurrent Shilling Behavior detected in current transacation by bidding agent B3(10345)
2/13/07 9:32 AM - State Module Information is modified for bidding agent B3(10345)
2/13/07 9:32 AM - Notification Send # Bidding Agent B3(10345)
2/13/07 9:32 AM - Notification Send # Memebers of auction 1769473
```

Figure 5 User interface for auction house administrator

6. Conclusions and Future Work

In order to build a trustworthy agent-based online auction system, we introduced a real-time trust management module (TMM) to restrict and prevent undesired bidding behaviors such as shilling behaviors in online auctions. Based on an agent's current and previous behaviors in agent-based online auctions, the real-time trust management module can assign agent roles dynamically, and grant or deny an agent for varying levels of access to auction related resources and activities. Meanwhile, any undesirable bidding behaviors performed by a bidding agent can be automatically detected by a security agent. We have defined different policy rules in Prolog for dynamic role assignment, access control mechanisms, and undesirable bidding behavior detection. The shill detection example, which is simulated on our prototype agent based online auction system, shows that our approach is feasible and efficient. For our future work, we will try to formalize various policy rules, and based on existing work on shill patterns [2, 16], we will try to develop a more accurate method for real-time shill detection in agent-based online auction systems.

References

[1] A. Vakali, L. Angelis, D. Pournara, "Internet Based Auctions: A Survey on Models and Applications," *ACM SIG on E-commerce Exchanges*, Vol. 2, No. 2, Jun 2001, pp. 5-13.

[2] H. Xu and Y. Cheng, "Model Checking Bidding Behaviors in Internet Concurrent Auctions," To appear in *International Journal of Computer Systems Science & Engineering (IJCSSE)*, 2007.

[3] Katia Sycara, "MultiAgent Systems," *AI Magazine*, Vol. 19, No. 2, Summer 1998, pp. 79-92.

[4] TimesOnline, "Revealed: How eBay Sellers Fix Auctions," *The Sunday Times*, Tech & Web, Jan 28, 2007. Retrieved on January 29, 2007, from http://technology.timesonline.co.uk/tol/news/

[5] T. Ito, N. Fukuta, T. Shintani, K. Sycara, "BiddingBot: A Multiagent Support System for Cooperative Bidding in Multiple Auctions," In *Proceedings of the Fourth International Conference on MultiAgent Systems*, July, 2000, pp. 399-400.

[6] E. Ogston and S. Vassiliadis, "A Peer-to-Peer Agent Auction," In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2002)*, pp. 151-159.

[7] J. Collins, W. Ketter, M. Gini, "A Multi-Agent Negotiation Testbed for Contracting Tasks with Temporal and Precedence Constraints," *International Journal of Electronic Commerce*, 7(1):35-57, 2002.

[8] Bhavani Thuraisinham, "Trust Management in a Distributed Environment," In *Proceedings of the 29th COMPSAX'05*, 2005.

[9] T. Gradison, M. Sloman, "A Survey of Trust in Internet Applications," *IEEE Communications Surveys*, Fourth quarter 2000.

[10] G. Zacharia and P. Maes, "Trust Management through Reputation Mechanisms," *Applied Artificial Intelligence*, 14:881-908, 2000.

[11] A. Herzberg, Y. Mass, J. Mihaeli, D. Naor, and Y. Ravid, "Access Control Meets Public Key Infrastructure," *IEEE Symposium on Security and Privacy*, Oakland, California, USA, 2000, pp. 2-14.

[12] H. Mouri, Y. Takata and H. Seki, "A Formal Model for Stateful Trust Management Systems," In *Proceedings of IASTED International Conference on Software Engineering and Applications (SEA 2005)*, Phoenix, USA, Nov. 2005, pp. 87-92.

[13] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-Based Access Control Models," *IEEE Computer*, 29(2):38-47, 1996.

[14] P. Moraitis and N. Spanoudakis. "Combining Gaia and JADE for Multi-Agent Systems Development," In *Proceedings of the 17th European Meeting on Cybernetics and Systems Research (EMCSR 2004)*, Vienna, Austria, April 2004.

[15] D. H. Chau, S. Pandit and C. Faloutsos, "Detecting Fraudulent Personalities in Networks of Online Auctioneers," *PKDD 2006*, Berlin Germany.

[16] J. Trevathan and W. Read, "Undesirable and Fraudulent Behaviour in Online Auctions," In *Proceedings of the International Conference on Security and Cryptography*, 2006, pp. 450-458.

GEOMETRIC THUMBNAILS FOR WEB SEARCHING

Chris Dunn, Beomjin Kim
Department of Computer Science
Indiana University-Purdue University
Fort Wayne, IN, U.S.A.

ABSTRACT: *Previous studies have shown the effectiveness of using thumbnails for scanning large amounts of information such as Internet searching. This paper introduces a technique that converts web pages to geometrically produced thumbnails. It shows the layout of web pages and search term distributions within the web pages through graphical illustrations. The structural preview of the documents will help the users to understand the components within the web pages including the number of non-text components and the image to text ratio. The color-coded geometrical objects represent the amount of text content, search term concentration, and their relationships with the text portions. By providing the layout and position of the search terms together, the user will be able to avoid irrelevant pages, and will have direct access to the portion of the web page that pertains to the users interests. Unlike other studies that use pixmap-based thumbnails that generate significant network usage, the proposed method represents the web page through simple geometrically produced thumbnails that will cut down on the amount of network traffic. We expect the developed visual abstract of web pages will improve the users ability to search by alleviating the need to sort through irrelevant information.*

1. INTRODUCTION

Internet provides us with a huge amount data. Further the amount of data available continues to grow, and the forms it takes constantly expand. Recently, the difficulty is not searching for information, but finding relevant information from the search. The search engine commonly returns a long list of relevant Web resources with limited descriptions, while the users frequently refine their queries only after reviewing a very small portion of the retrieved results [6]. With so many pages to evaluate, an effective strategy that does not require the searcher to evaluate every page or even many of them is of paramount importance.

The traditional summary based approach, which poorly utilizes the screen space, can only shows a few pages' summaries on the screen simultaneously. The words deliver little information about the page in question for the amount of space they take up, and large amounts of white space often appear on summary based search results. And the summaries reveal information only about the portion of the webpage where the text is extracted from, and if the summary does not use verbatim contiguous text, it can be misleading in terms the amount of distance between the search terms and the information density of the page. In addition, it does not provide us search terms distribution or their correlations within the document.

Visualization of the information is an effective technique to assist the user in exploring a vast space. Various visualization techniques and graphical user interfaces assisted users in exploring Internet [7]. In multiple studies, subjects that participated in empirical studies have shown satisfaction in using the visual abstraction of information for searching [9].

WebTOC is a visualization method for presenting the contents of a website using a hierarchical table of contents [8]. WebTOC uses several visual attributes such as lines, color, length, and shadow to present a link to a Web resource (document, image, multimedia), file type, file size, and number of items subordinate to the document. This visual abstraction assists the user by giving an overview of an entire website, the hierarchical relationship of Web resources, and in finding the location of the website where it has numerous resources.

Thumbnail is a small version of image that helps the user to visualize documents or Web pages. This provides the preview of the underlying information that helps the users' in scanning the information quickly and assists their search activities [5]. Beyond the easiness in organizing the thumbnail, it is also an effective memory aid, which helps users to recall previous reviewed information [7].

Woodruff et al. (2001) introduced a technique for creating thumbnails that utilized the advantages of text summaries and plain thumbnails [4]. In their experimental study, the user's search time for desired information had decreased when they used textually enhanced thumbnails over summary-based and plain thumbnails. Although the system has supported users' search activities, the usage of thumbnails in studies was mainly confined to the organization of visited Web pages. This application could not deliver the distributional pattern of search terms, which is important in understanding the relevancy of Web pages. In addition, these thumbnails require sending a large amount of content to the end user, in the form of a pixmap. The thumbnail view of a Web page can be further abstracted, while providing more information.

The visualization we are proposing in this paper attempts to synthesize the advantage of graphical abstraction of the page with the information stored in the structural layout and term positions. The proposed technique presents the layout of web pages and search term distribution in the web sources graphically. This illustration will utilize more screen space. The abstracted structural layout of the Web page will allow users to understand the underlying contents of the webpage. The color-coding technique transforms the frequency and distribution of search term into an intuitive visual abstraction that helps the user to find the location in the webpage that has the sought information. This method does not rely on a pixmap of the page or on sending the content of the page to the end user. Instead, once analyzed by the search provider, only geometric and color information needs to be sent, reducing the amount of associated traffic.

2. METHODOLOGY

In the proposed visualization, each web page is illustrated as a rectangular shaped thumbnail whose dimension represents the length of the web document. The visualization procedure that generates the thumbnail of each web page utilizes color-coding and layout information can be divided into two major procedures. The first step is analyzing the web documents and producing the layout information. The second step is transforming the stored layout information to a color-coded visualization based on the terms contained in the search query being answered.

The parsing will convert a complex web page into a simple form, an ordered list of tokens. A token is any html element or block of text. An html element is enclosed in enclosed in "<…>" [1]. The text is found interwoven between the html elements. A text token can consist of any amount of text, and can be divided wherever is desired as long as it is divided at white space. A text token may not contain an HTML element and must end before or at the beginning of the next html element. The token list will be in the order they are encountered in the document, and this order cannot be changed. These tokens will be mapped to the polygons that appear in the thumbnail of web document. The magnitude and position of each polygon reflects the actual size and location of the corresponding web component in the web page. In addition, each polygon will retain information on the type of token or tokens that generated it. It maintains this type information to allow it to be appropriately color-coded in the visualization phase.

The text of the web page is divided into multiple segments based on the paragraph tags within the HTML. Each text segment can have multiple text blocks such as headings and actual contents. To show the level of concentration of search terms in a text segment, all of the text in a segment (T_i) is treated as group and the frequency of search terms in T_i determines the color-code of the segment. So, all the text polygons representing text blocks in a segment will have the same color-code.

The types of web resources and document formats are developing so rapidly such as various forms of style sheets and stylesheet language [1]. Although it is difficult to predict the evolution of web technologies, we decided to use a recursive descent parser in the prototype. This has the advantage of speed and simplicity, since each token can be processed its own method and there was no need to worry about upcoming tokens. Standard generalized markup language (SGML) and extensible markup language (XML) data type definitions (DTD) are context-free grammars [2]. HTML uses context-free grammars that can be parsed in a linear time [1, 3].

The parsing is accomplished using a two-pass recursive descent parser. A recursive descent parse was chosen because it trivially solves the issue of embedded tags. The first pass corresponds to the first step, the analysis of the layout information. This step works by placing each component on a separate line whenever possible. Each tag has an associated code block, and these are independent of each other, with the exception that each modifies a state variable, which controls the available drawing area. The second step actually produces the polygons that are used in the visualization. This step uses the same process, except that all the components are placed horizontally across until the current drawable area can no longer contain another component horizontally, at which point the polygons are then placed on the next line. However, some tags will affect the drawable area in other ways. This is done by modifying the state variable. For example, a <br> will immediately drop the drawing to the next line, and <table> tags will subdivide the drawable are into multiple components.

The parser will yield an ordered list of tokens, say D. The list of tokens will be partitioned into segments. These segments should represent blocks of the webpage that end with white space. To accomplish this, two constraints need to be met: first, $D = \bigcup_{i=1..n} D_i$ where D_i is sublist of tokens of D. This will ensure that all the tokens appear in a segment, and that there is no duplication. Second, for all segments, the last token of D_i forces the browser to produce white space. Such elements include but are not limited to <p>, <table>, and <br>. Any method following these two constraints will produce an acceptable partitioning of D. A simple way to do this is to iterate through D until encountering tokens that force the production of white space, and using such tokens to produce the partitions. These segments will be mapped to a list of polygons.

The polygon list can be generated by using essentially the same method that is used to render web pages by web browsers. The rendering of polygons representing text tokens does not worsen the time complexity. It renders the page using the methodology used by a web browser, however, instead of rendering text, the prototype replaces it with a polygon. This can be done by replacing each text token in the text with a rectangle with a width and height the same as the highest and widest points of the text token rendered in the current font. Multimedia, images, and other components are also mapped to polygons based on their actual magnitude within the web page. For each defined polygon, we assigned a type based on the token that caused its production. Once this is done, rendering the whole list of polygons, P, will produce a close approximation of the page, sans the details inside the polygons (Figure 1). The polygons are partitioned into sublists, P_i, that meet the criteria $P=\bigcup_{i=1.n} P_i$ where P_i is a sublist of polygons in P. Since each polygon is produced by at least on token, and never produced by tokens from two separate D_is, the P_i can be associated with the D_i by index. It is possible that some of the P_i maybe empty. For example, a document may contain "…<p><font size="+3"></p><p>…". The segment starting at the first <p> will contain a font size token, which is required to properly render the rest of the page, but this segment will not have any polygon associated with it.

The text of each D_i is mapped to text segments, T_i, by concatenating all the text from text tokens. Let $d_{ik} \in$ D$_i$ and a text token. Therefore, $T_i = \sum_{k=1..n} d_{ik}$ where + is text concatenation. Note that some T_is may be empty, a D_i might only contain images, for example. P_i may also be empty, but it will be empty only if T_i is empty. If T_i is not empty, then at least one polygon will be in P_i to represent the text.

After mapping a text segment, T_i, to polygons, P_i, we applied a color-coding algorithm to compute a color for the text polygons in P_i. The given queries are divided into three subsets of search terms (S). A primary color (RGB color model) is assigned to a subset of search terms that informed the frequencies and relationships of the search terms in T_i.

For each T_i, the frequencies of search terms, $(F^S)_i$, is computed respectively where the frequencies of the search terms in the same S are added together. This procedure can be formulated as follows: $(F^S)_i = (\sum_{j=1}^{n} t_j^S)_i$ where t_j^S is the frequency of a search term in a subset, $(t_j^S)_i$ is the frequency of a search term in the ith segment T_i, and S is one of search terms subset among S -> *{R, G, B}*.

The intensity factor is defined by computing the frequency ratio of query subset to the magnitude of T_i. The ratio $(R^S)_i = (F^S)_i / Mag(d_i)$ where $Mag(d_i)$ is the total number of words in T_i, is mapped to an intensity scale among N different levels to determine the intensity of the subset's color. The higher the ratio, $(R^S)_i$, maps to the brighter intensity of the corresponding color. This color encoding function is applied to each search term subset for T_i separately that defines three intensities of the primary colors, I^{Red}_i, I^{Green}_i, and I^{Blue}_i. By blending I^{Red}_i, I^{Green}_i, and I^{Blue}_i together, a color code, C_i, is formed to represent the search terms distribution in T_i.

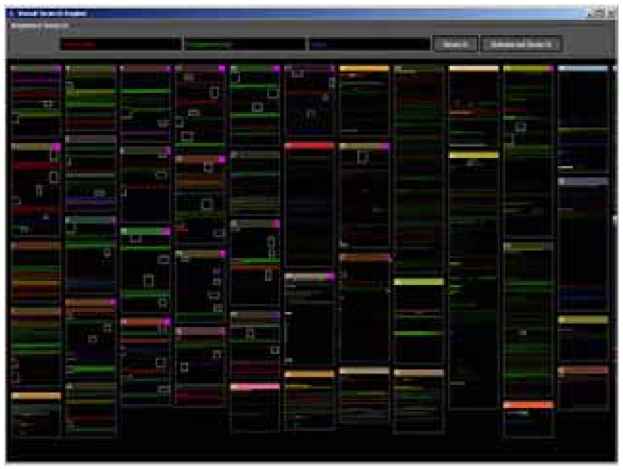


Fig.1 A view of visualized Internet search results.

Through the frequency-to-intensity mapping, segments with a higher concentration of the search terms' occurrences will be mapped to brighter colors. Finally, the color code C$_i$ is painted to the polygons in P_i that are associated with text tokens D_i. Figure 1 shows a view of the visual illustration of about 40 web pages.

3. DISCUSSION

This paper introduced a visualization method to present web search result through graphical illustrations. It will allow the user to utilize their perceptual cognition in exploring Internet search. While executing our pioneer study, we found that there can be three pertinent issues to address: precision, usability testing, and combining the visualization with a search engine.

Precision in the layout increases the detail and amount of layout information in the thumbnail, but comes with two costs. First, the higher precision, the more likely there is to be useless information in the result. Consider, for example, the visualization of a paragraph of text. A very precise version will be very jagged along the edge as the polygon moves in and out due to differences in the position of word wrapping on each line. A simpler visualization would give a more intuitive understand of the overall structure of the resource in this case. Secondly, this leads to excess network traffic. Every vertex in the polygon needs to be transmitted, therefore, smoothing the edge of a paragraph instead of displaying jaggedly will reduce the number of vertices that need to be sent across the network.

Web pages may contain many decorative images such as bullets, icons, and separators. Such decorations do not affect the information content of the page, they instead serve as visual formatting of the information. Filtering these images will reduce the amount of network traffic and hence increase performance. These decorations can be found by looking for images that have dimensions that are too small to contain information, or that have an unbalanced aspect ratio, so that one of their dimensions is too small to contain useful information. This separation can be conveyed just as usefully to the user by leaving white space in the search instead of adding an image, and removing the image reduces the amount of data to be transmitted and the amount of clutter in the final thumbnail.

Currently, the visualization technique lacks objective usability testing. In order to perform a non-biased evaluation of its usefulness, it will require comparison testing between the search effectiveness of the proposed method and the traditional methods of searching. It would also be useful to perform similar comparisons between this visualization technique and other thumbnail visualizations and variations on the precision of this visualization technique to determine what factors are most useful in a thumbnail, and where the limited screen space and network bandwidth resources can be most effectively allocated. Currently, we are in the process of preparing a series of experimental studies with the TREC data sets provided by National Institute of Standards and Technology (NIST) [10].

Finally, further studies are expected to apply our method to a search engine for realizing a visual search engine. The primary issues are storing the layout information and associating the information with the search terms. The layout information is comprised of a list of polygons. These can be stored in a database in much the same way the text of pages is, and indexed appropriately. Each collection of polygons must be associated with a set of words, so that they can be color-coded. In our prototype, we used two indices. One stored the actual words of the document, and their frequency in each segment. The second stored the polygons for each segment, and the total number of words in each segment. In the future, the search terms indexing associated with visualization components should be investigated in depth, while considering performance and optimization, to make the proposed visualization can be applicable to practical Internet search applications.

4. CONCLUSION

This paper presents a visualization technique that improves upon previous thumbnail visualizations for Internet searching. We have increased the amount of structural and substantial information projected to the user by transforming the web contents to a graphical representation while applying color-coding. Simultaneous, we have reduced the amount of network traffic generated by the thumbnails by using geometrically produced thumbnails instead of pixmaps. The proposed algorithm increases the users search speed and allows them to avoid portions of the page that don't include information that they are searching for. This will eventually improve the searcher's efficiency in performing web searches, which is an important improvement considering the size and growth of the Internet.

Acknowledgements

The authors would like to give thanks Philip Johnson who has provided a great contribution in developing the prototype system.

References

[1] HTML specification, World Wide Web Consortium (W3C), http://www.w3.org/TR/html4/

[2] Kilpeläinen, P. & Wood, D.: "SGML and XML document grammars and exceptions," *Information and Computation*, Vol. 169, No. 2 (2001) 230-251.

[3] Nederhof, M. & Bertsch, E.: "Linear-time suffix parsing for deterministic languages," *Journal of the ACM*, Vol. 43, No. 3 (1996) 524-554.

[4] Woodruff, A., Faulring, A., et al.: "Using thumbnails to search the Web," *Proceedings of the SIGCHI conference on Human factors in computing systems* (2001) 198-205.

[5] Suh, B., Ling, H., et al.: "Automatic Thumbnail Cropping and its Effectiveness," *Proceedings of the 16th annual ACM symposium on User interface software and technology* (2003) 95-104.

[6] Jansen, B. J., Spink, A., et al.: "Real life, real users, and real needs: A study and analysis of user queries on the Web," *Information Processing and Management*, Vol. 36, No. 2 (2000) 207-227.

[7] Hightower, R.R., Ring, L.T., et al.: "Graphical multiscale Web histories: a study of padprints," *Proceedings of the ninth ACM conference on Hypertext and hypermedia* (1998) 58-65.

[8] Nation, A.: "WebTOC: a tool to visualize and quantify Web sites using a hierarchical table of contents browser," *CHI 98 conference summary on Human factors in computing systems* (1998) 185-186.

[9] Card, S.K., Mackinlay, J.D., Shneiderman, B.: *Readings in information visualization using vision to think*, Morgan Kaufmann (1999).

[10] National Institute of Standards and Technology (NIST), The Text REtrieval Conference (TREC), http://trec.nist.gov

AD$_{kwik}$: Web 2.0 Collaboration System for Architectural Decision Engineering

Nelly Schuster, Olaf Zimmermann, Cesare Pautasso
IBM Zurich Research Laboratory, Säumerstrasse 4, 8803 Rüschlikon, Switzerland
{nes, olz, cpa}@zurich.ibm.com

Abstract

Capturing and sharing software architecture design rationale has always been particularly challenging in complex application domains such as enterprise computing. Facing the ongoing acceleration of technology innovation and economic forces such as outsourcing and offshoring, conservative knowledge management practices and existing tools are no longer sufficient, offering only rudimentary support for knowledge exchange and collaboration on the Internet. In this paper, we present AD$_{kwik}$, a Web 2.0 collaboration system supporting the cooperative decision making work of software architects. We describe the realization of AD$_{kwik}$ as a situational application wiki and discuss initial evaluation results. Thanks to its ease of use, AD$_{kwik}$ has already shown concrete benefits to its users, including rapid team orientation, tangible decision making advice, and simplification of asset harvesting.

1. Introduction

Architectural decisions capture the rationale behind software architecture design. In current practice, this knowledge is tacit and rarely captured explicitly [1]. However, an explicit knowledge engineering approach to architectural decision capturing is beneficial, e.g., to attain *regulatory compliance*. Governance and maturity models such as Capability Maturity Model Integration (CMMI) desire architectural decisions to be captured and archived along with justifications.

A second motivation for explicit decision capturing is *team collaboration*. On large software development projects (e.g., in enterprise computing), architectural decision making is a team effort. Typically a lead architect has the overall technical responsibility, but delegates certain decisions to subsystem architects, chief developers, and platform specialists. Communication problems between these roles occur frequently; it is challenging to reach a shared view and a consensus over the architecture. This is even more difficult to achieve due to the current offshoring and outsourcing trends; more and more development teams are geographically distributed.

A third motivator for architectural decision capturing is *reuse*. A vast amount of architectural knowledge exists in practitioner networks such as company-wide *Community of Practice (CoP)* networks [2]. Often architectural knowledge is tacit or embedded in code. If documented at all, it resides in inappropriate, therefore rarely visited data stores such as personal mail archives and poorly structured team repositories.

In response to the regulatory compliance requirements, the need for collaboration during architectural decision making and the opportunities for reuse of architecture design rationale in CoPs, we have started to apply architectural decision trees as a fine-grained unit of knowledge exchange within and between project teams [3]. In this paper, we present how such a knowledge exchange can be facilitated by AD$_{kwik}$[1], a Web 2.0 collaboration system supporting the cooperative work of software architects. AD$_{kwik}$ embeds a rich domain model into a situational application wiki, with the goal of making it easy for practitioners to share their knowledge about architectural decisions across project boundaries. The system is in use within a small community of software architects already. Their feedback has shown us the benefits of the approach, in terms of the acceleration of project initiation (team orientation), improvement of decision making quality, and simplification of project result sharing (asset harvesting). Users have also pointed out critical success factors: ease of use and dealing with extreme change dynamics.

The remainder of this paper is structured as follows: Section 2 introduces the context of this work and presents related work. Section 3 gathers requirements for AD$_{kwik}$. Section 4 discusses the knowledge engineering aspects of AD$_{kwik}$, while Section 5 focuses on its architecture and implementation. Section 6 presents our preliminary evaluation; Section 7 concludes the paper.

[1] AD$_{kwik}$ stands for “Architectural Decision Knowledge Web Interchange Kit” and pronounces “AD-quick”.

2. Background and Related Work

In the 1990s, *Design Decision Rationale (DDR)* research [4] proposed techniques such as QOC diagrams [5] structuring the decision making process for the general design process of software systems and human-computer interfaces. *Knowledge-Based Software Engineering* (KBSE) proposals such as Argo [6] stressed that tools for designers should support their cognitive needs such as "Reflection in Action, Opportunistic Design, Comprehension and Problem Solving". To achieve this, a "Managed To Do List" was seen as one of several key features. At that time, the regulatory compliance and team collaboration forces were not as dominating as today; therefore, aspects specific to these forces were not addressed. Furthermore, DDR and KBSE did not provide any support specific to architectural decision modeling.

Recently, the DDR ideas were revived and applied to the particular domain of *architectural design decisions* [7][8]. Each decision describes a concrete, atomic design issue for which several alternate solutions with pros and cons exist. Examples for such decisions include: selection of programming language and tools for any development project, of communication protocols in client-server environments, of architectural patterns [9] in certain application domains, and of highly available network topologies in enterprise computing. In general, we defined architectural decisions as "conscious design decisions concerning a software system as a whole, or one or more of its core components" [3]. Many inhibitors for capturing such decisions have been reported, including no appreciation from project sponsors and missing short-term benefits, as well as lack of time, budget, and tool support [10]. Several architectural decision capturing tools have been proposed [11][12]. For example, PAKME [13] is the prototype of an architecture knowledge management system implemented on top of an existing groupware platform. It uses 25 tables to capture various forms of architectural artifacts, including design rationale. PAKME is populated from patterns repositories and the literature.

As potential building blocks for our solution, we also evaluated existing assets such as the Eclipse-based Architects' Workbench (AWB) [4], UML tools, plain HTML and wiki technologies. None of these met all of our requirements. Due to its powerful refactoring capabilities, AWB is well suited for architectural decision content capturing; it can generate reports, thus addressing the regulatory compliance issues. However, it was not designed for knowledge exchange and team collaboration over the Internet. UML tools are strong in capturing analysis- and design-level structure models such as use cases, class, activity, and sequence diagrams; they fall short when it comes to modeling knowledge comprising of text, often semi-structured, combined with other formats, e.g., images and Web links, that motivate and justify the rationale behind the designs captured in UML models.

Plain HTML and standard wiki engines provide flexible human user interfaces when designed and configured appropriately. One advantage is that many development projects already use plain wikis for collaboration and information sharing. Still, using plain wikis does not fully meet our needs. First, no explicit domain model exists since the content is left unstructured and blended with presentation elements in HTML or wiki code. Furthermore, there is no API to access the content apart from the HTML data sent to the browser via HTTP. Thus, it is difficult to populate the system from third party software, or to extract any well-structured knowledge content for further automatic processing.

3. Requirements and Use Cases

We believe that a lack of *collaboration and systematic knowledge reuse features* are key deficiencies of existing approaches. Unlike passive knowledge bases, we aim to guide the user through the content in the spirit of Argo's Managed To Do List, providing *team orientation*. This To Do list should be organized according to domain-specific engagement types and patterns, e.g., business process integration in enterprise computing. To keep its value, the knowledge base must be updated continuously with new decisions, experiences, and rationale gathered both on successful and failed projects. We refer to this collaborative knowledge maintenance activity as *asset harvesting*.

In response to these shortcomings, we propose AD_{kwik}, a Web-centric collaboration system, providing explicit support for sharing and reusing knowledge elements from the domain of architectural decision capturing. We see the following use cases for AD_{kwik}:

- *Obtain* architectural knowledge, captured in decision models, from other projects and CoP.
- *Tailor* imported decision models according to project-specific needs, e.g., filtering content.
- *Involve* experts from other projects and CoP leaders when looking for advice.
- *Manage* dependencies between correlated decisions automatically to guide the user.
- *Share* gained architectural knowledge with other projects and CoP (after sanitization).

Discussion and interaction support, e.g., via email, comments and issue tracking, *document management*, and *versioning* are additional functional requirements shared with existing wiki-like collaboration systems. Integration with other tools is also an important factor to ease the adoption of AD_{kwik}. An *Application Pro-*

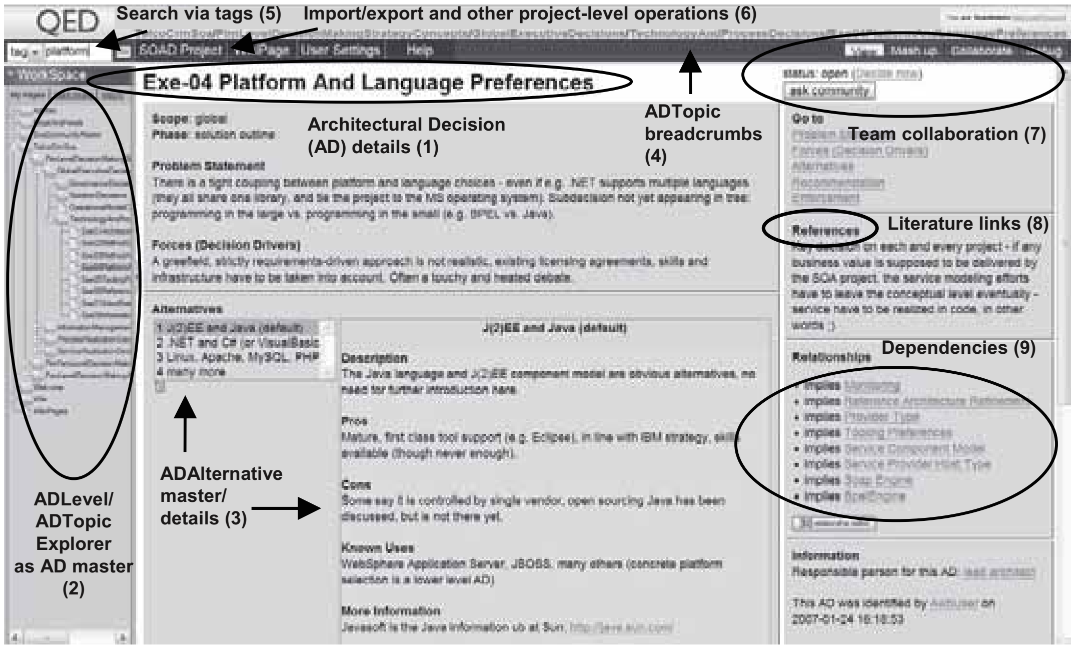
gramming Interface (API) should be provided so that import and export mechanisms to automatically populate the AD$_{kwik}$ content repository can be built, e.g., from requirements management systems and UML design tools. The system must be *highly usable*, as practitioners do not appreciate having to work with yet another tool to fulfill extra obligations. It must be intuitive to browse the content, and users should be attractted to contribute new knowledge. *User management*, including simple *workflow and basic security support* (authentication, authorization) is required if decision making responsibilities are shared within the team. A *thin client* eases deployment and remote access.

4. Knowledge Engineering in AD$_{kwik}$

In this section we present AD$_{kwik}$ from the user's point of view, both including an overview of its domain model and also briefly describing the most important features of the user interface. For the organization of the knowledge content in AD$_{kwik}$, the main elements of the domain model are: *Architectural Decision (AD)*, *ADAlternative*, *ADOutcome*, and *ADTopic*. AD is the core entity describing the context of a decision including decision drivers [3] and relationships with other decisions [8]. ADAlternative instances present solution design options for ADs with their pros and cons. ADOutcome elements record the selection of ADAlternatives and the justification for decisions. ADTopic is a simple hierarchical grouping construct. We also define three *ADLevels* of abstraction in our domain model: *conceptual*, *technology*, and *assets*.

To give an example: on the conceptual level, the choice of programming language and runtime platforms such as application servers and databases are among the key *executive decisions* according to the ontology defined in [8]. A screenshot with this exemplary decision is shown in Figure 1. Several of the domain model elements, e.g., an AD (here: "Platform And Language Preferences") are visible at first glance (1). Applying the *master-details pattern*, ADs can be browsed using the hierarchical ADLevel/ADTopic Explorer (2). Clicking on entries then displays the details about the AD and its ADAlternatives in the main window (1,3). The *breadcrumb pattern* provides additional means of orientation, flattening the ADTopic hierarchy into a link list (4). The knowledge is organized and displayed in a hierarchical structure (2), but also tagged to enable searches (5).

The same user interface can be used for decision *identification*, decision *making*, and decision *enforcement* in a development team: Rather than identifying decisions from scratch, an initial set can be imported, e.g., from AWB, realizing the Obtain use case (6); export features also exist, supporting the Share use case (6). Decision drivers (forces) provide basic decision making support (1); more detailed documentation including scoring spreadsheets and DDR QOC diagrams can be attached to the page. ADs carry owner and status information to further facilitate team collaboration (7). In support of the Involve use case, there are literature links (8). The domain model is shared between projects so that knowledge can be exchanged, e.g., via generated e-mails.


Search via tags (5)
Import/export and other project-level operations (6)
QED
SOAD Project
Page
User Settings
Help
ADTopic breadcrumbs (4)
Architectural Decision (AD) details (1)
Team collaboration (7)
Literature links (8)
Dependencies (9)
ADAlternative master/ details (3)
ADLevel/ ADTopic Explorer as AD master (2)
Work Space

Exe-04 Platform And Language Preferences

Scope: global
Phase: solution outline

Problem Statement
There is a tight coupling between platform and language choices - even if e.g. .NET supports multiple languages (they all share one library, and tie the project to the MS operating system). Subdecision not yet appearing in tree: programming in the large vs. programming in the small (e.g. BPEL vs. Java).

Forces (Decision Drivers)
A greenfield, strictly requirements-driven approach is not realistic, existing licensing agreements, skills and infrastructure have to be taken into account. Often a touchy and heated debate.

Alternatives
1 J(2)EE and Java (default)
2 .NET and C# (or VisualBasic
3 Linux, Apache, MySQL, PHP
4 many more

J(2)EE and Java (default)

Description
The Java language and J(2)EE component model are obvious alternatives, no need for further introduction here.

Pros
Mature, first class tool support (e.g. Eclipse), in line with IBM strategy, skills available (though never enough).

Cons
Some say it is controlled by single vendor, open sourcing Java has been discussed, but is not there yet.

Known Uses
WebSphere Application Server, JBOSS, many others (concrete platform selection is a lower level AD)

More Information
Javasoft is the Java information ub at Sun. http://java.sun.com/

status: open (Decide now)
ask community

Go to
Problem Statement
Forces (Decision Drivers)
Alternatives
Recommendation
Enforcement

References
Key decision on each and every project - if any business value is supposed to be delivered by the SOA project, the service modeling efforts have to leave the conceptual level eventually - service have to be realized in code, in other words ;)

Relationships
- implies [illegible]
- implies Reference Architecture Refinement
- implies Provider Type
- implies Tooling Preferences
- implies Service Component Model
- implies Service Provider Host Type
- implies Soap Engine
- implies [illegible]

Information
Responsible person for this AD: lead architect

This AD was identified by [illegible] on 2007-01-24 16:18:53

Fig. 1. User interface of AD$_{kwik}$: ADTopic Explorer as master, detail views organized according to domain model

ADs can be related to each other. Dependencies between them are shown (9), e.g., “Tooling Preferences”. For example it is no longer required to select between C# and Visual Basic as a programming language if it has already been decided that .NET will not be used as a platform. On the technology and asset level, many more such constraining relationships exist, often buried in vendor information and best practices documents. Explicit, fine-grained representation of decision dependency relationships helps uncovering implicit assumptions, contradictions, and implementation limitations so that a more objective technical discussion becomes possible (Manage use case). Active dependency management leads to a more dynamic and therefore up-to-date knowledge base than static content repositories can provide, which is very important when dealing with the complexity and change dynamics of current enterprise computing environments. It also helps attaining regulatory compliance.

5. Software Architecture of AD$_{kwik}$

Conceptual Design. AD$_{kwik}$ combines the benefits of a *rich Web 2.0 front end* [14] with those of the *domain model pattern* from [9]. The use cases from Section 3 and the user interface design from Section 4 lead to a logical decomposition as shown in Figure 2.

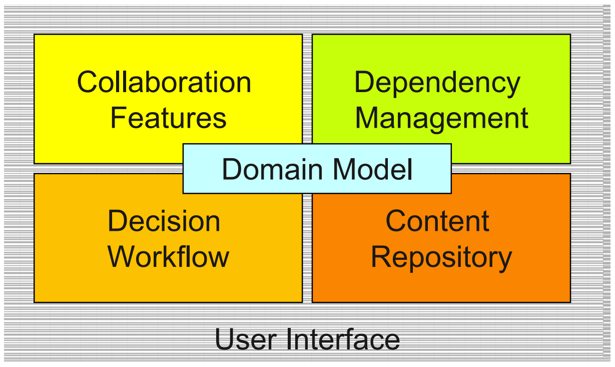


Fig. 2. High-level building blocks of AD$_{kwik}$

There are four functional building blocks: *Collaboration Features, Decision Workflow*, *Content Repository*, and *Dependency Management*. A *Domain Model* is orthogonal to these four building blocks. The Collaboration Features and Decision Workflow building blocks realize the Obtain, Involve, and Share use cases. The Content Repository provides Create, Read, Update, Delete, and Search (CRUDS) operations for the Domain Model elements. Dependency Management structures the knowledge into a graph. As opposed to a simple decision catalog, the graph improves the user’s navigation across related decisions and provides the basis for advanced features such as context-specific, dynamic decision tree *morphing* and what-if *simulations*.

An example of an abstract conceptual decision in the Web services integration domain is the message exchange pattern (request-response vs. one-way) [3], which can be refined into a technology decision dealing with Web Services Description Language (WSDL) contract design (in and out message vs. in message only), which in turn lead to two asset decisions (which SOAP engine and WSDL tool to use when realizing the abstract pattern as a WSDL-described service that can be invoked via SOAP). These decision dependencies modify the asset-level To Do List for the user depending on the outcome of the conceptual and the technology decisions.

To refine this functional view into a logical component model we use logical layering [9] as our governing architectural pattern. The three layers of AD$_{kwik}$ are: *Presentation*, *Domain*, and *Persistence Layer* (Figure 3). The Presentation Layer supports all functional building blocks, e.g., Collaboration Features. Blogs and feeds from vendor forums such as IBM developerWorks [15] and industry thought leaders [16] can be integrated here without development effort. Dependency Management and Decision Workflow are key Domain Layer responsibilities. The Persistence Layer implements the Content Repository as a Relational Database Management System (RDBMS); hence, the full power of the RDBMS technology can be leveraged, e.g., for reporting purposes during technical audits (in response to the regulatory requirements). The Domain Model affects all layers: each model element is represented by one user interface component, related domain layer logic, and a corresponding database entity.

Implementation. The design of AD$_{kwik}$ resembles traditional enterprise application architectures. However, using wiki technology as the presentation layer of such an enterprise application is a new approach requiring an *application wiki* rather than a plain wiki engine. Application wikis extend the user and page management capabilities of plain wikis with application server features and a mash-up API. This allows us to create and manage content programmatically.

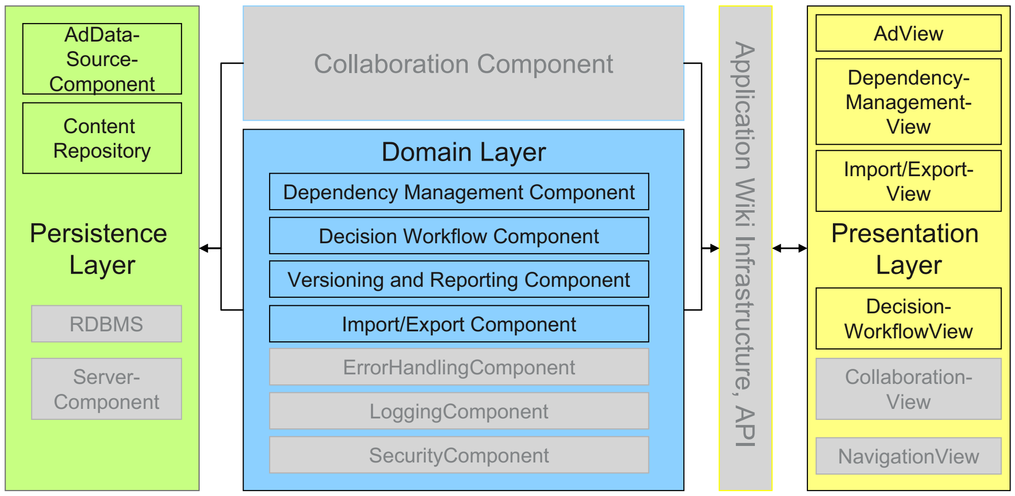


Fig. 3. Logical view on architecture of AD_{kwik}, our Web 2.0 collaboration platform for architectural knowledge exchange

More specifically, AD_{kwik} is implemented in a situational application and Web 2.0 mashup environment called *QEDWiki* [14]. QEDWiki can be characterized as a hybrid wiki engine and PHP application server, providing access to incoming HTTP request data via a command interface. QEDWiki extends the Zend PHP Framework and uses the XAMPP distribution from apachefriends.org. It includes the Apache HTTP server and the MySQL RDBMS. HTTP server and QEDWiki provide the required user management.

Through predefined commands, QEDWiki provides out-of-the-box support for adding comments, attachments, and email threads. We customized and extended these commands to provide native support for our domain model, also using the Dojo JavaScript library in order to provide a user experience as attractive as that of rich clients. The domain layer – comprising the model elements outlined in Section 3 – is implemented in PHP. It accesses the persistence layer via the *active record pattern* [9], requiring little coding effort. The integration with other tools is realized via file import, and a RESTful interface that can be accessed remotely via HTTP.

6. Evaluation

AD_{kwik} has knowledge acquisition and presentation capabilities similar to, for example, PAKME [13]. AD_{kwik} also provides support for dependency management, decision workflow, and decision maker guidance. Its initial content comes from large-scale industry projects conducted since 2001. Since then we have updated the repository continuously with input from additional projects and refactored it many times according to the needs of practitioners [3]. At present, AD_{kwik} contains 130 decision nodes capturing reusable knowledge about enterprise application architectures and Web services integration.

We started to make the system available to selected colleagues and clients in December 2006. To obtain usability feedback, we conducted several workshops. AD_{kwik} already is in use within one industry project. From these engagements, the initial user feedback regarding the value and usability of AD_{kwik} is encouraging: users appreciate that all knowledge required during architectural decision making can be conveniently located in a single place, and that the system comes with a rich set of initial content. Despite the large size of the decision space, early users reported to be productive without major training efforts. AD_{kwik} leverages Web 2.0 application wiki technologies; first users perceive the HTML-based user interface to be compelling and well designed. Thanks to the modeling and collaboration features, we can already report improvements in the quality of the decision making experienced by several AD_{kwik} users. For example, one architect consulting to an IBM client in a SOA coach role reported that he could locate and reuse detailed advice regarding 13 of 15 required decisions related to the usage of Web services [3].

We also have received constructive criticism regarding the challenges of modeling a large and complex decision space facing a high degree of change. Numerous new ADs and even more new ADAlternatives become available almost on a daily basis. If we aimed for completeness, just for enterprise applications organized according to SOA principles, we estimated that AD_{kwik} would contain thousands of decision nodes with numerous dependencies and alternatives. While this complexity is inherent to the problem domain, we run the risk of being criticized for exposing it. However, experienced practitioners report that they prefer to be made aware of this complexity and to have a system that manages it collaboratively, rather than to let the knowledge remain tacit and unmanaged. As we further improve AD$_{kwik}$ based on the feedback of this initial evaluation, we will continue the usability studies on a larger scale.

7. Conclusion

In this paper, we described the conceptual design and implementation of AD$_{kwik}$, a Web-centric collaboration platform for architecture knowledge capturing and exchange. AD$_{kwik}$ supports five use cases, Obtain-Tailor-Involve-Manage-Share. Its design employs both domain modeling concepts and a layered architecture. AD$_{kwik}$ features an API to populate the initial decision model from existing requirements models, reference architectures, and other community assets. The presentation layer is a Web 2.0 application wiki facilitating community collaboration. The persistence layer is implemented as a RDBMS so that database reports can be generated to meet regulatory requirements.

Using Web 2.0 technology for team collaboration and project internal documentation purposes is state-of-the-practice; layered software architectures and domain modeling are known concepts as well. We combine these technologies in a novel way, and apply them to the domain of architectural decision knowledge exchange. Key features of the AD$_{kwik}$ knowledge management approach include pre-population of content, a rich domain model with decision relationship management, support for collaborative decision making, and project result sharing over the Internet. AD$_{kwik}$ has been tested by a small number of early adopters. Extended user tests are planned already.

A critical success factor for AD$_{kwik}$ is to create incentives for users to contribute, not only consume, content, which according to our experience has been a challenge for many industrial knowledge management approaches in the past. Through close contacts with practicing architects, e.g., via a CoP, we have continuous access to up-to-date project results and lessons learned which have to be quality assured and generalized before they can be added to the knowledge base. Architectural decision engineering is a broad, complex, and continuously changing domain. Therefore, keeping the organization of the hierarchical classification of the decision space consistent and manageable is another important factor for future success.

Future research work will investigate additional use cases. For instance, we plan to study design space pruning and recommendation making algorithms. When team collaboration support becomes available on top of the Eclipse platform, additional integration opportunities will arise. Improving usability even further is another focus area. Finally, we are interested in the interdisciplinary aspects of architectural decision making. For instance, will investigate the relationship of architectural decision knowledge with project management concerns such as effort estimations, status reporting, and work breakdown structure creation.

References

[1] Tyree, J., Akerman, A., Architecture Decisions: Demystifying Architecture, IEEE Software, 22 (2005)

[2] Gongla P., Rizzuto C.R., Evolving Communities of Practice: IBM Global Services Experience, IBM Systems Journal Vol. 40, 4/2001

[3] Zimmermann O., Koehler J., Leymann F., The Role of Architectural Decisions in Model-Driven Service-Oriented Architecture Construction, Workshop on Best Practices and Methodologies in SOA, OOPSLA 2006

[4] Lee J., Lai, K, What's in Design Rationale?, Human-Computer Interaction, 6(3&4), 1991

[5] MacLean A., Young R., Bellotti V., and Moran T., Questions, Options, and Criteria: Elements of Design Space Analysis, Human-Computer Interaction, 6 (3&4), 1991

[6] Robbins J. E. , Hilbert D. M. , and Redmiles D. F.: Extending Design Environments to Software Architecture Design, KBSE 1996

[7] Farenhorst R., de Boer R., Deckers R., Lago P., van Vliet H., What's in a Domain Model for Sharing Architectural Knowledge?, SEKE 2006

[8] Kruchten P., Lago P., van Vliet H, Building Up and Reasoning About Architectural Knowledge, QOSA 2006

[9] Fowler M., Patterns of Enterprise Application Architecture, Addison Wesley 2003

[10] Tang W., Ali Babar M., Gorton I., Han J., A Survey of the Use and Documentation of Architecture Design Rationale, WICSA 2005

[11] Abrams S. et al, Architectural thinking and modeling with the Architects' Workbench, IBM Systems Journal Vol. 45, 3/2006

[12] Jansen A., Bosch, J., Software Architecture as a Set of Architectural Design Decisions, WICSA 2005

[13] Ali Babar M., Gorton I., Jeffery R., Capturing and Using Software Architecture Knowledge for Architecture-Based Software Development, QSIC 2005

[14] IBM QEDWiki, http://services.alphaworks.ibm.com/qedwiki

[15] IBM developerWorks, http://www.ibm.com/developerworks

[16] Booch G., Handbook of Software Architecture, http://www.booch.com/architecture

Improving Usability of Web Systems with Similar Business Objectives

Rashid Ahmad, Zhang Li, Farooque Azam
School of Computer Science & Engineering
Beijing University of Aeronautics & Astronautics (BUAA)
No.37, XueYuan Road, HaiDian District, Beijing 100083, P. R. China
{r.ahmad, lily, farooque}@buaa.edu.cn

Abstract
Large web systems are being made so complex that often the users have to make excessive amount of navigational efforts to complete their tasks. This inflicts heavy navigational burden upon users, thus seriously damaging usability of the web based systems. In this paper, we present findings from our recent empirical study of the websites for usability improvement of the web based systems. Our findings are based on the analysis of an experimental data obtained from the prospective students who were invited to interact with the websites of few world class universities to perform a information search task. The data collected in first phase allows us to determine that despite active research in the area of usability engineering, the users still experience heavy 'navigational burden' while performing same task using similar systems. Based on this analysis we proposed to resort to 'Generic Information Architecture' as a standard for organizations in public domain serving similar functions such as universities, hospitals etc. In the second phase, to support our idea, we developed artificial websites (with minimal content and no functionalities) for these universities with generic information architecture and repeated the same experiment. We allowed the prospective students to access these artificial sites from our experimental site. In this paper we report the results from the second phase. Comparing results with those of first phase we observed dramatic improvement in the usability. This work shows the strength of the generic-ness of the information architecture and demonstrates extraordinary improvement in the usability of the web systems which serve similar business objectives.

Keywords
Usability, Generic information architecture, Navigational burden.

1. Introduction
Usability is now viewed as issue of human rights. Guru of usability Mr. Nielsen [1] says that "usability is an ideology -- the belief in a certain specialized type of human rights: The right of people to be superior to technology i.e. if there's a conflict between technology and people, then technology must change. And the right of people to have their time respected i.e. awkward user interfaces waste valuable time". Web professionals talk about user experience as a way to describe user's successes, failures, and thoughts about these events as they browse or complete tasks on the Web. So, for example, we might say that a particular site or application has a positive user experience or a negative user experience. [2] It is in fact the user interface design and the information architecture that earns this positive or negative user experience for the websites. Organizations in fact ignore the preference of functionality over the aesthetics. Many organizations serve similar business objectives e.g. universities, but their user interface would be so different (different in all respects i.e. look and feel, content and navigational structure) that a user performing same task on one institution's website will not feel any improvement in familiarity with the website content while he performs the same task on the other institution's website. The navigational burden experienced by the user in performing the task on one institution's websites remains same while he performs same task on the other institution's website or sometimes is even more. This 'familiarity' and 'navigational burden' is the focus of our work.

In our previous work (we call it phase 1) reported in [3] we conducted an experiment on the websites of four universities of world class repute. We determined that usability of the website is not their focus. Having explored one website for a certain task, the user does not feel any ease in performing same task on another university website. This is what we have proved in our previous work with the help of an empirical study. We then proposed that we should resort to 'Generic Information Architecture" as a standard for all the organization in public domain which serve similar business objectives.

In this work (we call it phase 2), we report the results of the experiment conducted on the artificial websites for these Universities after implementing out idea of generic information architecture. We compared the result with the previous work and found a dramatic improvement in the usability of these website.

This paper is structures as follows: In section 2 gives the motivation. Section 3, gives background and short summary of our previous work i.e. phase 1. Section 4 reports the results of the experiment on the artificial websites with generic information architecture i.e. phase 2. In section 5 we compare the results of both the phases and we witness a remarkable improvement in the usability.

2. Motivation

The reason behind the success of Microsoft products is its generic user interface. Analogically speaking, when a user learns how to work in the environment of Microsoft Word, he takes no time to learn their other products because he then knows for saving a document he has to go to 'File' tab. He also knows that for editing purpose, he can find tools in 'Edit' tab and so on. He only has to learn few extra functions that might be specific to the new Microsoft product.

Moving this analogy to our generic information architecture for web based system; when a user explores one website for first time, will have increased familiarity in his next visit to the Website serving similar function. This will improve and increase understanding & familiarity of the users which will result in reduction of navigational burden.

Other than reducing the navigational burden, many other advantages are envisaged. For example: Statistically speaking, this will reduce the unwanted network traffic amazingly. Unfamiliarity requires that users locate information through trial and error [4], increasing the traffic that the network experiences and the likelihood of choosing incorrect paths will be decreased and will improve efficiency of the servers/routers. Similarly work load of web servers will be reduced proportionally as the user will spend less time to use the services of the servers and will accomplish their task quickly. This reduction in network traffic and web server load can also be researched through empirical analysis. This will also allow implementing ISO and other standards on usability in real sense. This will facilitate the development of CASE tools for automated development of User Interfaces.

3. Background

In this section we discuss the background and give a short summary of our previous work.

3.1 Navigational Burden

In the recent past the web has become the key vehicle for accessing the organizational applications. These applications often provide essential business, educational or government services, and hence the quality of the user interaction and the extent to which users are able to achieve their goals is vital to the success of the system. Given this, effective design of the web application interface is crucial. This in turn raises the issue of what is actually meant by the effective design? Effectiveness can be defined in terms of the ability to support the user's goal; yet for web applications these goals are typically both complex and diverse, with different users having different expectations and objectives. [5] This definition again raises an issue of what is actually meant by "ability to support the user's goals"? This ability can be defined in terms of navigational effort and cognitive burden a user has to suffer before he achieves his goal. Together these efforts can be termed as 'Navigational Burden'. Navigational burden is inversely proportional to the definition of the usability. The heavy the navigational burden, the worst would be the usability of the website.

3.2 Test Hypothesis

In phase 1 we had set following hypothesis which was falsified by the results of our empirical study: "Websites A, B, C and D have common business objectives i.e. serving similar function (such as Universities, Banks etc). Performing a task, of retrieving particular information, on Website A and then repeating the same task on Website B will increase the familiarity of the user with the contents (information architecture) of these websites. Thus, the same user, if repeats the same task on Website C, will acquire the target information with comparatively less time and with minimum possible number of clicks i.e. navigational burden will be reduced."

3.3 Adopted Approach

We had taken the approach as mentioned in [6] that the number of usability problems found in a usability test with n users is: $N\,(1-(1-L)^n)$

Where N is the total number of usability problems in the design and L is the proportion of usability problems discovered while testing a single user. The typical value of L is 31%, averaged across a large number of projects authors [6] studied. Plotting the curve for L=31% gives the following result (Figure 1):

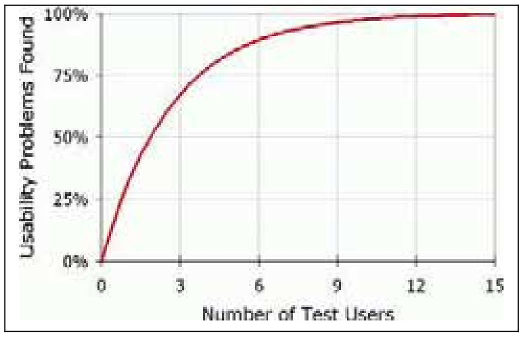


Figure 1: Number of users required to find usability problem, source: [7]

The curve clearly shows that you need to test with at least 15 users to discover the usability problems in the design.

3.4 Research Methodology

In the first phase we invited respondents to interact with the websites of few world class universities to perform a search task of retrieving particular information. We asked the respondents to repeat the same task on all the four websites. We noticed that the respondents did not observe any improvement in 'understanding & familiarity' with the website content while performing the same task on other institution's website. The focus of the study was to examine the websites from the perspective of a student who was considering enrolling in a post graduate course of his interest after graduation in his country and documenting the information on these sites.

3.5 Respondents

A combination of postgraduate and undergraduate students enrolled in different majors and in different universities of different countries was invited to participate in the study.

3.6 Adopted Procedure

We sent an email to the respondents. The email included a Requirement Statement (Figure 2) to explain to them what is required of them to do. The search for information began on the institution's main page.

Requirement Statement

You are an international student and you have recently graduated from your country. Now you are preparing to take admission in Master of Information Technology OR Master of Computer Science OR Master of Software Engineering (what ever option is available in the target University). You have selected 4 Universities (UTS Australia, MIT USA, Oxford University UK, and University of Toronto Canada). You will visit the websites of these Universities to find information according to the following requirements (requirement 1 and 2) given below.

While searching for this information you will have to **count two things**: **1)** Total time it took to reach the required information, **2)** Total number of clicks it took to reach the required information.

When to start counting time and clicks: Time and Click counting starts from the first click after the University home page appears.

When to stop counting time and clicks: When you reach to the page where you can read the desired information or you are ready to download the desired information.

1. **Requirement No 1**: Find the information you will need for admission application for example:
   a. Whether or not, the University is offering the desired course.
   b. Course code, duration, pre-requisites and credits hours etc.
2. **Requirement No 2:** Download admission application form/kit for the desired course (Do not actually download the form/kit but just stop counting time/clicks when you reach to the page where you are ready to download).

Please fill in the following columns:

Site visited	Requirement 1		Requirement 2	
	Time (Min)	Clicks	Time (Min)	Clicks

Access Media (Dialup, ADSL, LAN etc):

Figure 2: Requirement statement

3.7 Results and Analysis (Phase 1)

Data we received from the respondents was analyzed statistically. We found that the results falsified our test hypothesis. We therefore proposed: that we should resort to 'Generic Information Architecture' as a standard for organizations serving similar functions such as universities, banks and hospitals etc. For detail discussion please see [1].

4. Experimentation Phase 2

In the second phase, to support our proposal, we developed artificial websites (with minimal contents and no functionalities) for these universities with Generic Information Architecture and repeated the same experiment as was did in first phase. This work was carried out in two steps. Below we discuss these steps in detail.

4.1 First Step

In first step we just developed an artificial user interface for the front page (home page) of these universities with some generic-ness. These artificial interfaces were integrated to the original information architecture of the universities. We allowed the prospective students to access these artificial sites from our experimental website [8].

A combination of postgraduate and undergraduate students enrolled in different majors and in different universities of different countries was invited to participate in the study. Of 30 students invited, 18 (60%) volunteered to participate. Of 18 volunteers 10 were postgraduate (PhD) students. As per our adopted approach mentioned in section 3.3, we included the first 15 results received, in our data analysis. Rest of the methodology was same as in phase 1. The respondents were given the same requirement as mentioned in section 3.6.

4.2 Second Step

In the next step we developed complete artificial websites introducing premeditated generic-ness to few more layers in the informational architecture behind the homepage and repeated the same experiment. We also repeated this experiment in the lab. The intention of this test in lab was to remove the internet delays and to provide similar test environment to all the respondents. We observed that in both the experiments the usability of the websites improved dramatically.

Again the same students who volunteered for the last experiment were requested to repeat the same experiment. And for experiment in the lab we invited 30 local students. Of 30 students invited, 25 (83%) volunteered to participate. Of 25 volunteers 8 were postgraduate (PhD) students, 17 were postgraduate (MS) students. As per our adopted approach mentioned in section 3.3, we included the first 15 results received, in our data analysis. Of these 15 data 4 were PhD student and 11 were MS. Students. The respondents were given the same requirement as mentioned in section 3.6.

4.3 Results and Analysis (Phase 2)

4.3.1 Data

Data we used for our statistical analysis has been placed in Appendix A. Requirement number 2 appeared to be very easy as in most cases the application form would be on same page as of requirement 1 so we analyzed the data of requirement 1 only. Average clicks were rounded up to whole number.

4.3.2 *Statistical analysis*

First of all we found the parameters Mean: x¯, Variance: s2, Std Dev: s which are given in Table 1 below:

Table 1. Sample set parameters

Parameters	Time				Clicks			
	A	B	C	D	A	B	C	D
Mean: x¯	1.31	0.75	0.59	0.3	5	4	3	3
Variance: s^2	0.47	0.10	0.19	0.1	2	1	1	1
Std Dev: s	0.68	0.32	0.44	0.3	2	1	1	1

Next we found 'Ordinates for Normal Distribution Curves' which suggests that acquired data is normally distributed and a z or t distribution function can be used for data analysis. Again to avoid too much of cluttering with the calculations, we have placed them in Appendix B. Statistical analysis of the data allows us to conclude that average time taken and the number of clicks used to finish the task is getting lesser as the familiarization with similar kind of sites increases.

5. Comparison Of The Results

For comparison we used the following parameters (Table 2):

Table 2. Sample set parameters

Parameters	Average Time				Average Clicks			
	A	B	C	D	A	B	C	D
Phase 1	5.26	5.75	7.43	4.00	7	10	11	6
P-II Step 1	1.31	0.75	0.59	0.35	5	4	3	3
P-II Step 2	2.35	3.12	3.23	2.30	6	7	7	5

Average time for a university is = total time taken by all 15 students to reach to the desired page on that university divided by 15. We compare the results of first phase with the results obtained from both steps in the second phase.

5.1 Usability Improved

The graph in Figure 3 shows that the understanding & familiarity of the user increases after exploring first website (University A).

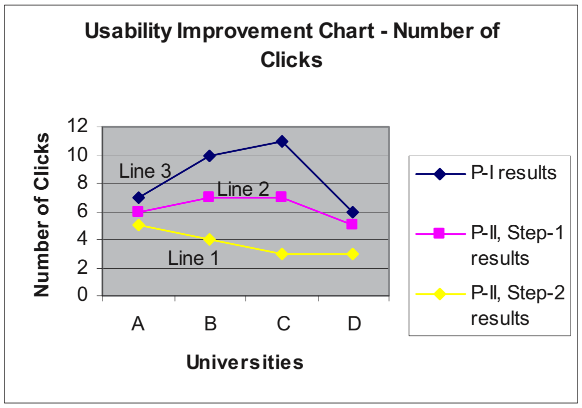


Figure 3: Chart showing improvement in usability

If we look at line 3 in the graph which represents results from phase 1 where the respondents were asked to interact with the original websites of the universities, we notice that the understanding & familiarity of the user does not increase after exploring first website (University A). Rather the users experience heavier navigational burden while performing the same task on websites B & C which is evident from the rising line from University A to B & C.

Line 2 in the graph represents the results from step 1 in phase 2, where the respondents were asked to interact with the partial artificial websites. We developed artificial user interface only for the front page (home page) of the websites of these universities and we integrated these interfaces with the original information architecture of the universities. We can notice improvement in the navigational burden as compared to phase 1 results. The number of clicks reduced which means the user had to waste less time in extra clicks.

However we noticed a remarkable improvement in the results when we asked the respondents to interact with full artificial websites with generic information architecture. Line 1 in the graph represents the results from this experiment. We can see the consistent improvement after exploring first website. A similar improvement was observed in the time the users spent in performing the task. The graph in Figure 4 shows that the understanding & familiarity of the user increases, with respect to time spent, after exploring first website (University A)

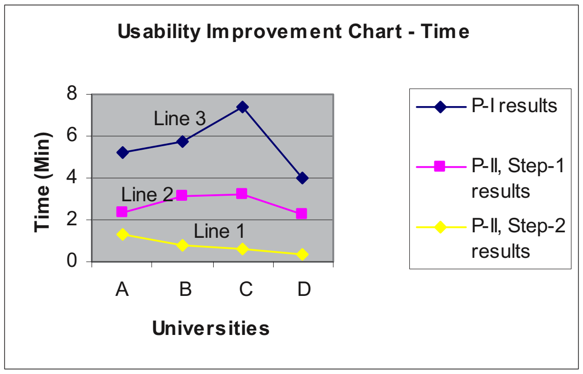


Figure 4: Chart showing improvement in usability

Line 3 in the graph shows the results from phase 1. This line rises from A to B & C which indicates that the understanding & familiarity does not improve after exploring first website (University A). When respondents were asked to interact with the partial artificial website, the usability improved to some extent as is evident from Line 2 in the graph. Again we noticed a remarkable improvement in the results when we asked the respondents to interact with full artificial websites with generic information architecture. Line 1 in the graph represents the results from this experiment. We can see the consistent improvement after exploring first website.

This clearly indicates that the usability improved with incorporating generic-ness in the information architecture of the websites of the organization which serve similar business objectives.

6. Critique

The most striking objection from the opponents is about the aesthetics of the interface. Their view point is that Universities, businesses, and other organizations all wish to present themselves as unique to visitors of their website. Initially when we discussed our idea of 'generic information architecture' with the gurus in one of the penal discussion [9] at International Conference on Web Engineering (ICWE 05), Australia, we heard the same voice there. A keynote speaker [10] Mr. Craig Erry who was also on the penal, however, agreed to our comments that user interfaces can be different in look and feel while still generic in information architecture. This is what we have proved by developing artificial websites for these universities. Figure 5 is the screen shots of these artificial websites which differ in look & feel but have some generic-ness in the information architecture.

7. Future Work And Conclusion

The hypermedia aspects of web systems bring on a slew of new design problems. It has been repeatedly shown in the hypermedia literature that building solid navigation and information architecture in a complex enterprise; we must insure that information is easy to find and that the user will not experience cognitive overhead i.e. navigational burden while exploring the information space.

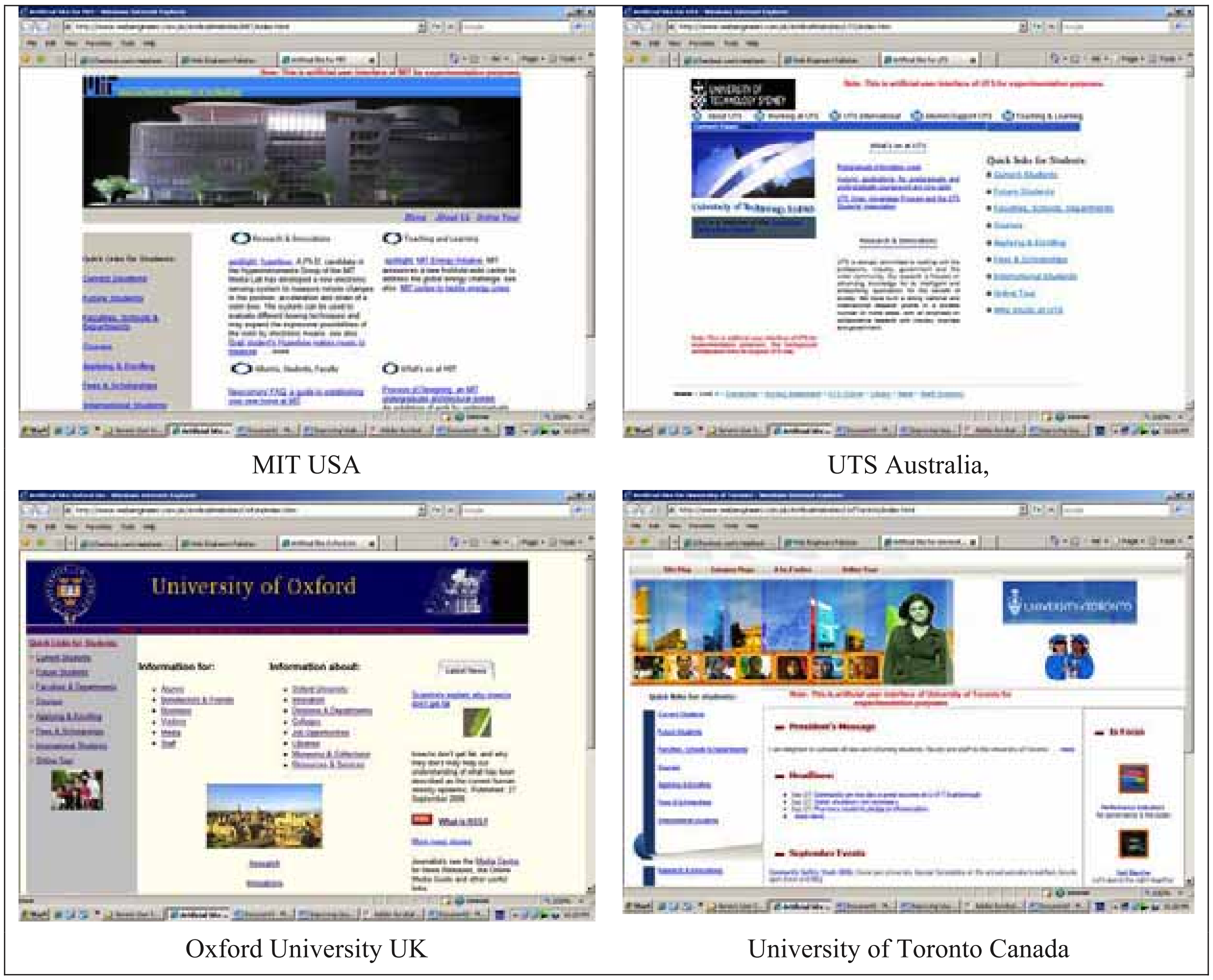


Figure 5: Screen shots of the artificial websites [http://www.webengineers.com.pk/artificialwebsites.htm]

Navigational Patterns build usable navigational architectures by pushing the hypermedia paradigm one step further. Our future work includes mining and collecting some patterns that appear recurrently in web system design. We are also working on translating usability guidelines to pattern language and integrating these usability guidelines into web engineering approaches such as UWE (UML-based Web Engineering) and WebML (Web Modeling Language). We have made pretty good success in this direction and the work will be reported elsewhere. We believe that this work can complement our abstract vision of the system information architecture.

Second, we want to motivate the W3 community to record its navigation experience in the form of patterns, as has been done by other software communities see for example [10]. Finally, we intend to submit our experiences from the work presented in this paper to the World Wide Web Consortium (W3C) so that 'Generic Informational Architecture' can be considered as a standard pattern for organizations serving similar functions such as universities, banks and hospitals etc.

8. References

[1] Nielsen, Jakob. "Usability: Empiricism or Ideology?" Jakob Nielsen's Alertbox June 27, 2005, http://www.useit.com/alertbox/20050627.html

[2] Technical team hesketh.com "User Experience As Corporate Imperative", 2002, publication section, available online: http://hesketh.com/publications/user_experience.pdf.

[3] Rashid Ahmad, Zhang Li, Farooque Azam (2006) "Towards Generic User Interface for Web Based Systems Serving Similar Functions", Proc SERA2006, ISBN 0-7695-2656-X IEEE Computer Society, Aug 2006

[4] Schwartz, J. P. and Norman, K. L. "The Importance of Item Distinctiveness on Performance Using A Menu Selection System," Behaviour and Information Technology, 1986

[5] Xiaoying Kong, David Lowe "NavOptim: On the Possibility of Minimizing Navigational Effort" ISSN: 0302-9743, pp 581, volume 3579 of LN C S, Springer-Verlag, July 2005

[6] Nielsen, Jakob, and Landauer, Thomas K.: "A mathematical model of the finding of usability problems," Proceedings of ACM INTERCHI'93 Conference Amsterdam, 1993.

[7] Nielsen, Jakob. "Why You Only Need to Test with Five Users." Jakob Nielsen's Alertbox March 19, 2000, *http://www.useit.com/alertbox/20000319.html*

[8] WebEngineers: http://www.webengineers.com.pk/artificialwebsites.htm

[9] Craig, E. "Bridging the Gap Between the Requirements and Design" Keynote International Conference on Web Engineering (ICWE2005) Sydney, Australia

[10] Patterns Library: http://hillside.net/patterns

Appendix A, data received from respondents during Phase 2

S/No	Time(Min)				clicks			
	University A	University B	University C	University D	University A	University B	University C	University D
1	0.8	0.54	0.3	0.21	4	4	3	2
2	0.53	0.26	0.14	0.11	3	2	2	2
3	2.5	1	2	1.5	5	3	5	4
4	0.41	0.37	0.24	0.18	3	3	2	2
5	1.05	0.46	0.24	0.21	4	4	3	3
6	1.2	0.63	0.3	0.23	4	3	2	2
7	1.6	0.9	0.6	0.3	4	3	3	2
8	2	1.1	0.7	0.22	6	3	3	2
9	1.1	0.8	0.7	0.33	4	4	2	2
10	2.4	1.5	0.8	0.32	7	4	2	2
11	0.9	0.6	0.6	0.4	3	3	2	2
12	0.88	0.6	0.4	0.4	3	3	3	3
13	1.06	0.7	0.6	0.3	4	3	3	2
14	0.92	0.8	0.6	0.25	4	3	2	2
15	2.3	1	0.6	0.28	6	4	2	2
Total	19.65	11.26	8.82	5.24	64	49	39	34

Appendix B, Statistical Calculations (Phase 2 data)

Confidence intervals: Since our data sets are based on small sample sizes (n=15 in each set), with unknown population variance and mean, we should first of all establish the confidence intervals for the data averages for the websites A, B, C and D, using t-distribution:
95% and 99% Confidence intervals:

(1- α)*100% confidence interval for population mean μ is given by:
For 95%, $\alpha = 0.05$ and for 99%, $\alpha=0.01$, therefore, while n=15, for n-1=14 degrees of freedom,
The values of $t\alpha/2$ are as follows: $t0.025 = 2.145$ & $t0.005 = 2.997$, (these values are taken from standard t-distribution table, Walpole 1976). The interval is given by: $\bar{x} - t\alpha/2 * s/\sqrt{n} < \mu < \bar{x} + t\alpha/2 * s/\sqrt{n}$
Using above formula, confidence intervals for the data sets are as follows:

Uni	Time				Clicks			
	95%		99%		95%		99%	
	Min	Max	Min	Max	Min	Max	Min	Max
A	0.932	1.688	0.782	1.838	3.589	4.944	3.320	5.213
B	0.575	0.927	0.505	0.996	2.938	3.595	2.807	3.726
C	0.344	0.832	0.248	0.928	2.141	3.059	1.959	3.241
D	0.168	0.531	0.096	0.603	1.938	2.595	1.807	2.726

Establishing critical region: From *Time* Data set A we have: $\bar{x}_A$ 1.31, s_A 0.682 & n 15 => v *or* degrees of freedom = 14
We shall test the data at 95% and 99% confidence intervals, therefore: $\alpha = 0.05$ and 0.01. Since we are only interested in the values falling in the lower side of confidence interval, therefore we shall take the negative values of $t_{\alpha,}$ and perform the One-tailed test for 95% and 99% confidence intervals
$t_{-0.05}$ -1.761 & $t_{-0.01}$ -2.624 (Values taken from Walpole, 1976) [The critical regions are based on t-distribution with v=14, therefore, these values will be the same for all data sets for Time as well as Clicks]. We shall systematically check every sample for 95% and 99% confidence interval and conclude about acceptance or rejection of our null hypothesis.

Test Hypotheses (for Time Sets)	Test Hypotheses (for Click Sets)
$H_o : \bar{x}_{B, C \text{ or } D} = \bar{x}_A = 1.31$ $H_a : \bar{x}_{B, C \text{ or } D} < \bar{x}_A = 1.31$	$H_o : \bar{x}_{B, C \text{ or } D} = \bar{x}_A = 4.267$ $H_a : \bar{x}_{B, C \text{ or } D} < \bar{x}_A = 4.267$

Using 0.05 and 0.01 levels of significance, the critical regions are:
(a) $t < -1.761$ (b) $t < -2.624$

Computations:

	Time			Click		
	B	C	D	B	C	D
$\bar{x}$ =	0.751	0.588	0.350	3.267	2.600	2.267
s =	0.318	0.440	0.328	0.594	0.828	0.594
n =	15	15	15	15	15	15

Time	Clicks
• $t = (\bar{x}_B - \bar{x}_A)/(s_B/\sqrt{n}) = -6.82101$, t in critical region for 99% confidence • $t = (\bar{x}_C - \bar{x}_A)/(s_C/\sqrt{n}) = -6.35728$, t in critical region for 99% confidence • $t = (\bar{x}_D - \bar{x}_A)/(s_D/\sqrt{n}) = -11.3453$, t in critical region for 99% confidence	• $t = (\bar{x}_B - \bar{x}_A)/(s_B/\sqrt{n}) = -6.52438$, t in critical region for 99% confidence • $t = (\bar{x}_C - \bar{x}_A)/(s_C/\sqrt{n}) = -7.79512$, t in critical region for 99% confidence • $t = (\bar{x}_D - \bar{x}_A)/(s_D/\sqrt{n}) = -13.0488$, t in critical region for 99% confidence

Conclusions:
There is not enough evidence to accept H_o, either in Time Data or in Click Data. Conversely, there is enough evidence in the acquired data to suggest the H_a is true. Therefore we conclude that the data indicates that the average time taken is getting lesser as the familiarization with similar kind of sites increase.

Processing Manipulations of Context Information on the Web

Roberto De Virgilio
Dipartimento di Informatica e Automazione
Università degli studi Roma Tre
devirgilio@dia.uniroma3.it

Abstract. *Recently, the literature proposes many approaches to system specification and interoperability based on the used of formal models. We consider Adaptive Web Applications: a relevant requirement is the ability to capture and manipulate context information expressed in different and heterogeneous formats. To this aim translating heterogeneous contexts from one representation into another is an important issue to facilitate the integration of such heterogeneous data and the maintenance of heterogeneous replicated data. In this paper we describe a general formalism to facilitate the integration of context information and, based on such description, a mechanism to achieve automatically translations between heterogeneous contexts. Translations are specified as compositions of elementary steps, defined by means of special operations. With these operations, we show how it is possible to reason on the models that are provided as input and output for a translation.*

1. Introduction

The increasing popularity of mobile devices, such as laptops, mobile phones, and personal digital assistants is enabling new classes of applications targeting environments characterized by being dynamic, mobile, reconfigurable, and personalized spontaneously. These applications and their targeted environments raise challenging problems for application developers, as they have to be aware of the variations in the execution context such as location, time, users' activities, and devices' capabilities in order to tune and adapt the behavior and functionalities of applications. In adaptive web system, it is widely recognized that the management of context information is a fundamental requirement to take into account the limited resources of mobile systems, to select data relevant to the user, to improve the interoperability with the environment, and, in general, to make the interaction with the system truly adaptive to highly change scenarios of use. In this framework, the interoperability of such applications is an important task. Metadata is critical to all aspects of interoperability within any heterogeneous environment. In fact, metadata is the primary means by which interoperability is achieved and the overall strategy for sharing and understanding metadata consists of the automated development, publishing, management, and interpretation of models [1, 2, 5].

This scenario changes the role of context information and semantics as compared to traditional information systems [7, 8], as now the physical environment immediately affects and interacts with the processing of data and communication. Unfortunately, current technologies do not fully support flexible and self-adapting models based on context. For example, if a mobile user today wants to use the computing resources of a new environment, he/she has to obtain the necessary information, assess it (format, semantics) and figure out manually how to continue his/her activities with the local resources of that new environment. This is unacceptable in pervasive computing environment and neglects the advances which have been made in other research domains dealing with context information and semantics.

The most relevant context modeling approaches identify a context as a set of *profiles* [11]. It is important to understand how the set of context models of interest can be defined, in a simple way, and the individual models be specified. The Object Management Group has introduces a number of important standard such as MOF [4], UML [10], and XMI [9] for data modeling in various research areas. Generally the MOF is used for modeling the adaptation specifications of a web application rather than context information, often considered trivial input values. To this aim, in [3] we studied the notion of *generic profile* and we represented contexts by means of the *General Profile Model* (GPM), a conceptual model for the uniform description of the various aspects of a context. Based on GPM, we proposed a rule-based *conversion* process to translate profiles from a representation into another. However the web designer has to define and select manually the different and right translations between each couple of representations. This can be a critical problem for the interoperability of adaptive systems, because many manipulations of context information are needed. In the plethora of context representations, different models exist, often just small variations of other ones. Many uses of a context involve managing the change in models and the transformation of data from one model into another. The set of possible models is potentially huge, as they are obtained by variants of constructs. It is probably the case that some combinations of variants are not meaningful, but with more constructs and more variants for each of them, we get a combinational explosion of the number of models. With this size for the space of context models, it would be hopeless to have translations between every pairs of models, as with n models we would need n^2 translations. If we assume that our GPM generalizes all the other models then we need one translation for each model (from GPM) but n can be still an unmanageable number. As a consequence it becomes meaningful to specify translations with reference to them.

This paper proposes a middleware data model that serves as a basis for the task. Our formalism permits to define different *Profile Models* to describe the primitives involved and the structure of a context model. Then we provide a mechanism to generate automatically a conversion from a Profile Model into another. The idea is to have many "basic" translations that perform *elementary steps* and can be combined to form actual conversions, but with a lot of reusability. With fine-grained decompositions, the basic steps can be reused in many other conversions. In this work, we illustrate the notion of *Mod* operation that executes an elementary step. Therefore a major issue arises: given a set of basic translations, how do we build the actual conversion we need? Or, at least, how do we verify that a given sequence of basic translations produces the model we are interested in? The idea is that all the Mod operations are assumed to be correct, and so, if we properly apply sequence of them, we obtain correct translations (this is sometimes called the "axiomatic approach" [1]).

The rest of the paper is organized as follows. In Section 2, we illustrate the basic notions of our context modeling approach, defining our formalism and the notions of Profile Model. In Section 3 we illustrate the Mod operation and describe the conversion process. In Section 4 we describe a practical implementation. Finally, in Section 5 we draw some conclusions and sketch future work.

2. Context Modeling

In this section we illustrate a general formalism to represent contexts, and the special notion of Profile Model.

2.1. General Profiles

A *general profile* is a description of an autonomous aspect of the context into which the Web site is accessed and that

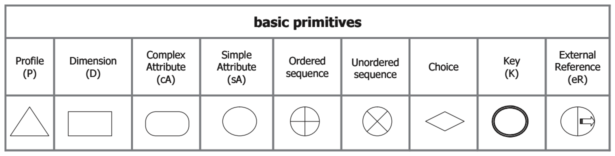


Figure 1. Basic Primitives of general profiles

should influence the structuring and presentation of its contents. Examples of profiles are descriptions of the user, the device, the location, and so on.

Our formalism presents a quite limited set of constructs, that we call *basic primitives*, to describe, in a graphical way, a conceptual representation of a context, as shown in Figure 1. Principal constructs are the *dimension* and the *attribute*. A *dimension* is a property that characterizes a profile. Each dimension is described by means of a set of *attributes*. Attributes can be *simple* or *composite*. A simple attribute has a domain of values associated with it (printable values such as string, integer, boolean and so on), whereas a composite attribute has a set of (simple or composite) attributes associated with it. It is possible to distinguish two roles for a simple attribute; it can be a *key* or an *external reference* to a component of a profile. It is possible to represent *ordered* and *unordered sequences* of attributes and a *choice* of a set of attributes, whose instances can be chosen among instances of the attributes (for instance a `<choice>` of an XML-schema). Finally the *cardinality* is expressed as a pair of integer values (*Min*,*Max*) that corresponds to a primitive whose instances are sets of instances of the primitive associated with it (these sets must have a cardinality included between *Min* and *Max*).

Definition 1 (Profile and context) *Given a set of dimensions $D_1, \ldots, D_n$ over a set of attributes $A_{i,1} \ldots A_{i,k_i}$ $(1 \leq i \leq n)$ respectively, a (general)* profile *over $D_1, \ldots, D_n$ is function that associates with each simple attribute of every dimension a value taken from its domain. A* context *is a collection of profiles.*

As an example, Figure 2 reports a graphical representation for the context schema of a client A composed by profile schemes for the user and the location. For instance, a user profile of a client can be represented by means of the dimension account, described by a choice between the simple attribute e-mail (so an access without a registration) or the complex attribute login, composed by the simple attributes username and password (so an access with a registration), and so on. The location profile presents the dimension GSM to characterize location information from the GSM cells: each cell has a unique ID, expressed by the key Cell-ID. We indicate the base type associated inside the attribute (for instance I means integer, R real and S string).

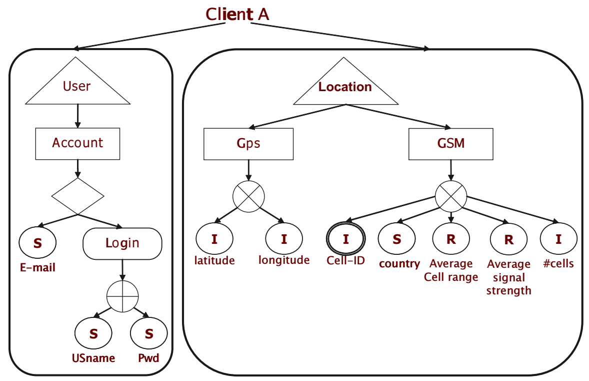


Figure 2. An example of context

2.2. Profile Model

Through our formalism, we want to describe the model of a profile by means of the constructs involved and its structure. To this aim we introduce the following notion of *Profile Model*.

Definition 2 (Profile Model) *We define a Profile Model PM as a special Multi-Graph $< L, V, E, \omega, v_c >$ where*

- *L is the set of labels, representing the basic primitives of our formalism such as $\{ \triangle, \bigcirc, \dots \}$)*
- *V is the set of vertexes $\{v_1, v_2, \dots\}$, each one represented by a pair (OID, l), where OID is the identifier and $l \in L$ is the label of the vertex*
- *E is the set of direct weighted edges (v_i, v_j) that means "v_i is composed by v_j". The graph allows self loops such that $v_i = v_j$*
- *ω is the function to assign a weight to an edge in E. A weight on an edge (v_i,v_j) is a pair (min,max) that represents the type of cardinality of v_j respect to v_i. The weights admissible are in $\{(0,1),(1,1),(0,n),(1,n),(n,n)\}$ and it is possible to establish an ordering between two weights $\omega_1 = (a,b)$ and $\omega_2 = (c,d)$ such that $\omega_1 \leq \omega_2$ if $a \leq c$ and $b \leq d$, where it is $0 < 1 < n$.*
- *v_c is the vertex (OID,$\triangle$) representing the center of the graph.*

A Profile Model would describe the representation used for a profile by means of primitives involved and how these primitives are combined (or composed) in the profile. For instance, Figure 3 shows the Profile Model PM_1. The model can articulate a profile in several dimensions. Each dimension can be composed by simple attributes of string or integer values, or by composite attributes. Each composite attribute can be composed by simple attributes of string or integer values or, recursively, by composite attributes of the same type. In PM_1, the function ω assign to each edge the type of cardinality *(1,n)*: each component can be articulated by other components using the cardinality *(1,1)* or *(1,n)*. Figure 3 shows an example of profile P described by PM_1. In a profile schema the cardinality of default is *(1,1)*. In a Profile Model each vertex is identified by an OID, for instance generated by using Skolem functions [6], and characterized by a label, that represents a basic primitive. So the set L represents the basic primitives involved in a profile: for instance in the previous example we have dimensions, simple and composite attributes, and basic types such as string and integer. The set E represents how the primitives involved are composed in a profile. The function ω describes the minimum and maximum cardinality to articulate a component.

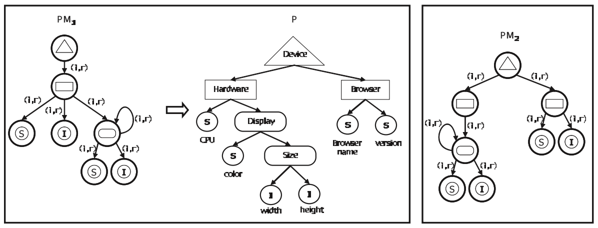


Figure 3. An example of Profile Model

A fundamental aspect is that different Profile Models can be compared making use of a subsumption relationship, denoted by $\lhd$. Intuitively, a Profile Model PM_1 subsumes a Profile Model PM_2 if PM_1 represents all the types of profiles that PM_2 can represent, and more others.

More precisely, we first say that two vertexes $v_1 = (OID_1, l_1)$ and $v_2 = (OID_2, l_2)$ are similar, denoted by $v_1 \approx v_2$, if $l_1 = l_2$. Then two pathes of vertexes are similar if the vertexes of the pathes, in order, are similar. The subsumption relationship is then defined as follows.

Definition 3 (Subsumption of Profile Models) *Given two Profile Models $PM_1 =< L_1, V_1, E_1, \omega_1, v_{c_1} >$ and $PM_2 =< L_2, V_2, E_2, \omega_2, v_{c_2} >$, we say that PM_1 is subsumed by PM_2, $PM_1 \lhd PM_2$, if for each vertex $v_1 \in V_1$ it exists a vertex $v_2 \in V_2$ such that for each edge $(v_1, v') \in E_1$ it exists the edge $(v_2, v'') \in E_2$ so that $(v_1, v') \approx (v_2, v'')$ and $\omega_1((v_1, v')) \leq \omega_2((v_2, v''))$.*

As an example, given the Profile Models reported in Figure 3, we have that $PM_2 \lhd PM_1$.

We can define *meet*, *join* and *difference* between Profile Models. Intuitively, given two Profile Models PM_1 and PM_2, the *meet* ($\sqcap$) represents the greatest Profile Model that is able to generate the productions (types of profile)

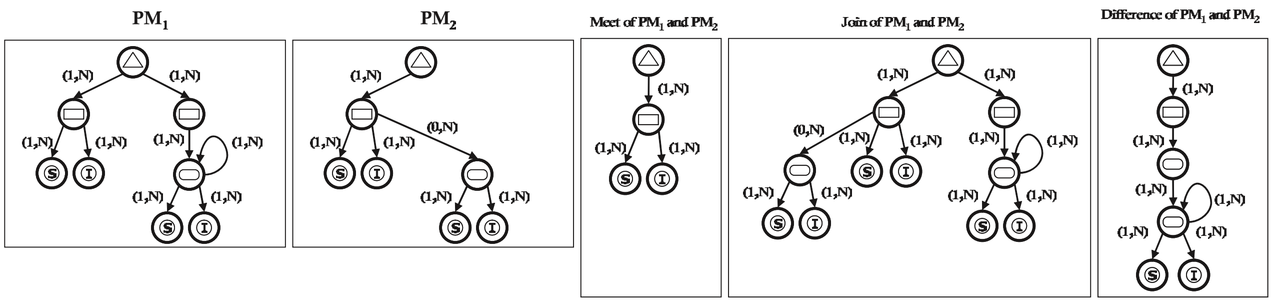


Figure 4. Example of operators ($\sqcap$), ($\sqcup$) and ($-$)

in common between PM_1 and PM_2, the *join* ($\sqcup$) the least Profile Model that is able to generate all the productions of PM_1 and PM_2, and the *difference* ($-$) the greatest Profile Model that is able to generate the productions of PM_1 not in common with PM_2.

Definition 4 (Meet of Profile Models) *The* meet *of two Profiles PM_1 and PM_2, denoted by $PM_1 \sqcap PM_2$, is a Profile Model PM such that, $PM \lhd PM_1$, $PM \lhd PM_2$ and, for each Profile Model $PM' \neq PM$ such that $PM' \lhd PM_1$, $PM' \lhd PM_2$, it is the case that $PM' \lhd PM$.*

Definition 5 (Join of Profile Models) *The* join *of two Profile Models PM_1 and PM_2, denoted by $PM_1 \sqcup PM_2$, is a Profile Model PM such that, $PM_1 \lhd PM$, $PM_2 \lhd PM$ and, for each Profile Model $PM' \neq PM$ such that $PM_1 \lhd PM'$, $PM_2 \lhd PM'$, it is the case that $PM \lhd PM'$.*

Definition 6 (Difference of Profile Models) *The* difference *of two Profile Models PM_1 and PM_2, denoted by $PM_1 - PM_2$, is a Profile Model PM such that, $PM \lhd PM_1$, $PM \sqcap (PM_1 \sqcap PM_2) = \emptyset$ and, for each Profile Model $PM' \neq PM$ such that $PM' \lhd PM_1$, it is the case that $PM' \lhd PM$.*

Figure 4 shows some example of meet, join and difference. The comparison between two Profile Models can be measured by the following idea of distance.

Definition 7 (Distance between Profile Models) *Given two Profile Models $PM_1 =< L_1, V_1, E_1, \omega_1, v_{c_1} >$ and $PM_2 =< L_2, V_2, E_2, \omega_2, v_{c_2} >$, and the difference $PM = PM_2 - PM_1 =< L, V, E, \omega, v_c >$, the distance of PM_1 from PM_2, $\delta(PM_1, PM_2)$, is given by $|L_2 - L_1| + |E|$.*

The distance $\delta(PM_1, PM_2)$ measures how much the Profile Model PM_1 can't interpret the Profile Model PM_2. In other words, this distance evaluates how many primitives and combinations of them in PM_2 are not comprehensible by PM_1. For instance, in the Figure 4 the distance of PM_2 from PM_1, $\delta(PM_2, PM_1)$, is 6. In this case PM_2 can't interpret the composition of composite attributes in other ones (the self-loop).

3. Conversion of Contexts

Given a source profile PI_s of a profile schema PS_s, described according to a Profile Model PM_1, we need to generate a *target* profile PI_t described according to a Profile Model PM_2, containing the same information as PI_s. This is done by a *conversion* ($\mathcal{C}$) of PI_s from PM_1 into PM_2, denoted as $PI_t = \mathcal{C}_{PM_1 \Rightarrow PM_2}(PI_s)$. In this section we present a mechanism to generate automatically a conversion by means of special *Mod* operations.

3.1. Mod (Γ)

An operation *Mod* (Γ) takes as input a profile (schema) P_1 according to a Profile Model PM_1 and returns a profile (schema) P_2 according to a Profile Model PM_2 containing the same information of P_1, denoting such as $\Gamma_{PM_1 \rightarrow PM_2}(P_1) = P_2$. A Mod operates an elementary translation of a profile (schema) from a representation into another, for instance of some primitives. For instance the Figure 5 represents a Mod operation to translate a profile schema from a Profile Model that articulates the schema on n levels, making use of composite attributes, into a Profile Model that articulates the schema on 2 levels, making only use of simple attributes (and external references) and no composite ones. Intuitively given a Mod operation $\Gamma_{PM_i \rightarrow PM_j}$, if we apply the operation to a profile P described by a Profile Model PM such that $PM_i \lhd PM$ then the operation will translate the part P' of P described by PM_i into P'' described by PM_j. The resulting profile will be described by a Profile Model such as $(PM - PM_i) \sqcup PM_j$. So we can denote also the following $\Gamma_{PM_1 \rightarrow PM_2}(PM) = (PM - PM_1) \sqcup PM_2$.
Let's consider a prefixed set of Mod operations $\mathcal{F} = \{\Gamma_{PM_1 \rightarrow PM_2}, \Gamma_{PM_3 \rightarrow PM_{4,\ldots}}\}$. From $\mathcal{F}$, we can define a special graph as follows

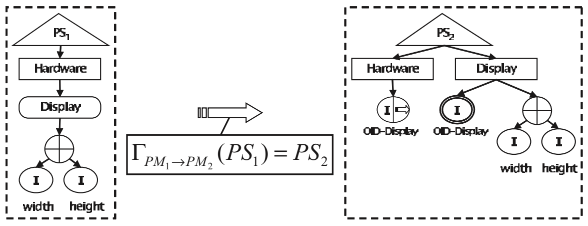
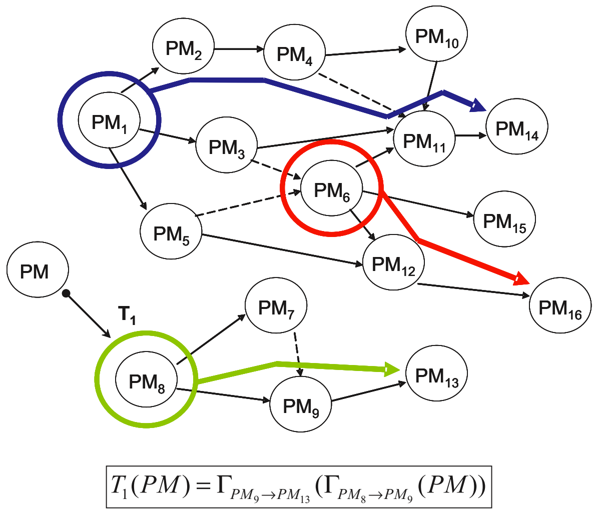


Figure 5. An example of Mod operation



PM1
PM2
PM3
PM4
PM5
PM6
PM7
PM8
PM9
PM10
PM11
PM12
PM13
PM14
PM15
PM16
PM
T1
$T_1(PM) = \Gamma_{PM_9 \rightarrow PM_{13}}(\Gamma_{PM_8 \rightarrow PM_9}(PM))$

Figure 6. An example of Model Graph

Definition 8 (Graph of Profile Models) *We define Model Graph on a set of Mod operations $\digamma$ as a graph $MG = \{N_{MG}, E_{MG}\}$ not connected, directed and acyclic, where*

- *N_{MG} is the set of nodes representing Profile Models*
- *$E_{MG} = \{L_{MG}, U_{MG}\}$, where*
  - *L_{MG} is the set of direct edges (n_i, n_j) such that $\exists\, \Gamma_{n_i \rightarrow n_j} \in \digamma$*
  - *U_{MG} is the set of dashed edges (n'_i, n'_j) such that $n'_i \lhd n'_j$*

Figure 6 shows an example. Moreover, we introduce the following notion of *grafted node* in a Model Graph MG, to compare a Profile Model PM with MG.

Definition 9 (Grafted Node) *Given a model graph MG and a model set PM, a node $PM' \in N_{MG}$ is a grafted node respect to PM if $PM' \lhd PM$ and we say that PM is comparable with MG.*

In MG a path $\{PM_1, PM_2, \ldots, PM_{n-1}, PM_n\}$ represents a set of elementary translations that we can apply sequentially to a Profile Model PM comparable with MG, such that $\Gamma_{PM_{n-1} \rightarrow PM_n}(\ldots(\Gamma_{PM_1 \rightarrow PM_2}(PM)))$.

3.2. Automatic generation of a Conversion

Given a set of Mod operations $\digamma$, a conversion $\mathcal{C}$ is performed by a sequence of Mod operations from $\digamma$.

Definition 10 (Conversion) *Given two Profile Models PM_1 and PM_2 and a set of Mod operations $\digamma$, a conversion $\mathcal{C}$ from PM_1 into PM_2 is a sequence $\{\Gamma_1, \Gamma_2, \ldots, \Gamma_n\} \in \digamma$ such that $\delta(\Gamma_n(\Gamma_{n-1}(\ldots(\Gamma_1(PM_1)))), PM_2) = 0$.*

We have to select the sequence of Mod operations of $\digamma$ such that the Profile Model resulting by applying the sequence to PM_1 has distance zero from PM_2.
Therefore a conversion from PM_1 into PM_2 can be generated automatically as follows.

1. we generate a Model Graph MG from $\digamma$. For the nature of the Mod operations, the Model Graph is not connected, directed and has to be acyclic.
2. we set the *current* Profile Model PM_c to PM_1, the *target* Profile Model PM_t to PM_2 and the sequence S of Mod operation to $\emptyset$.
3. we have to select one or more pathes in MG. So we search in MG the set of grafted nodes $Gn = \{gn_1, gn_2, \ldots, gn_k\}$ respect to PM_c, ordered respect the increasing distance from PM_t;
4. we operate a backtracking strategy on each gn_i of Gn.
5. we select a path p of MG starting from gn_i as follows: (i) from a node select the outgoing edge that applied to PM_c returns the Profile Model with the lowest distance from PM_t, (ii) stop the selection if we arrive in a node without outgoing edges or each outgoing edges makes greater the distance from PM_t.
6. add the selected path p to S and recalculate with p the new PM_c
7. if $\delta(PM_c, PM_t) = 0$ then return S, otherwise select a new set of grafted nodes in MG respect to PM_c and iterate the backtracking.
8. the fault condition of the backtracking is to obtain a set of grafted node empty. So we go back considering S without the last path added and selecting another grafted node, as shown in Figure 7.
9. if it is not possible to find a sequence S such that $\delta(PM_c, PM_t) = 0$, then the web engineer has to write the missing elementary translations and update the set $\digamma$.

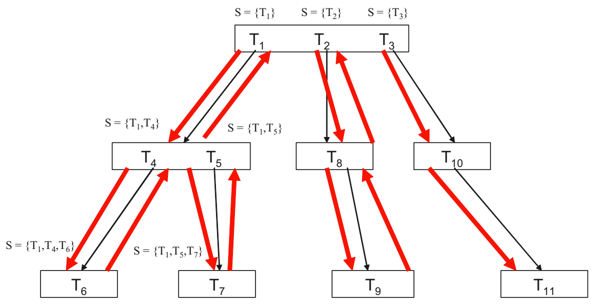


Figure 7. The backtracking strategy

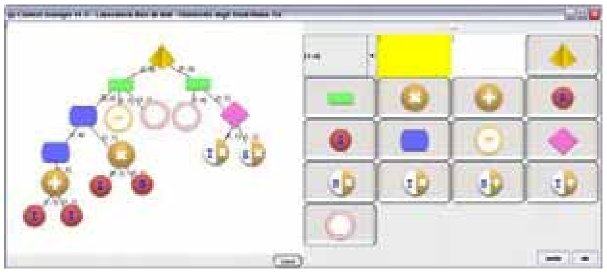


Figure 8. User-interface of the tool

4. Implementation

It was extended the tool presented in [3], introducing a module to define Profile Models. Figure 8 shows a screenshot of the user-friendly interface. The Mod operations are written as production rules by the syntax described in [3]. Profile Models are implemented using RDF/XML syntax and the Mod operations as a set of RDF inference rules (`http://jena.sourceforge.net/inference/`).

5. Conclusions and Future Work

In this paper we proposed a data model that consists of a general abstraction of existing formats that allows to define different context models. The data model proposed is enriched by special operators (Mod) to embed the conversion of context information from one representation into another. The conversion process is resulting by a composition of elementary translation steps. We illustrated a technique to reason and generate automatically a conversion by the form of a sequence of Mod operations. The results presented in this paper are subject of further conceptual and practical investigation. From a conceptual point of view we believe that it is possible to define a framework for *context management*, as a new approach to manipulating context information. We are defining an algebra of operators (a lattice) to embed the main functionalities of context management. This framework can be an important support in different application scenarios such as contexts translation, contexts integration or contexts classification by means of clusters. From a practical point of view, an important issue strongly suggests that an implementation of context management would provide major programming productivity gains for a wide variety of context management problems. Of course, to make this claim compelling, an implementation is needed.

References

[1] P. Atzeni and R. Torlone. Management of multiple models in an extensible database design tool. In *Proc. of the 5th Int. Conference on Extending Database Technology: Advances in Database Technology (EDBT'96), Avignon, France*, 1996.

[2] P. A. Bernstein. Applying model management to classical meta data problems. In *Proc. of the 1th Biennial Conference on Innovative Data Systems Research (CIDR'03), CA, USA*, 2003.

[3] R. De Virgilio and R. Torlone. Modeling Heterogeneous Context Information in Adaptive Web Based Applications. In *Proc. of 6th ACM Int. Conference on Web Engineering (ICWE'06), California, USA*, 2006.

[4] M. J. Emerson, J. Sztipanovits and T. Bapty. A MOF-Based Metamodeling Environment. In *J. UCS, 10(10):1357-1382*, 2004.

[5] R. Hull and R. King. Semantic database modelling: Survey, applications and research issues. In *ACM Computing Surveys, 19(3):201-260*, 1987.

[6] R. Hull and M. Yoshikawa ILOG: Declarative Creation and Manipulation of Object-Identifiers In *Proc of the 16th Int. Conference on Very Large Databases (VLDB'90),CA*, 1990.

[7] R. Kaschek, K. D. Schewe and B. Thalheim. Integrating Context in Modelling for Web Information Systems. In *WES, LNCS, Vol. 3095, pp. 77-88*, 2003.

[8] R. Pitrik. An IntegratedViewon theViewing Abstraction: Contexts and Perspectives in Software Development, AI, and Databases. In *Journal of Systems Integration, 5(1):23-60*, 1995.

[9] Object Management Group. XML Metadata Interchange (XMI) v2.0. 2005.

[10] J. Rumbaugh, I. Jacobson and G. Booch. The Unified Modeling Language Manual. *Addison Wesley*,1998.

[11] T. Strang and C. Linnhoff-popien. A context modeling survey. In *Proc. of Int. Workshop on Advanced Context Modelling, Reasoning and Management. England*, 2004

A Tag-Level Web-Caching Scheme for Reducing Redundant Data Transfers

Steven E. Cox, Du Zhang, Jinsong Ouyang
Dept. of Computer Science
California State Univ. Sacramento
Sacramento, CA 95819
{coxs, zhangd, ouyangj}@ecs.csus.edu

Abstract. Markup languages such as HTML and XML are popular formats for data exchange across networks. This is in spite of their inefficient utilization of network bandwidth. We introduce a compaction technique that reduces the overall size of collections of markup data by identifying and eliminating redundant copies of common elements. This general-purpose technique is applied to a proxy-based web-caching scheme, Tag-Level Web Caching (TLWC). A highlight of TLWC is its recognition and utilization of nested markup elements. Markup elements are treated as cacheable objects, and their nested syntax is used to identify maximal commonalities between requested objects and reference objects. We implement and evaluate a prototype TLWC system consisting of server-side proxy and client-side proxy components that intercept HTTP requests from a web browser. The prototype is tested on a set of web sites and is shown to reduce overall data volumes significantly for most of the sites.

1. Introduction

While markup languages such as HTML and XML allow easy exchange of data over networks, it is well known that they do not make very efficient use of network bandwidth. Markup documents are text-based, usually uncompressed, and often include verbose meta-data. Making matters worse, typical client-server data exchange sessions, such as when a World Wide Web user browses a web site, involve significant amounts of repetitive data transfer requests.

Current web-caching schemes mitigate the amount of repetitive data transfers. Web browsers utilize caches for HTML pages and other static files, while many popular web sites employ distributed content caches to store their content closer to web clients. However, a high amount of repetition still occurs with markup language data transfers. In the case of web pages, standard URL-based caches are only effective for static pages. Many web sites serve dynamic content where pages are continually changing.

A solution to this problem is Tag-Level Web Caching (TLWC). Markup elements, rather than entire pages, are cached on the client. In this scheme, the web server (or a proxy server standing between the web server and the client) gives each element an identifier, which it passes along to the client when the element is first transferred. The server stores the element and its identifier in a hash map for future use. In this data structure, the element string (or a digest of this string) serves as the key while the identifier is the value. Any element that contains child elements is abbreviated by replacing child element strings with special marker tags that reference the child elements.

The client also caches the element and its identifier in a hash map. On this end, the identifier is the key and the element string is the value. The second time the element is encountered in a response message, the server replaces the element with its identifier. As it parses the response message, the client replaces the identifier with the element with which it is associated.

Compared to traditional web caches, TLWC has the following potential benefits:

- It can eliminate fine-grained redundancies. In contrast, most web caches work at the document level, meaning they cannot deal with small differences between documents.
- TLWC can work in conjunction with data compression techniques such as *gzip*.
- TLWC works transparently to existing browser caches.
- When TLWC is implemented using proxies, no participation by the origin web server is required.
- TLWC normally requires no additional HTTP round trip times (RTTs).

2. Background

Automated methods of reducing redundant data transfers over the Web can be generalized as falling into three categories: caching, compression and differencing. Of these, caching is most widely used. Most web caches index objects using URLs. If a requested URL matches a cached URL, and if it is determined that the cached version is up-to-date, a cache hit can occur [1]. However, there are three common occurrences that confound URL-indexed web caches: 1) Many web sites are dynamic, meaning that each request to a single URL can produce a different response; 2) identical web objects are often served from different URLs, a phenomenon known as aliasing [2]; and 3) the complexity of HTTP 1.1 makes it difficult to configure servers and proxies to perform caching optimally [1][2].

Data compression is also widely used on the Web. Two web compression algorithms, *gzip* [3] and *deflate* [4], are freely available and well supported by HTTP 1.1, by most web servers and by most browsers. However, compression works only on one page at a time; thus, redundancies distributed over multiple documents are missed.

Differencing, also known as delta encoding or delta compression, is the expression of one object as a difference, or delta, of another. A proposal from the Internet Society published in RFC 3229 [5] outlines how delta encoding can be used to compress HTTP responses. However, the difficulty of managing prior instances of web pages has slowed the actual use of delta encoding on the Web.

3. Related Work

The inability of URL-based web caches to suppress the transfer of duplicate objects is the subject of a detailed study by Mogul [6]. The paper also describes how MD5 digests [7] can be used to index cached web objects by a hash value rather than by a URL. Mogul and Kelly expand this idea in [2] and (with Chan) in [8]. MD5 digests are also used by Santos and Weatherall in an approach to duplicate suppression described in [9]. Their system introduces a compressor-decompressor configuration designed to reduce traffic over a congested network hop.

Other researchers have focused on methods of caching fragments of web pages rather than entire pages. An early work in this area comes from Douglis, Haro and Rabinovich in [10]. That approach requires that web page authors identify cacheable page fragments at the time of content creation. A more automated approach is introduced by Ramaswamy, Iyengar, Liu and Douglis in [11]. Other fragment-level caching techniques eliminate the need for origin servers to participate in the fragmentation process. Digests, using algorithms such as MD5 or SHA-1 [12], are normally used to fingerprint and index fragments based on their string value. Such fingerprints are applicable to web caching, as shown by Rhea, Liang and Brewer in [13]. Their value-based web caching (VBWC) system involves a parent proxy cache that communicates with a child proxy cache over a slow network connection. The parent proxy divides pages into blocks averaging 2KB in length, indexing each block by its MD5 fingerprint. Another fragment-based approach is the "cache-based compaction" technique proposed by Chan and Woo in [14].

4. TLWC Scheme

TLWC uses a pair of separate, synchronized caches. The server-side cache can be maintained either at an origin server or by a proxy located anywhere between the server and the client-side proxy. The client-side cache can be located at the client or anywhere between the client and the server-side cache. For our prototype implementation of

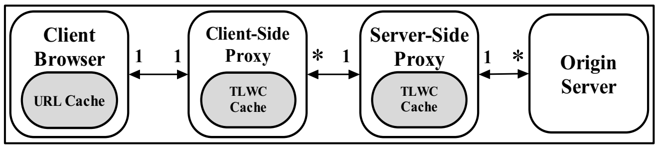


Figure 1. System model

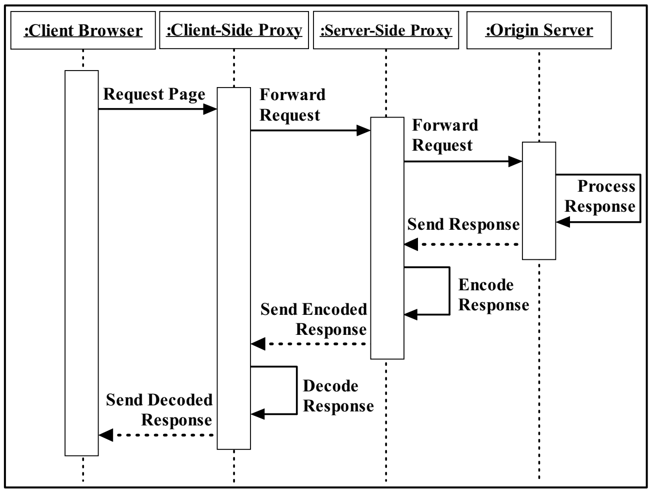


Figure 2. Basic sequence diagram

TLWC, we adopt an architecture where the client-side proxy is located on the same host as the client web browser and the server-side proxy is located at the other end of a slow network link. Another design choice was whether to support single or multiple clients. We choose the more challenging proposition of supporting multiple clients. The system model overview is shown in Figure 1. The basic sequence of events is shown in Figure 2.

This dual proxy architecture is transparent to origin servers. The only special requirement of the client browser is that it assigns the client-side proxy as its HTTP proxy. Note that TLWC works only for unencrypted markup text messages over HTTP. HTTP responses containing plain text, image and other media files are routed through the proxy chain, but are not encoded in any way.

4.1 Encoding approach

The TLWC server-side proxy checks the content-type of every HTTP response it receives from origin servers. If the content is either HTML or XML, the message body is TLWC-encoded before it is forwarded to the client-side proxy. The encoder parses the message body from beginning to end, detecting each markup element. The process is similar to the one used by HTML and XML parsers to form a document object model (DOM). However, the TLWC notion of a node is simpler than that used by parsers that must examine the semantics of a document. TLWC is interested in a node only as a string, so a TLWC node is simply a sequence of characters beginning with the left angle bracket ("<") of an initial tag and ending with the right angle bracket (">") of a

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
    4.0 Transitional//EN">
<html>
  <head>
    <title>Example 1</title>
  </head>
  <body bgColor="#2f2210">
    <h1>Example 1</h1>
    <img height=128 src="images/ex1.jpg">
  </body>
</html>
```

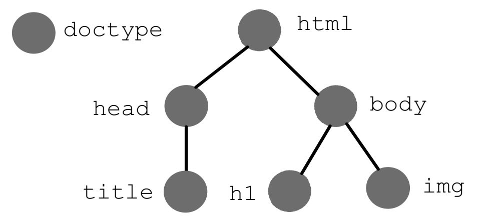


Figure 3. An HTML document and its TLWC parse tree

```
<0><!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
   4.0 Transitional//EN"></>
<6><html>
  <2><head>
    <1><title>Example 1</1>
  </2>
  <5><body bgColor="#2f2210">
    <3><h1>Example 1</3>
    <4><img height=128 src="images/ex1.jpg"></>
  </5>
</6>
```

Figure 4. The HTML document after TLWC encoding

```
<100/>
<3><html>
  <102/>
  <1><body bgColor="#2f2210">
    <0><h1>Example One</0>
    <104/>
  </1>
</3>
```

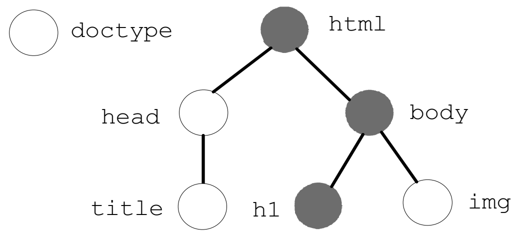


Figure 5. The encoding of the modified response

corresponding closing tag. If the initial tag is of a type that does not require a separate closing tag, the character sequence for the node ends with the right angle bracket of the initial tag. Every node contains at least one tag. As far as TLWC is concerned, every tag is of one of three types: start tag, end tag or self-closing tag.

Note that in TLWC, the identity of a node is determined by the string value rather than by semantics. Therefore `<B>TLWC</B>` is not identical to `<b>TLWC</b>`, nor is `<name></name>` identical to `<name/>`. Also note that TLWC has no notion of node attributes and that any text between a node's start and end tags is part of the immediate node – unlike in traditional HTML and XML parsing, there is no notion of a text node. Figure 3 illustrates a simple HTML document and its TLWC parse tree.

TLWC works most efficiently with markup documents that are "well formed" as defined in the XML specification, but this is not a strict requirement. In fact, only two of the many specified constraints of a well-formed XML document are relevant to TLWC. These are 1) every start tag must have a corresponding end tag and 2) nodes must be properly nested. TLWC tolerates documents that do not have these properties, but works more efficiently on documents that do. When nesting errors are encountered during encoding, the node or nodes involved are simply skipped over by the encoder.

4.2 Encoding technique

The TLWC server-side proxy encodes response messages by checking every node in the message body to see if it is already in its cache. Every node that is not in the cache is enclosed in a pair of identifier tags that indicate that this is a new node. If the new node has an end tag, the identifier tags are in the form `<new-id>` and `</new-id>`. Otherwise the identifier tags are in the form `<new-id>` and `</>`. The enclosed node's end tag is removed (if present). The new node string is entered into the cache, while the node and its enclosing id tags are sent to the client as part of the encoded output string.

Every node that is already in the cache is replaced by a reference identifier tag in the form `<ref-id/>`. The ref-id is the identifier that was assigned to this node string when it was first put into the cache. Figure 4 shows these encoding rules applied to the HTML file of Figure 3. In this example we assume an initially empty cache, so all the nodes are recognized as new.

For purposes of illustration, the node counter in this example is initialized at 100. However, in the encoded document the node count appears to begin at 0 rather than 100. These are relative identifiers, calculated as the difference from a base identifier that is added to the HTTP message as a new header. For example, this encoded message would include the HTTP header `tlwc-base-id: 100`. Relative identifiers are used to minimize the amount of bandwidth overhead required by the encoding process. However, in the cache, all nodes are mapped to absolute identifiers. Table 1 shows the contents of the server-side cache after the proxy has encoded and forwarded this response.

Table 1. Contents of server-side cache after first encoding

Key	Value
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">	100
<title>Example 1</title>	101
<head><101/></head>	102
<h1>Example 1</h1>	103
<img height=128 src="images/logo.jpg">	104
<body bgColor="#2f2210"><103/><104/></body>	105
<html><102/><105/></html>	106

When the TLWC client-side proxy receives the encoded response as shown in Figure 4, it decodes the message by reversing the encoding process. Every pair of `<new-id>` and `</new-id>` tags is stripped from the output string, the enclosed node's end tag is re-appended, and the node string is entered into the cache. Every tag in the form `<ref-id/>` is replaced by the associated node string from the cache. The decoded output is identical to the original, while the contents of the client-side cache are now similar to the contents of the server-side cache – except that the identifiers are the keys and the node strings are the values.

We next suppose that the client has re-requested this page, and that the response from the origin server this time is slightly different: the text in the *h1* node has changed from "Example 1" to "Example One". When the server-side proxy receives this response, it finds that some of the nodes are already in its cache. The encoding of this response is shown in Figure 5, which also shows the parse tree. In this tree, the dark gray nodes are those that were not found in the cache, while the white nodes are those that were found.

Table 2. New server-side cache entries after second encoding

Key	Value
<h1>Example One</h1>	107
<body bgColor="#2f2210"><107/><104/></body>	108
<html><102/><108/></html>	109

There are three things worth pointing out here. First, the *title* node is no longer explicitly present in the encoding, but it is implicitly included since it is contained in the *head* node (102). Second, since the *h1* node has changed, all its ancestor nodes have changed as well. Thus the *html* and *body* nodes in this response are assigned new identifiers. Third, new nodes are found in a depth-first manner, so the first new node in this case is the *h1* node, which is assigned a relative identifier of 0. Its parent node, *body*, is relatively identified as 1 and *body*'s parent, *html*, is identified as 2. If there have been no new cache entries since the first version of this page was encoded, the base identifier (provided in a response header) would be 107. Thus the server-side cache now contains the three new entries shown in Table 2.

At the client-side proxy, the decoding of the second response is performed using the rules already described. As on the sever side, the client-side cache is updated with three new entries. Again, these new key-value pairs are the mirror images of the new entries on the server-side cache: the hash keys are identifiers, the values are node strings.

The basic encoding algorithm can be enhanced and fine-tuned by adding two types of adjustable thresholds. The first is a MIN_CACHE_LEN threshold that a node string is compared against; if the node is shorter than the threshold, encoding of the node is aborted. For example, a threshold setting of 14 or greater would prevent the caching and encoding of the node `<b><106/></b>`.

The second type of threshold, MIN_DIGEST_LEN, can be used along with a digest algorithm such as MD5 to limit the storage demands imposed by longer nodes. Nodes of length equal to or greater than this threshold are hashed by the digest algorithm, and the hash value is used in the cache in place of the node string. Node digests can be used only on the server-side cache, since the client-side cache needs to retain the node string's actual value.

4.3 Cache management

There are two issues that must be handled regarding cache management: replacement and synchronization. First, when either the server-side cache or a client-side cache reaches capacity, it must remove old cache entries to make room for new ones. Second, in order to avoid sending references to nodes that are no longer in a client proxy's cache, the server-side proxy must know the current state of all the proxies it is serving. Our general approach to solving both of these problems is to make extensive use of timestamps. Each cache entry is assigned a "last-accessed" timestamp, which is used by a least-recently used algorithm running independently on each proxy to choose replacement victims. Synchronization between the server-side and client-side proxies is achieved via the exchange of timestamp information that is piggybacked onto HTTP messages as special HTTP headers. Space limitations prevent us in this paper from detailing this synchronization protocol, but the key point is that, assuming a reliable communication link, synchronization is achieved between a server-side proxy and multiple client-side proxies without additional RTTs.

Since communication links in practice are not always reliable, TLWC also includes a technique for recovering from synchronization errors once they are detected. When a client-side proxy encounters a reference to a cache entry that it has already removed from its cache, it sends a special request to the server-side proxy to resend the missing data. Only in this infrequent event does TLWC require additional RTTs.

5. Experimental Results

In this section we report performance tests conducted using the prototype TLWC implementation. All TLWC software components were implemented in Java and deployed on a pair of PCs that were connected to a LAN with a high-speed Internet connection. On the client, we configured Firefox to use a local proxy on the port at which the client-side proxy was listening for requests. In turn, the client-side proxy was configured to forward requests to the TLWC server on a pre-configured port. For our test data, we chose the ten web sites that were deemed to be a fair representation of the Web in general. For each web site, we cleared the browser cache before testing, and then downloaded 20 pages at random. For each of the ten sites, we repeated the 20-page test five times, and took the average results for each site.

TLWC is orthogonal to *gzip* and other data compression techniques. We believe combining data compression with TLWC will produce better results than using either technique on its own. However, to manage the scope of our initial prototype, we did not implement support for data compression. In the tests, we disabled decompression by removing the *Accept-Encoding* header from the request.

Figure 10 plots the average bandwidth reduction for the ten tested sites. The accumulated bandwidth savings for all tested sites is shown in columns 2 and 3 of Table 4. The formulas used to calculate the percentage of bandwidth savings are:

$$\text{Markup reduction} = ((p - e) / p) * 100$$
$$\text{Total reduction} = ((p - e) / t) * 100$$

Where p is the number of pre-encoded markup bytes, e is the number of markup bytes after TLWC encoding, and t is the total number of bytes delivered to the client (p plus all non-markup).

To understand the amount of processing overhead required by TLWC, we measured the amount of time used to encode and decode each page. The overall average time needed to encode a page was 0.158 seconds; the average decoding time was 0.828 seconds. We also captured the numbers of cache searches and cache hits for each page, as measured on the server-side cache. The average results are provided in columns 3-5 of Table 4.

For the average 20-page browsing session, TLWC initially adds a small amount of bandwidth overhead (one or two percent) for the first couple pages, but then begins to provide bandwidth savings by the third page. The steep initial curve in savings gradually flattens out. By the 20th page, the curve is nearly flat at 44.8 percent reduction for markup data and 22.4 percent the total bandwidth savings after images and other non-markup files are factored in.

The test results confirm an expectation that the amount of bandwidth savings provided by TLWC during a browsing session is directly determined by the amount of redundant

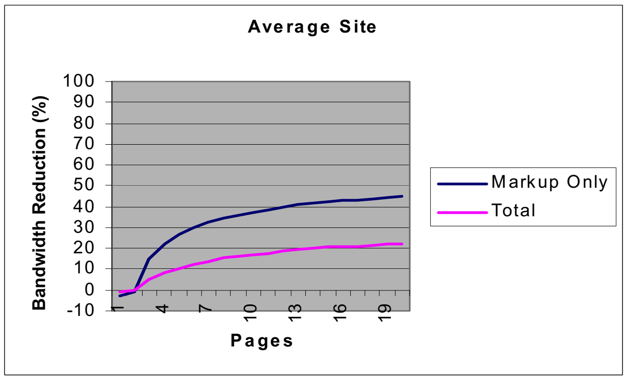


Figure 10. Average bandwidth reduction for tested sites

markup data on the web site and the proportion of markup data compared to non-markup data. Thus, the web sites that most benefit from TLWC are those with large amounts of common page elements and relatively few images.

Including both encoding and decoding, TLWC added less than one second of processing overhead for the average web page. These measurements are a function not only of the efficiency of the algorithms, but also of the hardware and the runtime environment. We are therefore cautious in drawing conclusions from these measurements. However, it is clear that encoding was significantly faster than decoding. We believe this wide difference was due to the fact that the decoding algorithm used was recursive while the encoding algorithm was non-recursive.

Table 4. Average results per 20-page session

	Markup Savings	**Total Savings**	**Cache Searches**	**Cache Hits**	**Hit Ratio**
W3 Schools	63.7%	34.8%	5,686	3,924	0.69
CNN	50.5%	20.4%	13,061	8,655	0.66
Allied Telesis	53.8%	29.5%	4,674	3,189	0.68
Yahoo Shopping	27.5%	20.3%	13,397	6,087	0.45
Smart Webby	59.7%	22.0%	5,860	3,986	0.68
Wikipedia	18.4%	8.6%	8,858	2,563	0.29
Google	15.4%	13.6%	5,706	1,984	0.35
Netgear	58.5%	25.5%	7,567	5,174	0.68
CSUS Class Schedule	58.7%	36.9%	14,411	10,385	0.72
Sun Micro	41.8%	12.1%	7,766	4,894	0.63
Average	**44.8%**	**22.4%**	**8,699**	**5,084**	**0.58**

6. Comparisons

At a system level, TLWC resembles several other approaches to the problem of redundant data transfers. The “dual-proxy” architecture is used by Santos and Weatherall [9], Chan and Woo [14], and VBWC [13]. One criticism of this architecture is that, in most implementations, the server-side proxy serves only a single client. Of these dual-proxy approaches, it appears from our readings that TLWC is the only one that has been developed to support multiple clients with a single server-side proxy.

A closely related criticism of the dual-proxy architecture is that it lacks the simplicity and robustness of stateless caching protocols (i.e., those involving a single cache). In a dual-proxy system, the server needs to know the state of the client. This arrangement is more prone to suffer from synchronization problems when network failures and other irregularities occur. However, we concur with the VBWC researchers that a stateful protocol is justified when it provides performance benefits and when it provides a mechanism to recover from synchronization errors [13]. Both TLWC and VBWC utilize a buffer on the server-side proxy to resend data when these errors occur.

Duplicate elimination techniques have varying levels of granularity. Many techniques work at the page level, which has the shortcoming of treating two pages with even the smallest of differences as being completely unalike. In contrast, fragment-level techniques, such as VBWC and TLWC, detect and eliminate duplicate fragments of pages. A key differentiator among fragment-level schemes is the method used to select fragments. In the technique described in [10], the fragments are selected by page authors and identified by special instructions that are embedded into the page. In the technique described in [11], algorithms running on the web server automate this process. In contrast, in both VBWC and TLWC, the task of selecting fragments is performed by the server-side proxy, and is thus independent of the origin server. However, VBWC and TLWC perform this task in very different ways. In VBWC, the fragments are blocks that are, on average, 2KB in length. In TLWC, fragment boundaries depend on the markup syntax of the content. Furthermore, in TLWC, child fragments are nested within parent fragments, and encoded parents represent their children implicitly. Thus, the level of granularity is dynamically determined by the amount of redundancy.

Finally, TLWC can be viewed as a form of delta encoding, since, when two similar pages are transferred, the second page is expressed as a difference of the first. But TLWC, like cache-based compaction [14], varies from conventional delta encoding because it expresses a page as a difference to a collection of reference pages, rather than just one. And, unlike the delta encoding technique described in RFC 3229 [5], TLWC specifies how reference objects for delta encoding are maintained and identified.

7. Conclusion & Future Work

This report has presented Tag-Level Web Caching, a new technique for reducing redundant transfers of markup data over networks. We have developed a prototype system that applies TLWC to forward proxy-based web caching. Tests of this prototype show TLWC can often reduce the amount of markup data transferred during a browsing session by over 50 percent.

Our prototype implementation of TLWC achieved its goals but, by necessity, left much to future work. Further development of TLWC should include the following:

Algorithm optimization. Both the TLWC encoding and decoding algorithms are first-generation efforts that can be improved substantially with further work.

Response-timing tests. We believe TLWC can improve user-perceived response times during typical web browsing sessions, but response timing tests are left for future work.

Explore other applications. TLWC is potentially applicable to other uses. Reverse proxy, XML acceleration, and wireless network acceleration are among the potential applications that may be worth investigating.

8. References

[1] Wessel, D. *Web Caching*, First Edition. O'Reilly & Associates. 2001. pp 22 - 36.

[2] Kelly, T. and Mogul, J. Aliasing on the World Wide Web: Prevalence and performance implications. Proc. WWW 2002. Honolulu, May 2002.

[3] Deutsch, P. GZIP file format spec. ver. 4.3. RFC 1952.

[4] Deutsch, P. DEFLATE compressed data format spec. ver. 1.3. RFC 1951.

[5] Mogul, J. C., Krishnamurthy B., Douglis, F., Feldman, A., Goland, Y., van Hoff, A., and Hellerstein, D. Delta encoding in HTTP, January 2002. RFC 3229.

[6] Mogul, J. C. A trace-based analysis of duplicate suppression in HTTP. Technical Report 99/2, Compaq Western Research Laboratory, Nov. 1999.

[7] Rivest, R. The MD5 message-digest algorithm. April 1992. RFC 1321.

[8] Mogul, J. C., Chan, Y. M., and Kelly, T. Design, implementation, and evaluation of duplicate transfer detection in HTTP. HP Tech Report HPL-2004-29.

[9] Santos, J. and Wetherall, D. Increasing effective link bandwidth by suppressing replicated data. Proc. USENIX Technical Conference, June 1998.

[10] Douglis, F., Haro, A., and Rabinovich, M. HPP: HTML macropreprocessing to support dynamic document caching. Proc USENIX Symposium on Internet Technology, Monterey, Calif., Dec. 1997.

[11] Ramaswamy, L., Iyengar, A., Liu, L. and Douglis, F. Automatic detection of fragments in dynamically generated web pages. Proc. WWW 2004. New York, May 2004.

[12] Federal Information Processing Standards. Secure hash standard. FIPS PUB 180-1, April 1995, http://www.itl.nist.gov/fipspubs/fip180-1.htm.

[13] Rhea, S. C., Liang, K. and Brewer, E. Value-based web caching. Proc. WWW 2003. Budapest, May 2003.

[14] Chan, M. C., Woo, T. Cache-based compaction: A new technique for optimizing web transfer. Proc. IEEE Infocom '99 Conference, New York, NY, March 1999.

Methodology for Reusing Human Resources Management Standards

Asunción Gómez-Pérez
Facultad de Informática,
Universidad Politécnica de Madrid
Boadilla del Monte, Madrid, Spain
asun@fi.upm.es

Jaime Ramírez
Facultad de Informática,
Universidad Politécnica de Madrid
Boadilla del Monte, Madrid, Spain
jramirez@fi.upm.es

Boris Villazón-Terrazas
Facultad de Informática,
Universidad Politécnica de Madrid
Boadilla del Monte, Madrid, Spain
bvillazon@fi.upm.es

Abstract

Employment Services (ESs), Public ones (PESs) and Private ones (PrEAs), are becoming more and more important for Public Administrations where their social implications on sustainability, workforce mobility and equal opportunities play a fundamental strategic importance for any central or local Government. The EU SEEMP (Single European Employment Market-Place) project aims at improving facilitate workers mobility in Europe. Ontologies are used to model descriptions of job offers and curricula; and for facilitating the process of exchanging job offer data and CV data between ES. In this paper we present the methodological approach we followed for reusing existing human resources management standards like NACE, ISCO-88 (COM) and FOET, among others, in the SEEMP project, in order to build a common "language" called Reference Ontology.

Keywords

Human Resources Management Standard, Human Resources Ontologies.

1. INTRODUCTION

Nowadays there is an important amount of investment in human capital for economic development. Human resources management refers to the effective use of human resources in order to enhance organizational performance [1]. The human resources management function consists in tracking innumerable data points of each employee, from personal records (data, skills, capabilities) and experiences to payroll records [1]. Human resources management has discovered the Web as an effective communication channel. Although most businesses rely on recruiting channels such as newspaper advertisements, online job exchange services, trade fairs, co-worker recommendations and human resources advisors, online personnel marketing is increasingly used with cost cutting results and efficacy.

Employment Services are becoming more and more important for Public Administrations where their social implications on sustainability, workforce mobility and equal opportunities play a fundamental, strategic importance for any central or local Government. The goal of the SEEMP[1] (Single European Employment Market-Place) project is to design and implement an interoperability architecture for e-Employment services which encompasses cross-governmental business and decisional processes, interoperability and reconciliation of local professional profiles and taxonomies, semantically enabled web services for distributed knowledge access and sharing. For this purpose, the resultant architecture will consist of: a Reference Ontology, the core component of the system, that acts as a common "language" in the form of a set of controlled vocabularies to describe the details of a job posting or a CV (Curriculum Vitae); a set of local ontologies, so that each ES (E-Employment Services) uses its own local ontology, which describes the employment market in its own terms; a set of mappings between each local ontology and the Reference Ontology; and a set of mappings between the ES schema sources and the local ontologies [3].

Studer et al. [7] defines an ontology as follows: "An ontology is a formal, explicit specification of a shared conceptualization. Conceptualization refers to an abstract model of some phenomenon. Explicit means that the type of concepts used, and the constraints on their use are explicitly defined. Formal refers to the fact that the ontology should be machine-readable. Shared reflects the notion that an ontology captures consensual knowledge, that is, it is not private of some individual, but accepted by a group".

A major bottleneck towards e-Employment applications of Semantic Web technology and machine reasoning is the lack of industry-strength ontologies that go beyond academic prototypes. The design of such ontologies from scratch in a textbook-style ontology engineering process is in many cases unattractive for two reasons. First, it would require significant effort. Second, because the resulting ontologies could not build on top of existing community commitment. Since there are several human resources management standards, our goal is not to design human resources ontologies from scratch, but to reuse the most appropriate ones for e-Employment services developed on the framework of the SEEMP project. In this paper we present the methodological approach we followed for reusing exist-

[1] http://www.seemp.org/

ing human resources management standards like NACE[2], ISCO-88 (COM)[2] and FOET[2] , among others.

This paper is organized as follows: Section 2 depicts the adopted methodological approach to build the SEEMP Reference Ontology from standards and already existing ontologies. Section 3 describes the resultant SEEMP Reference Ontology. Then section 4 describes some considerations with respect to the building process of the local ontologies. Then section 5 depicts the related work. Finally, section 6 offers some final conclusions.

2. METHODOLOGY FOR REUSING HUMAN RESOURCES MANAGEMENT STANDARDS

In this section we describe the adopted approach to build the SEEMP Reference Ontology. This methodological approach follows and extends some of the identified tasks of the ontology development methodology METHONTOLOGY [4]. This methodological approach consists of:

- Specifying, using competency questions, the necessities that the ontology has to satisfy in the new application.
- Selecting the standards and existing ontologies that cover most of the identified necessities.
- Semantic enrichment of the chosen standard.
- Evaluating the Ontology content.
- Integrating the resultant ontology in the SEEMP platform.

2.1 Specifying, using competency questions, the necessities that the ontology has to satisfy in the new application.

This activity states why the ontology is being built, what its intended users are, and who the end-users are. For specifying the ontology requirements we used the competency questions techniques proposed in [5].

- Intended uses of the ontology. The purpose of building the Reference Ontology is to provide a consensual knowledge model of the employment domain that could be used by ESs, more specifically within the ICT (Information and Communication Technology) domain.
- Intended users of the ontology. We have identified the following intended users of the ontology: candidates, employers, public or private employment search service, national and local governments; and European commission and the governments of EU countries.
- Competency Questions. These questions and their answers are both used to extract the main concepts and their properties, relations and formal axioms of the ontology. We have identified sixty competency questions; they are described in detail in subsection 7.1.3 of the SEEMP deliverable D32 "Supporting the State of the Art". An example of the competency questions is: *Given the personal information (name, nationality, birth date, contact information) and the objectives (desired contract type, desired job, desired working conditions, desired salary) of the job seeker, what job offers are the most appropriate?*
- Terminology. From the competency questions, we extracted the terminology that will be formally represented in the ontology by means of concepts, attributes and relations. We have identified the terms (also known as predicates) and the objects in the universe of discourse (instances); they are described in detail in subsection 7.1.4 of the SEEMP deliverable D32 "Supporting the State of the Art".

2.2 Selecting the standards and existing ontologies that cover most of the identified necessities.

In order to choose the most suitable human resources management standards for modeling CVs and job offers, the following aspects have been considered: The degree of coverage of the objects identified in the previous task, this aspect has been evaluated taking into account the scope and size of the standard. However, a too wide coverage may move us further away the European reality, therefore we have tried to find a tradeoff between this aspect and the following one: the current european needs, it is important that standard focuses on the current European reality, because the user partners involved in SEEMP are European, and the outcoming prototype will be validated in European scenarios; and the user partners recommendations, in order to asses the quality of the standards, the opinion of the user partners is crucial since they have a deep knowledge of the employment market.

Besides, when choosing the standards, we also took into account that the user partners of SEEMP selected the ICT domain for the prototype to be developed in SEEMP. Hence, the chosen standards should cover the ICT domain with an acceptable degree. The standards that finally were chosen are outlined in section 3.1. In the case of the occupation taxonomy, as it will be shown, we have chosen one standard, but then we have taken some concepts coming from other classifications, in order to obtain a richer classification for the ICT domain.

When specifying job offers and CVs, it is also necessary to refer to general purpose international codes such as country codes, currency codes, etc. For this aim, the chosen codes have been the ISO codes, enriched in some cases with user partners classification.

Finally, the representation of job offers and CVs also require temporal concepts such as interval or instant. So, in

[2] Available through RAMON Eurostat's Classifications Server at http://ec.europa.eu/comm/eurostat/ramon/

order to represent these concepts in the final Reference Ontology, the DAML time ontology[3] was chosen.

2.3 Semantic enrichment of the chosen standard.

This activity states how we enrich the human resources management standards, the time ontology, the currency classification, the geographic location classification and language classification. We have followed the process of:

- verifying all concept taxonomies;
- establishing ad hoc relationships among concepts of different taxonomies;
- specifying concept attributes for describing concept features needed;
- defining formal axioms.

2.4 Evaluating the Ontology content.

The evaluation activity makes a technical judgment of the ontology, of its associated software environments, and of the documentation. We will evaluate the Reference Ontology using the competency questions identified in the first task.

2.5 Integrating the resultant ontology in the SEEMP platform

UPM[4] and LFUI[5] will work together in this task in order to integrate the resultant ontology into WSML language at design time, so that, the SEEMP Platform will be able to deal with this ontology at run time.

3. SEEMP REFERENCE ONTOLOGY

The Reference Ontology described in this section will act as a common "language" in the form of a set of controlled vocabularies to describe the details of a job posting and the CV of a job seeker. The Reference Ontology was developed following the process described in detail in section 2 and with the ontology engineering tool WebODE [4]. The Reference Ontology is composed of thirteen modular ontologies: *Competence*, *Compensation, Driving License, Economic Activity, Education, Geography, Job Offer, Job Seeker, Labour Regulatory, Language, Occupation, Skill and Time*. Figure 1 presents:

- These thirteen modular ontologies (each ontology is represented by a triangle). Ten of them were obtained after wrapping the original format of the standard/classification, using ad hoc translator or wrapper for each standard/classification.
- The connections between the ontologies by means of ad hoc relationships. Such relationships will be defined (identifying its domain and range) between specific concepts inside these ontologies later on.

3.1 Wrapping human resources management standards

As it was mentioned before, these ontologies have been developed following existing human resources management standards and systems classifications, and they are:

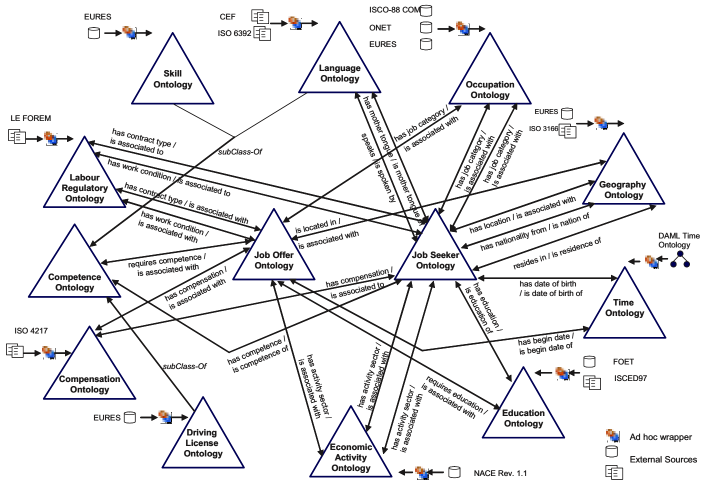

EURES
CEF
ISO 6392
ISCO-88 COM
ONET
EURES
Skill Ontology
Language Ontology
Occupation Ontology
LE FOREM
EURES
ISO 3166
has contract type / is associated to
subClass-Of
has mother tongue / is mother tongue of
speaks / is spoken by
has job category / is associated with
has job category / is associated with
has job category / is associated with
has work condition / is associated to
Labour Regulatory Ontology
has contract type / is associated with
Geography Ontology
has work condition / is associated with
is located in / is associated with
has location / is associated with
has nationality from / is nation of
DAML Time Ontology
requires competence / is associated with
Job Offer Ontology
Job Seeker Ontology
resides in / is residence of
Competence Ontology
has compensation / is associated with
has compensation / is associated to
has date of birth / is date of birth of
has education / is education of
Time Ontology
ISO 4217
has begin date / is begin date of
has competence / is competence of
subClass-Of
FOET
ISCED97
Compensation Ontology
has activity sector / is associated with
has activity sector / is associated with
requires education / is associated with
has activity sector / is associated with
Education Ontology
EURES
Driving License Ontology
Economic Activity Ontology
Ad hoc wrapper
External Sources
NACE Rev. 1.1

Figure 1. Main ad-hoc relationships between the modular ontologies

- *Compensation Ontology* which is based on the ISO 4217[6] . The ISO 4217 is expressed in HTML format. It is a list of 254 currency names and codes. The resultant Compensation Ontology has 2 concepts: `Currency` and `Salary`. For every currency element specified in the ISO 4217 a different instance of the `Currency` concept is defined. So, the `Currency` concept has 254 instances. An example of instance of the `Currency` concept is `UNITED STATES - US Dollar`.
- *Driving License Ontology* which is based on the levels recognized by the European Legislation[7]. This classification is expressed in HTML format and it is a list of 12 kinds of driving licenses. The resultant Driving License Ontology just has the `Driving License` concept; and for every kind of driving license specified in the European Legislation a different instance of the `Driving License` concept is defined. An example of instance of the `Driving License` concept is `A1 - Light weight motorcycle`.
- *Economic Activity Ontology* is based on the NACE Rev. 1.1[8]. This standard is expressed in MS Access da-

[3] http://cs.yale.edu/homes/dvm/daml/time-page.html

[4] Universidad Politécnica de Madrid is a technical partner of the SEEMP project, http://www.oeg-upm.net/

[5] Leopold-Franzens University Innsbruck is a technical partner of the SEEMP project, http://www.deri.at/

[6] http://www.iso.org/iso/en/prods-services/popstds/currencycodeslist.html

[7] http://ec.europa.eu/transport/home/drivinglicence/

[8] Available through RAMON Eurostat's Classifications Server at http://ec.europa.eu/comm/eurostat/ramon/

tabase format and it is a classification of 849 economic activities. The resultant Economic Activity Ontology has 849 concepts. In this case we have defined a concept for every element of the NACE taxonomy in order to preserve the hierarchy.

- *Occupation Ontology* is based on the ISCO-88 (COM)[9], ONET[10] and European Dynamics[11] classification of occupations. ISCO-88 (COM) and ONET are expressed in MS Access database format; European Dynamics classification of occupations is stored in an ORACLE database table. ISCO-88 (COM) is a classification of 520 occupations; ONET is a classification of 1167 occupations and the European Dynamics classification has 84 occupations. The resultant Occupation Ontology has 609 concepts. For this ontology we have extended manually the ISCO-88 (COM) classification with European Dynamics and ONET classifications for ICT occupations. In this case we have defined a concept for every element of the resulting extended taxonomy in order to preserve the hierarchy.
- *Education Ontology*, the education fields are based on the FOET[9] and the education levels are based on the ISCED97[9]; both of them are expressed in MS Access database format. FOET has 127 education fields and ISCED97 has 7 education levels. The resultant Education Ontology has 130 concepts. For the education levels we have defined the `Education Level` concept; and for every education level specified in ISCED97 a different instance of the `Education Level` concept is defined. For the education fields we have defined a concept for every element of the FOET taxonomy in order to preserve the hierarchy.
- *Geography Ontology* is based on the ISO 3166[12] country codes and the European Dynamics classifications: Continent and Region. The ISO 3166 is expressed in XML format; Continent and Region classifications are stored in ORACLE database tables. The ISO 3166 has 244 country codes and names; Region classification has 367 regions and Continent classification has 9 continents. The resultant Geography Ontology has four concepts, a `Location` as main concept, which is split into three subclasses: `Continent`, `Region` and `Country`. For every country element specified in the ISO 3166 a different instance of the `Country` concept is defined, so the `Country` concept has 244 instances. For every region element specified in the Region classification a different instance of the `Region` concept is defined, so the `Region` concept has 367 regions. Finally for every continent element specified in the Continent classification a different instance of the `Continent` concept is defined. An example of instance of the `Continent` concept is `EU - Europe`. An example of instance of the `Country` concept is `SPAIN - ES`. An example of instance of the `Region` concept is `Galicia`.
- *Labour Regulatory Ontology* is based on the LE FOREM[13] classifications ContracTypes and WorkRuleTypes, both of them expressed in XML format. ContractTypes classification has ten contract types and WorkRuleTypes has 9 work rule types. The resultant Labour Regulatory Ontology has 2 concepts. For every type of work condition or contract type considered by LE FOREM, a different instance of one of these two concepts (`Contract Type` or `Work Condition`) is included in the ontology. An example of instance of the `Contract Type` concept is `Autonomous`. An example of instance of the `Work Condition` concept is `Partial time`.
- *Language Ontology* is based on the ISO 6392[14] and the Common European Framework of Reference (CEF)[15]. The ISO 6392 is expressed in HTML format and CEF is a description in PDF format. The ISO 6392 has 490 language codes and CEF has 6 language levels. The resultant Language Ontology has 3 concepts: `Language`, `Language Level` and `Language Proficiency`. For every language element specified in the ISO 6392 a different instance of the `Language` concept is defined, so the `Language` concept has 490 instances. For every language level element specified in the CEF a different instance of the `Language Level` concept is defined, so the `Language Level` concept has 6 instances. An example of instance of the `Language` concept is `eng - English`. An example of instance of the `Language Level` concept is `A2 - Basic User`.
- *Skill Ontology* is based on European Dynamics Skill classification. This classification has 291 skills and it is stored in an ORACLE database table. The resultant Skill Ontology has 2 concepts: `Skill` concept with its subclass `ICT Skill`. For every skill element specified in the European Dynamic classification a different instance of the `ICT Skill` concept is defined. An example of instance of the `ICT Skill` concept is `Hardware programming`.
- *Competence Ontology* defines a concept called Competence as a superclass of the imported concepts

[9] Available through RAMON Eurostat's Classifications Server at http://ec.europa.eu/comm/eurostat/ramon/

[10] http://online.onetcenter.org/

[11] http://www.eurodyn.com/

[12] http://www.iso.org/iso/en/prods-services/iso3166ma/index.html

[13] LE FOREM is an user partner of the SEEMP project, http://www.leforem.be/

[14] http://www.iso.org/iso/en/prods-services/popstds/languagecodes.html

[15] http://www.cambridgeesol.org/exams/cef.htm

`Skill`, `Language Proficiency` and `Driving License`.

- *Time Ontology* is based on DAML ontology[16] and it is expressed in OWL format.

In order to make possible the enrichment of the standards, it was necessary to import them into the ontology engineering tool WebODE [4]. This process consisted of implementing the necessary conversions mechanisms for transforming the standards into WebODE's knowledge model. For this purpose we have developed for each standard/classification an ad hoc translator (wrapper) that transformed all the data stored in external resources into WebODE's knowledge model.

3.2 Enriching the ontologies

Once we transformed the standards into ontologies, the next step is to enrich them introducing concept attributes and ad hoc relationships between ontology concepts of the same or different taxonomies. We perform this task, doing the following.

- We created from scratch the Job Seeker Ontology, which models the job seeker and his/her CV information.
- We created from scratch the Job Offer Ontology, which models the job vacancy, job offer and employer information.
- We defined relationships between the concepts of the Job Seeker Ontology and the concepts defined on the standard (classification) based ontology.
- We defined relationships between the concepts of the Job Offer Ontology and the concepts defined on the standard (classification) based ontology.

Finally we present the Reference Ontology statistics. The Reference Ontology is composed of twelve modular ontologies. The Reference Ontology has 1609 concepts, 6727 class attributes, 60 instance attributes, 94 ad hoc relationships and 1674 instances.

4. LOCAL ONTOLOGIES BUILDING PROCESS

As it was mentioned before, the other components of the resultant SEEMP architecture will be: a set of local ontologies, so that each ES (E-Employment Services) uses its own local ontology, which describes the employment market in its own terms; a set of bidirectional mappings between each local ontology and the Reference Ontology; and a set of bidirectional mappings between the ES schema sources and the local ontologies.

In this section we provide some guidelines for the building process of the local ontologies. Based on the proposed SEEMP architecture, the possible options for building the local ontologies are:

- Option 1: Building local ontologies from the Reference Ontology.
- Option 2: Building local ontologies as a reverse engineering process from ES schema sources.

4.1 Building local ontologies from the Reference Ontology

In this case, we will probably need a specialization of the Reference Ontology and also an extension; by specialization we mean extending in depth the concepts we already have in the Reference Ontology; by extension we mean including application dependent concepts that appear in each ES schema source. Also mappings between local ontologies and Reference Ontology will not be complex. But on the other hand, mappings between local ontologies and ES schema sources will be complex. The building process is structured/guided by the architecture of the Reference Ontology and scoped with applications needs. The result should be a Reference Ontology friendly "local" ontology. If the customer needs data exchanges, he has to accept the exchange protocol with some readiness. This is an opportunity to impose an 'ontological order' on various users and systems. Regarding the evolution and change propagation dimension we have:

- Changes in the Reference Ontology imply a change in the mappings between local and global ontologies as well as probably changes in the mappings between the local ontologies and the ES schema sources.
- Changes in the Reference Ontology imply a change in the local ontology; in this case, the mappings Reference Ontology – local ontology would remain as they were. The mappings between the local ontologies and the ES schema sources should be updated.
- Changes in the ES schema sources imply changes in its local ontology (probably the part that is not a mirror of the Reference Ontology) and the mappings between local ontologies and ES schema sources, and probably minor changes in the mappings between local ontology and the Reference Ontology.

4.2 Building local ontologies as a reverse engineering process from ES schema sources

In this case, mappings between local ontologies and schema resources should not be complex. On the other hand, complex mappings will appear between the Local and Reference Ontology. The building process requires more sophistication of knowledge engineering and good acquaintance of all the data and their structures of the application: not easily found skill set in ES or any other operational/research organizations. Regarding the evolution and change propagation dimension we have:

- Changes in the ES schema sources imply a change in the local ontologies and, consequently, in mappings between sources and local ontologies as well as mappings between local and the Reference Ontology.

[16] http://cs.yale.edu/homes/dvm/daml/time-page.html

- Changes in the Reference Ontology imply changes in the mappings between local ontologies and the Reference Ontology, but it is not necessary to modify anything at the ES level.

4.3 Approach followed by SEEMP

In SEEMP project we follow a hybrid approach. On one hand, we select option 1 (building local ontologies from the Reference Ontology) for Job Seeker and Job Offer ontologies and other general purpose ontologies like, for example, the Time Ontology. On the other hand, we select option 2 (building local ontologies as a reverse engineering process from ES schema resources) for Occupation, Education, Economic Activity, Language, Compensation, Labour Regulatory, Skill and Driving License ontologies.

The reason of selecting option 1 for Job Seeker and Job Offer ontologies is because there are not significant differences between these ontologies and the way how each ES structures job seeker and job offer information. Consequently mappings between local ontologies and Reference Ontology will be simple, but mappings between local ontologies and ES schema sources will be complex. For the job seeker and job offer information local ontologies will share the same vocabulary (see [8]).

And the reason of selecting option 2 for the ontologies mentioned above is because each ES may have its own classification systems for the related information. It may happen that the local ontology shares some classification with the reference ontology (as there will happen in the European scope with the driving license classification). In that case, the reverse engineering process for that classification will be skipped, and that part of the reference ontology will be reused. By using option 2, mappings between local ontologies and Reference Ontology will be complex, but mappings between local ontologies and ES schema sources will be simple.

5. RELATED WORK

Currently the Human Resource Semantic Web applications are still in an experimental phase, but their potential impact over social, economical and political issues is extremely significant. [2] presents a scenario for supporting recruitment process with Semantic Web technologies but within German Government. In [6] we can find a brief overview of a Semantic Web application scenario in the HR sector by means of describing the process of ontology development, but its final goal is to merge ontologies.

6. CONCLUSIONS

In this paper we have presented the methodological approach we followed for reusing existing human resources management standards in the SEEMP Project. We also described the resultant Reference Ontology which acts as a common "language" in the form of a set of controlled vocabularies to describe the details of a job posting and the CV of a job seeker. The Reference Ontology was developed with the proposed methodology and with the ontology engineering tool WebODE. Finally, we have provided some guidelines for the building process of the local ontologies, and we conclude that the best option for building the local ontologies is building them following a hybrid approach.

An important conclusion of the work that we have carried out is that we can reuse human resource management standards in new applications following a systematic approach. Moreover, it is clear such a reuse can save time during the development of the whole system. However, it is not always possible to reuse a standard in a straightforward way, because sometimes the ideal standard does not exist for different reasons (different scope, outdated, etc.), and it is necessary to extend some "imperfect" standard with additional terminology coming from other standards or ad hoc classifications.

ACKNOWLEDGMENTS

This work has been partially supported by the FP6 EU SEEMP Project (FP6-027347).

REFERENCES

[1] Legge, K.: Human Resource Management: Rhetorics and Realities. Anniversary ed. Macmillan. (2005).

[2] Bizer C., Heese R., Mochol M., Oldakowski R., Tolksdorf R., Eckstein R.:"The Impact of Semantic Web Technologies on Job Recruitment Processes"; 7th Interna-tional Conference Wirtschaftsinformatik (2005)

[3] FOREM, UniMiB, Cefriel, ARL, SOC, MAR, PEP: User Requirement Definition D.1.SEEMP Deliverable (2006).

[4] Gómez-Pérez A., Fernández-López M, Corcho O.: Ontological Engineering. Springer Verlag. (2003)

[5] Grüninger M, and Fox MS.: Methodology for the design and evaluation of ontologies In Skuce D (ed) IJCAI95 Workshop on Basic Ontological Issues in Knowledge Sharing, (1995) pp 6.1–6.10

[6] Mochol M., Paslaru Bontas E.: Simperl: "Practical Guidelines for Building Semantic eRecruitment Applications", International Conference on Knowledge Management (iKnow'06), Special Track: Advanced Semantic Technologies (2006)

[7] Studer, R., Benjamins, V.R., Fensel, D.: Knowledge Engineering: Principles and Methods. Data and Knowledge Engineering. 25: (1998) 161-197.

[8] Swartout, W.R., Patil, R., Knight, K., and Russ, T.: Towards Distributed Use of Large-Scale Ontologies, AAAI-97 Spring Symposium on Ontological Engineering, Stanford University, May, (1997)

An Adaptive Resource Management Approach for a Healthcare System

Claudia Raibulet, Luigi Ubezio, Stefano Mussino
Università degli Studi di Milano-Bicocca
DISCo – Dipartimento di Informatica Sistemistica e Comunicazione
Via Bicocca degli Arcimboldi, 8, I-20126, Milan, Italy
{raibulet, ubezio, stefano.mussino}@disco.unimib.it

Abstract: *ARMS (Adaptive Resources Management System) represents our solution to address dynamic adaptivity. It defines appropriate abstractions and mechanisms to capture and manage at runtime non-functional information about the components of a system and the services they provide. The objectives of ARMS are (1) to identify the most suitable system component able to execute the current service request and (2) to ensure that the required qualities of services are guaranteed during its execution. This paper aims to present the issues raised by applying our ARMS architecture to an actual case study regarding a healthcare system. Attention is focused on how to exploit both domain specific knowledge and system knowledge in the adaptation process. Moreover, we describe three scenarios related to this case study and the problems encountered and addressed through ARMS.*

1. Introduction

Adaptivity is one of the challenging issues of today information systems [5, 8, 11, 14]. It is achieved by exploiting at runtime information which is not usually modelled in a software representation of a system. Thus, it claims for appropriate abstractions to represent this information and for efficient mechanisms to implement activities usually performed by humans based on their own knowledge and experience.

The aim of ARMS is to provide an example on how to design flexible systems predisposed to adaptation. Adaptation in ARMS regards various aspects ranging from the selection of the most appropriate system component to execute a service based on aspects behind its functionality (i.e., performance, quality, cost) to the observation of services' execution to check if the non-functional aspects are ensured at runtime and to the management of emergency requests and situations. In the scientific literature adaptation is considered mostly as a solution to critical problems (i.e., fault tolerance, security attacks) [9, 11]. ARMS aims not only to address this type of situations, but also to prevent them.

Our solution for adaptive systems exploits reflection to observe and control various aspects of the underlying system. Reflection introduces meta-representations of the system's components and a mechanism to maintain them up-to-date with the status and the non-functional features of the system's components. This mechanism is used also to apply the changes required by the adaptation process after the observation of the meta-representations. Reflection may introduce several disadvantages (i.e., increase number of objects, modifications at the reflective level may cause overall damage if reflection is not properly exploited), but it provides significant advantages too: separation of concerns, modularity, reusability, and an easier overall maintainability and evolution.

In previous works we have described in detail various aspects of our solution: the design of the reflective entities [15], the service-oriented aspects of ARMS [16], and the mapping between high-level qualities of services and low-level features of the system's components [17]. In this paper we present how ARMS is exploited by an actual system. The case study is related to a healthcare system. Attention is focused on the usage of both domain and system knowledge to achieve adaptivity. We described the addressed problems and the solutions adopted.

The rest of the paper is organized as following. Section 2 presents on overview of ARMS introducing the main concepts it defines for runtime adaptation. Section 3 describes HARMS (Health-ARMS) by focusing on three different scenarios which claim for adaptivity. Discussions and further work are dealt within Section 4.

2. Our Approach

ARMS defines a service oriented based architecture [6] which exploits reflection [4, 10] at the architectural level to achieve adaptivity at runtime. Essentially, reflection defines the *causal connection* mechanism to observe and control the underlying components of a system through appropriate *metadata* [10]. Metadata models non-functional information (i.e., qualities of services (QoS) [1, 13]) of the systems' components which is used at runtime in the adaptation process. This information represents the reflective knowledge. The causal connection mechanism ensures that any modification in a system component is propagated to its corresponding meta-representation, and vice-versa, any modification at the meta level is reflected on the corresponding system component (whenever this is possible) (see Figure 1).

In ARMS, reflective knowledge is modeled through four elements: *reflective objects*, *reflective services*, *QoS*, and *properties*. Reflective objects are the meta-representations of the system objects modeling their current status, while reflective services are meta-representations of the services offered by the system objects. *Low-level* QoS (representing measurable values) model the non-functional features of services which may be used to achieve adaptivity (i.e., bandwidth, cost, provider). Properties represent additional information about the system's components which are not strictly related to the services they provide, but which may be meaningful for adaptation (i.e., location of a user or of a resource).

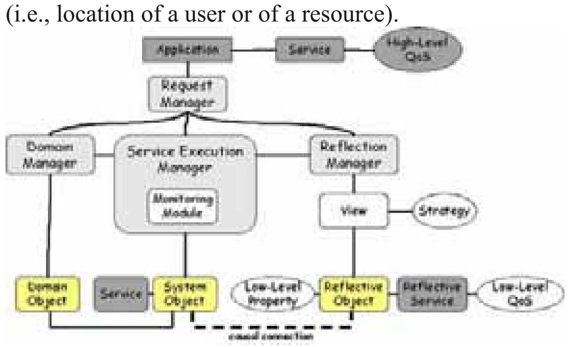


Figure 1. ARMS Main Components

To improve the organization and the management of the reflective knowledge we have introduced two additional elements: *views* and *strategies*. A view is defined as an organizational structure on the reflective objects which has its own semantic and its own computational strategies to evaluate the elements under its control [12].

Strategies implement the logic necessary to take decisions. A view has associated a best-effort strategy which assigns scores to the reflective objects it manages based on its own semantic. The higher the score is, the better suits the object to the current service request. In this way, a view is able to indicate a classification or the most appropriate object able to fulfil a request. For more details see [12].

Note that a reflective object may be used in various views, but it has only one representation in the system. Views contain references to the reflective objects they manage.

When applying ARMS to an actual case study three types of knowledge should be managed: *domain knowledge* (i.e., defining objects specific to the current example), *system knowledge* (i.e., defining objects specific to the underlying physical components of the system), and *reflective knowledge* (i.e., defining objects which influence the performances of the system). Each of the three types of knowledge is manipulated through a manager. Managers can interact among them to exchange information related to the services' requests. As views, also managers have associated strategies. For example, the *reflection manager* has associated a strategy which decides the views to be used to solve each service request.

The *request manager* receives the services' requests from applications and chooses, based on the information available in each request, the types of knowledge to exploit to solve it. Whenever a request claims for adaptivity, the reflection manager is asked to identify the most appropriate resource and/or to set the QoS on the chosen resource as close as possible to the once specified in the request.

At the application level services are characterized by *high-level* QoS, which may be either qualitative (i.e., high resolution) or quantitative (i.e., 1600 x 1200 resolution). Note that high-level QoS include both QoS and properties. For example, a user may claim for a printing service with a *high quality* on the *nearest printer* to her/his current location. These high-level QoS are translated into resolution, colour depth, and printing speed as low-level QoS of a reflective service and into the location property of its corresponding reflective object. These mappings between high-level and low-level QoS and properties are performed by strategies associated to views [12].

The service execution manager has a monitoring module which verifies if the QoS claimed in a request are guaranteed during the execution of the service. Currently, two solutions are considered in case QoS become out of an expected range. The first regards the specification of the first two most appropriate system objects able to execute a service. If the first system object fails to ensure the promised QoS or a failure occurs, the second one is asked to execute the service. In an alternative solution, the service execution manager asks the reflection manager to choose another solution for the current request. In both cases the request manager (and implicitly the application or user) is not aware of the problems occurred.

3. Applying ARMS to a Healthcare System

The HARMS case study models the main information regarding a hospital: patients with their clinical records including the medical tests they have done, the medical tests which can be performed in the hospital and the equipment necessary to do them, employees (i.e., doctors/physicians, nurses), the organizational structure of the hospital (i.e., hospital departments). A simplified class diagram describing these elements is shown in Figure 2.

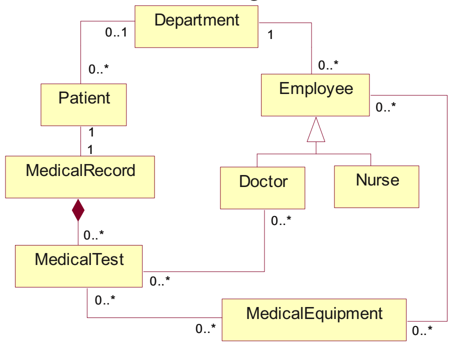


Figure 2. Entities of the Domain Application

In the following we present three scenarios to point out how ARMS can be used in this case study and how adaptation is performed exploiting both domain and system knowledge at runtime.

3.1 First Scenario

This scenario regards the request of information by the patients or by the medical stuff. The result of this request is the visualization of the information in a clinical record of a patient. Adaptation here can involve different aspects.

When a user asks for information (i.e., through a SMS) specifying only her/his current location [2], the system identifies the component which can display information (i.e., wall monitor, PC) and which is the nearest one to the user. The reflective part of the architecture uses the display service view and the location view to solve this request. When a user asks for information by specifying also the QoS the displayed information should be characterized by, the system identifies the nearest component to the user with the required QoS. As previously, the reflective part uses both the display service and the location view to identify the suitable resource. The difference consists in the scores associated by each view to the reflective objects they manage based on the information specified in the request. Hence, the identified resources in the two cases may be different or identical. In both cases the system sends all the information in the clinical record of a patient (specified in the request as InputData – see Figure 4) and notices the user where information is displayed (specified in the request as Resource – see Figure 4).

A version of this scenario may require also the adaptation of the requested information to the characteristics of a user (i.e., preferences, disabilities) and/or to the device claiming for information (i.e., mobile phone, PDA). For example, when using a PDA, a user would like to receive only textual information, or when using a high resolution monitor s/he may receive detailed information including also images. Or, a doctor may want to receive only the information of a patient regarding her/his specialization. In this case, the resource and QoS elements are already specified in the request and only domain specific knowledge is involved in the decision process. The resource and its related QoS are implicitly included in the request (see Figure 4). Adaptation here consists in the extraction of the information based on the user profile or on the QoS of the indicated resource. This has been done by introducing strategies into the domain knowledge as shown in Figure 3.

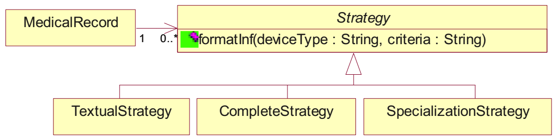


Figure 3. Strategies to Extract Information from a Clinical Record based on Various Criteria

There are also cases in which both domain and system knowledge are required for adaptation. For example, a doctor claims the clinical record of a patient specifying her/his specialization and current location. The system should identify the nearest display device and extract the requested information from the clinical record.

To summarize, adaptation may use:

- only system knowledge to identify the most appropriate component to provide a service;
- only domain specific knowledge to adapt the requested information to the user's preferences and/or to a system component features;
- both domain and system knowledge to identify the most appropriate component to provide the service and in the same time to adapt the requested information to the identified component.

3.2. Second Scenario

This scenario considers that a person arrives in critical conditions in a hospital or the conditions of a recovered patient become critical and s/he needs immediate assistance. Such scenarios and their related requests can be seen as particular, unexpected situations in which the system should react immediately and efficiently by providing the most appropriate solution for the current context. However, they are quite usual in such an application domain.

The obvious solution to this type of problems identifies a doctor with the required preparation or specialty (or with the closest one) available in that moment (present in the hospital), and not implied in a critical activity (a surgery) to treat the case.

In this case, adaptation uses only the location view of the reflective knowledge, while it exploits heavily the domain knowledge. It identifies the doctor heaving the required specialty, the tests which should be performed, the available equipment for doing medical tests, etc. Adaptation can be significantly improved by introducing reflective knowledge also for domain entities, which however is implicitly used in everyday life. For example, there are various medical tests which can be performed to identify a disease, each of them having different duration, response time, and precision. Thus, it is fundamental to choose the one providing the most valuable result for the current situation. Or, there are doctors with the same specialty, but with different experience or treatment methodology. To decide which/who the most appropriate solution is, a system should model and exploit this information at runtime.

To improve the adaptation process, we exploit reflection for the domain knowledge too. We have reused all the reflective entities defined for the system's objects: reflective objects, reflective services, QoS, properties, views and strategies. Hence, the only conceptual modification in Figure 1 is that domain objects have a causal connection with reflective objects similarly to the one between system's objects and reflective objects.

To summarize, adaptation exploits both domain and system knowledge. It is a challenging issue the definition of domain QoS as well as domain strategies, while there is little agreement in what should be exploited at runtime for adaptation and how much influences each reflective entity the adaptation process.

3.3. Third Scenario

The third scenario regards the request of a medical test which should be performed immediately or within a specified period of time. When required immediately, the system verifies who is the available doctor able to perform the test and if required, the available medical equipment. If there are no immediate solutions it checks who/which will finish first the current activity and uses it/s/he as soon as possible.

If the test is not linked to an emergency request, it should be booked in a specified period of time (i.e., after 24 hours and before 48 hours) or within a time limit (i.e., within seven days). The system books the first available resource meeting the time constraint.

This type of scenario raises two issues which are related to each other: the time dimension and the allocation of resources. There are requests in which a service should be provided immediately, or as soon as possible, while there are requests that claim for a service booking. In the first scenario we have implicitly considered that all the system's components are always available. This is not always the real case.

The solution we provide to these problems is to introduce the BookingTime class in a service request to specify a time interval when the service should be executed (see Figure 4). If the supLimit variable is zero, than the service should be immediately executed. If the supLimit is not zero, than the service should be booked. The result of the booking is a *Contract* indicating the resource which will execute the service, the QoS which will be guaranteed, and the expected time when the service will be executed (see Figure 5).

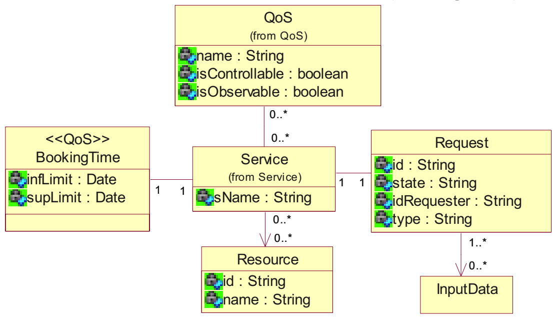


Figure 4. Services' Requests

A contract represents an agreement between the requester of a service and a component of a system which can provide the service. Contracts are checked and supervised by the request manager of ARMS. Each agreement has an expiration time (the expected execution time).

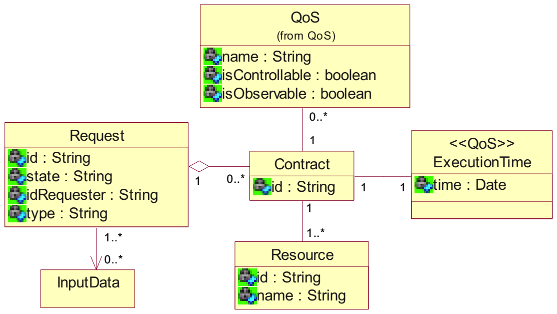


Figure 5. Contract for Booking Services

Note that booking and execution time are considered as QoS of the requested, respectively booked service. Hence, they are addressed as all the other QoS in ARMS. Both represent absolute time.

A service contract is sent to the user too, who may claim its cancellation before its execution. Modifications of contracts are currently implemented through a cancellation and a new service request.

Through the various QoS specific to each service, we have introduced one indicating the average time the component needs to execute that type of service (see Figure 6). Each reflective component (R_Object) has associated a timetable to trace the contracts it has to execute. The timetable is common to all the services (R_Service) provided by a component. It is used to book services and to execute the scheduled contracts.

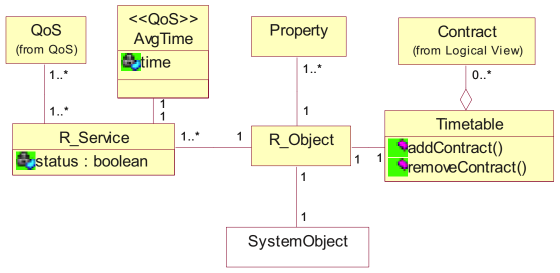


Figure 6. Services, Timetables and Contracts

The request manager maintains a list of the contracts, too. It uses this list to cancel a contract and to execute a contract. When a cancellation request arrives to the Request Manager it verifies if the contract exists, and if it exists it asks the system component which should have provide the service to cancel the contract. The actual execution of a service booked through a contract is requested by the Request Manager at the expected execution time. In this way, a customer/application does not have to re-send the request for the execution of the service. The list of contracts of the Request Manager is implemented as a view on all the timetables in the system.

Figure 7 shows the sequence diagram for the execution of a contract. The Request Manager checks the execution time of each contract. When a contract should be executed, it creates two types of requests: a setting request (if in the contract are specified the QoS which should be guaranteed during the execution of the service) and a service execution request. The setting request is sent to the Reflection Manager which is in charge to manipulate the reflective knowledge. The reflection correspondent of the component which should execute the service checks if the contract in its timetable corresponds to the contract indicated in the request and if the answer is positive than it sets the QoS on the values specified in the contract. Once finished this operation it notifies the system component through the causal connection mechanism that settings are ready and cancels the contract from its timetable.

To summarize, there are requests which claim for an immediate (or as soon as possible) execution of a service, and requests which book a service. Time becomes important because it determines when to execute a service, and how long a resource is used to provide a service.

4. Conclusions and Further Work

In this paper we have described how our solution for adaptivity is applied to an actual case study and which have been the problems encountered and the solutions adopted. Attention was focused on the possibility to exploit additional information about domain knowledge which is not usually modelled in traditional approaches, but which improves adaptivity. Usually, this information is the property of the actors

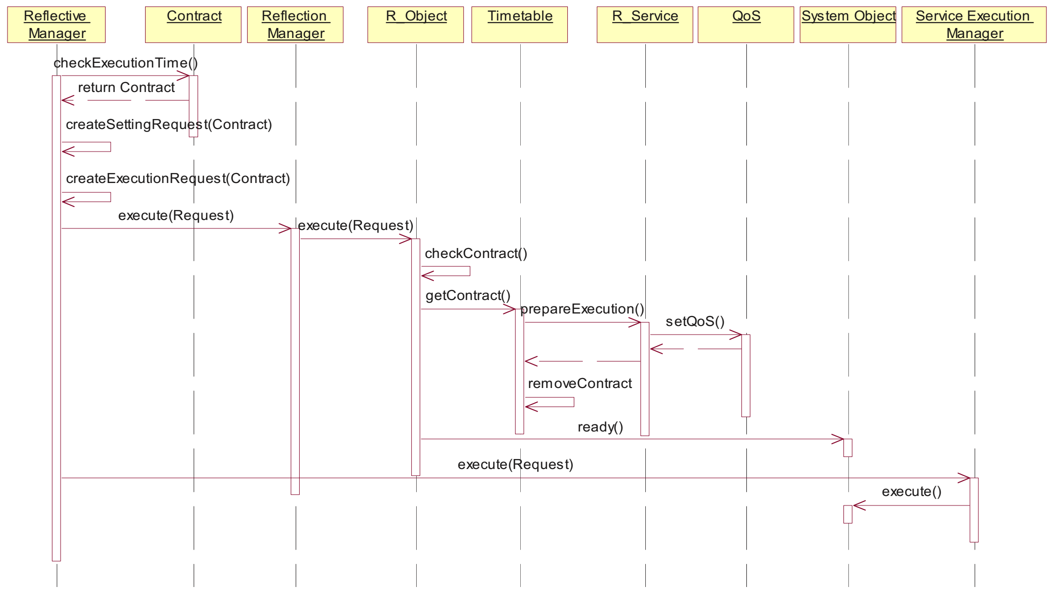


Figure 7. Execution of a Contract

of the system and they use it implicitly when making requests or offering solutions, but there is no explicit representation and usage of it in the system.

Furthermore, we have considered the possibility to book services, not only to execute them. Booking services implies the definition of additional elements to trace the accepted contracts, but in the same time it allows a system to improve the usage of its components by grouping together those services claiming for the same QoS. The evolution of ARMS involved the addition of elements such as booking time, contracts and timetables. It has not lead to the modification of the other elements of our architecture.

The introduction of contracts raises additional issues which regard the check that the solution established through a contract remains the best for a request also at the execution of the contract. This may be addressed by associating timetables to the service views and not to reflective objects. In this case it is supposed that services may be executed in parallel (i.e., two or more different components may execute the same type of service at the same time). The allocation of resources becomes more complicated because the same component may provide two or more different services booked in parallel by different views.

As anticipated by experts, the actual application of HARMS in a hospital may encounter difficulties of various types. For example, there are medical departments where location sensors are not welcome (i.e., cardiology).

Several related works consider the healthcare system as a case study to apply various architectural models to achieve adaptivity [3, 7, 9]. They address similar issues related to the variety of the devices used to access information and to the availability and efficient usage of the resources especially in emergency situations. The main differences with our solution are that we use reflection to achieve adaptivity at runtime, we use domain specific non-functional aspects and we consider also the booking of services not only their immediate execution.

5. References

[1] Aagedal, J.O. *Quality of Service Support in Development of Distributed Systems*. PhD Thesis, University of Oslo, Norway, 2001.

[2] Arcelli, F., Raibulet, C., Tisato, F., Ubezio, L. Designing and Exploiting the Location Concept in a Reflective Architecture. In *Proceedings of the 14th International Conference on Intelligent and Adaptive Systems and Software Engineering)*, 2005

[3] Bardram, J. E. Hospitals of the Future – Ubiquitous Computing support for Medical Work in Hospitals. In *Proceedings of the 2nd International Workshop on Ubiquitous Computing for Pervasive Healthcare Applications*, 2003

[4] Cazzola, W., Sosio, A., Savigni, A., Tisato, F. Architectural Reflection. Realising Software Architectures via Reflective Activities. In *Proceedings of the International Workshop on Engineering Distributed Objects*. LNCS, Springler Verlag, 2000, pp. 102-115.

[5] Cheng, S. W., Garlan, D., Schmerl, B. Architecture-based Self-Adaptation in the Presence of Multiple Objectives. In *Proceedings of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, 2006, pp. 2-8

[6] Erl, T. *Service-Oriented Architecture: Concepts, Technology and Design*, Prentice Hall PTR, USA, 2005.

[7] Favela, J., Rodriguez, M., Preciado, A., Gonzalez, V. M. Integrating Context-Aware Public Displays Into a Mobile Hospital Information System. *IEEE Transactions on Information Technology in Biomedicine*, 8, 3, 2004, pp. 279-286

[8] Kon, F., Costa, F., Blair, G., Champbell, R.H. Adaptive Middleware: The Case for Reflective Middleware. In *Communications of the ACM*, Vol. 45, No. 6, 2002, pp.33-38

[9] Kumar, M., Shirazi, Das, S. K., Sung, B. Y., Levine, D., Singhal, M. PICO: A Middleware Framework for Pervasive Computing. *IEEE Pervasive Computing Mobile and Ubiquitous Systems,* 2, 3, 2003, pp. 72-79

[10] Maes, P. Concepts and experiments in computational reflection. In *Proceedings of the Object-Oriented Programming Systems Languages and Applications (OOPSLA'87)*, 1987, pp. 147-155.

[11] McKinley, P. K., Sadjadi, S. M., Kasten, E. P., Cheng, B. H. C. Composing Adaptive Software. *Computer, IEEE Computer Society,* 37, 7, 2004, pp. 56-64.

[12] Mussino, S. *ARM (Adaptive Resource Management): Design and Development of Adaptive Applications*. MSc Thesis, University of Milano-Bicocca. Italy. 2005.

[13] OMG Adopted Specification. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms. ptc/2004-06-01, http://www. omg.org, 2004.

[14] Poladian, V., Sousa, J.P., Garlan, D., Shaw, M. Dynamic Configuration of Resource-Aware Services. In *Proceedings of the 26th International Conference on Software Engineering,* 2004, UK, pp. 604-613.

[15] Raibulet, C., Arcelli, F., Mussino, S., Riva, M., Tisato, F., Ubezio, L. Components in an Adaptive and QoS-based Architecture. In *Proceedings of the ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, 2006, pp. 65-71

[16] Raibulet, C., Arcelli, F., Mussino, S. Exploiting Reflection to Design and Manage Services for an Adaptive Resource Management System. In *Proceedings of the IEEE International Conference on Service Systems and Service Management*, 2006, 1363-1368

[17] Raibulet, C., Arcelli, F., Mussino, S. Mapping the QoS of Services on the QoS of the Systems' Resources in an Adaptive Resource Management System. In *Proceedings of the 2006 IEEE International Conference on Services Computing*, Work-in-Progress, 2006, pp. 529-530

Study of the Relationships between Personality, Satisfaction and Product Quality in Software Development Teams

Marta Gómez
Escuela Politécnica Superior
Universidad San Pablo-CEU
mgomez.eps@ceu.es

Silvia T. Acuña
Escuela Politécnica Superior
Universidad Autónoma de Madrid
silvia.acunna@uam.es

ABSTRACT
In this article we analyse the relationships between personality, task and group characteristics, product quality and satisfaction in software development teams. The data were collected from a sample of 35 teams of students (105 participants) at a Spanish university. The arithmetic mean is used as an index for aggregating individual team members' scores.
The teams most satisfied with their work are precisely the ones that score higher on the agreeableness and conscientiousness personality factors. Satisfaction levels are also higher when members can decide how to develop and organize their work. On the other hand, the level of satisfaction and cohesion fall the more conflict there is among team members about the task. Finally, there is a significant positive correlation in teams between the extroversion personality factor and the quality of the developed software product.

Keywords: Personality factors; Software quality; Satisfaction; Team building

1. INTRODUCTION

People are a fundamental and critical concern in software development success or failure. There is research that takes this aspect into account and incorporates people into the software process [1][2][3]. These studies analyse people individually and establish relationships to the activities performed as part of the software project. They agree on the fact that these people work together to perform interdependent development tasks, and these group interrelationships are complex. This leads to the need to examine software development team formation.
Social psychology is now researching team formation considering a series of factors that have been found to affect team performance. This research analyses team member personality factors and their relationship to task characteristics [4][5][6]. However, other team behaviour factors, like interactions between people, including conflict, cohesion, cooperation, communication and climate [7], play a role in this relationship.
The results of social psychology studies are task dependent (although there are general trends that appear to affect all tasks equally). Therefore, software development teams need to be examined separately to discover what factors are influential in this particular field and how alike these specialized results are to the outcomes for other tasks.
There are few studies in the field of software development on the impact of some group factors —cohesion, conflict, team structure, coordination, expertise— on team performance [8][9]. Other research analyses the motivational issues that govern the behaviour of developers and their teams [10].
The key question addressed here is whether software product quality and the level of team member satisfaction depend on team member personality. The study we conducted measures the personalities of the members of each team on the basis of the Big Five personality factors [11]: extroversion, neuroticism, agreeableness, conscientiousness and openness to experience. We also measure the characteristics of the task the teams perform. This measurement is confined to interdependency and autonomy, which Molleman et al. [4] established as being the most important factors influencing team behaviour. Finally, we also account for other factors that are important for getting a better understanding of these relationships. They are the group behaviour factors conflict and cohesion.
This article is structured as follows. Section 2 describes the work related to team formation in the field of software engineering. Section 3 details the experimental study method used to relate personality factors to software quality and team satisfaction, as well as the task characteristics and some group processes. Section 4 presents the data analysis and results. Section 5 discusses the results and sets out the conclusions.

2. TEAM FORMATION RELATED WORK

This section analyses studies related to team formation and behaviour in the field of software engineering, highlighting only the factors most closely related to our study, i.e. the five personality factors, cohesion, conflict and task characteristics (autonomy and interdependency).
To quote DeMarco and Lister [12], "Most software development projects fail because of failures with the team running them." Nowadays there is widespread recognition that software process productivity and efficiency is critically dependent on human and social factors [13]. Most research examines the individual qualities of the people involved in the software process. This research tends to consider what personality factors and competencies are required depending on the characteristics of the task at hand [2][3]. We need to go a step further and examine the development team, its interrelationships and personality traits to find out more about what factors influence team performance.
There is little research on group aspects applied to software development. Some studies use a standard test, such as the

MBTI (Myers-Briggs Type Indicator) [14] [15] [16], to determine guidelines for team success depending on software engineer personality types. Another study determines the relationship between competencies, personality traits and team performance [17].
There are also team forming methods based on a quantitative competencies model [18], but they do not consider aspects like team member personality factors. Another team forming method is based on analysing required and available skills [19], although it does not specify how to evaluate people's skills. Nor does it consider group factors or the task.
Zuser and Grechening [20] propose the use of a questionnaire based on abilities and personality traits that provide the team with information during development on finished software projects to improve team performance. The team is built according to the team forming phases of Tuckman's model: forming, storming, norming, performing and adjourning [21].
Despite its importance, little research has been done on the person-team-climate fit in software development. We expect this experimental study to provide a tool for use in software development team formation.

3. EXPERIMENTAL STUDY SET-UP AND DESIGN

The empirical study run fits the description of a quasi-experiment [22]. Quasi-experiments are run when the subjects cannot be randomly assigned to an experimental condition or, alternatively, a treatment cannot be randomly assigned to a group. A quasi-experiment tends not to be very intrusive and relatively low cost. Selected statistical tests are applied to the collected data.
In the following, we describe the participants, response variables and measurement instruments used in the quasi-experimental procedure.

3.1. Participants

The participants were 2nd-year computing students at Autonomous University Madrid's Higher Technical School of Computing during the 2004/2005 academic year. These students were taking the Data Structures and Algorithms subject. The project set was to design and implement software of medium complexity using the Extreme Programming (XP) agile methodology [23].
The number of subjects participating in this empirical study was 105, of which 83 were male (79%) and 22 were female (21%). Of the students, 75% (77 students) were under 21-year-olds and 25% (26 students) were aged from 21 to 30 years.
This group of students was divided into 35 three-member teams working together on software development. These teams were formed at random and their members were blind to the quasi-experimental conditions.

3.2. Response Variables: Software Product Quality and Satisfaction

One of the response variables in this study is the quality of the software product evaluated through code analysis, project documentation and observed team member participation. The projects developed by the teams were graded according to the following weighted formula:

$$\text{Grade} = (((\text{Modularization} * 2 + \text{Testability} * 2 + \text{Functionality} * 2 + \text{Reusability} * 2 + \text{Style} * 2) / 4) * 0.8) + ((\text{Participation} * 10 / 4) * 0.2))$$

The other response variable is development team member satisfaction. The Gladstein questionnaire was used as a measurement instrument to evaluate team satisfaction [24]. People fill in this questionnaire stating how satisfied people they are with their team mates, group work, etc. The scale ranges from 1 (completely disagree) to 5 (completely agree).

3.3. Measurement instruments

The NEO Personality Inventory (NEO-PI-R) [11] was used to study the personality of the participants of each team. This test is composed of 60 questions that measure five personality factors: neuroticism (N) extroversion (E), openness to experience (O), agreeableness (A) and conscientiousness (C). These factors are each divided into 12 items. All the factors are evaluated using a Likert scale. Students are told that there are no right or wrong answers and that they should concentrate on answering accurately and truthfully, scoring responses on a scale from 1 (not at all) to 5 (completely).
The group processes for cohesion and conflict were measured using 13 items taken from the Gross Cohesion Questionnaire [25] and an inter-group conflict questionnaire developed by the Autonomous University of Madrid's School of Psychology. Task interdependency was determined according to Van der Vegt et al.'s questionnaire [26] on a scale from 1 (not at all) to 5 (completely). Finally, task autonomy was measured using a Molleman questionnaire [27].

3.4. Validity threats and control

The internal validity threats known before running this quasi-experiment were:

a) There is no way of guaranteeing that students will do all the specified tasks in the right order or that the work will be shared out adequately. This was countered by allocating two professors for every 50 students responsible for following up project development.
b) Not all teams are as knowledgeable about the task to be undertaken. This was countered by forming teams at random.
c) Questions about the project under development. The professors are directing the quasi-experiment and should take care not become a factor of bias, that is, they should not answer individual questions related to software development. The countermeasure was to establish practical classes to answer these questions so that all the teams were given the same information which they later used as best they could in their project.

4. DATA ANALYSIS AND RESULTS

The following statistical techniques were applied to analyse the data and output the results.

1. Analysis of the correlations between personality factors, task autonomy and interdependency, group processes (cohesion and conflict), satisfaction and final software product quality.
2. Regression between satisfaction and personal conflict, task conflict, task cohesion and interdependency.

4.1. Team Factor Correlations

The analysis of the correlations between personality factors, task autonomy and interdependency, group processes (cohesion and conflict) and satisfaction revealed some positive and significant associations (see Table 1).

Table 1 shows that extroversion correlates positively with another two team member personality factors, openness (r = 0.414 Sig. 0.014) and agreeableness (r = 0.480 Sig. 0.003), as well as with the group process of cohesion (r = 0.469 Sig. 0.005). Similarly, neuroticism has negative correlations with conscientiousness (r = -0.636 Sig. 0.000) and extroversion (r = -0.336 Sig. .049).

Table 1. Table of correlations between personality factors, task characteristics, group processes and satisfaction

	N	E	O	A	C	AUTONOMY	INTERDEPENDENCY	SATISFACTION	CONFLICT	COHESION
N	1	**-0.336***	-0.112	-0.151	**-0.636****	-0.148	-0.214	-0.129	-0.034	-0.186
E		1	**0.414***	**0.480****	0.256	0.302	0.281	0.153	-0.192	**0.469****
O			1	0.302	0.052	-0.085	0.090	0.068	0.236	-0.061
A				1	0.167	**0.503****	**0.500****	**0.334***	-0.157	**0.376***
C					1	0.216	**0.476****	**0.341***	0.007	0.261
AUTONOMY						1	**0.599****	**0.471****	-0.314	0.333
INTERDEPENDENCY							1	**0.797****	-0.243	**0.421***
SATISFACTION								1	**-0.464****	0.294
CONFLICT									1	**-0.496****
COHESION										1

sig. 0.05*
sig. 0.01**

Conflict has a significant negative correlation with cohesion (r = -0.496 Sig. 0.002) and satisfaction (r = -0.464 Sig. 0.005). Conflict is divided into personal conflict and task conflict, a distinction that we will analyse later.

There is a positive linear relationship between students' mean satisfaction and their mean team personality scores for agreeableness and conscientiousness (r = 0.334 Sig. 0.050 and r = 0.341 Sig. 0.045). Satisfaction also correlates positively to task interdependency (r = 0.797 Sig. 0.000) and autonomy (r = 0.471 Sig. 0.004).

Development team interdependency is related to autonomy (r = 0.599 Sig. 0.000), agreeableness (r = 0.500 Sig. 0.002), conscientiousness (r = 0.476 Sig. 0.004) and cohesion (r = 0.421 Sig. 0.012).

No significant relationships were found between software quality and other evaluated factors like autonomy, interdependency, satisfaction, conflict or cohesion. However, a significant correlation was identified between the personality factor extroversion and software quality (r = 0.455 Sig. 0.038) in all the evaluated teams. Finally, there was found to be a significant relationship (r = 0.201 Sig. 0.040) between the individual performance of the 105 participants and team performance.

4.2. Linear Regression of Satisfaction

We adopted a backwards regression model to which all the independent variables were added and then removed stepwise based on the output criteria (lowest absolute value of R2; F criterion >= 0.1). This produced the model shown in Table 2.

Note that the procedure was divided into two steps, yielding a coefficient of determination or corrected correlation coefficient of R2 = 0.733. In other words, the model predicts or explains from interdependency, cohesion and task conflict a variance (VD) of 0.733 in the satisfaction variable. Personal conflict is removed. It is clear that personal conflict makes no contribution to the model's explanation.

Table 2. Table of correlations between personality factors, task characteristics, group processes and satisfaction

	Non-standardized regression coefficients	Standardized regression coefficients
	B	**Beta**
(Constants)	8.013	
INTERDEPENDENCY	0.521	0.765
COHESION	-0.168	-0.182
TASK CONFLICT	-0.674	-0.380

R^2 = 0.733

The coefficient carrying the heaviest predictive weight for satisfaction is interdependency with Beta = 0.765. Teams are small and have to complete the activities in a very short time frame. This they would be unable to do without interdependency, and would find failure very frustrating.

Table 3 shows how the linear regression converges with correlations between the model variables, and interdependency and satisfaction correlate at r = 0.797 (Sig. 0.000). It also shows how task conflict (r = - 0.526 Sig. 0.001) is more negatively related to satisfaction than personal conflict (r = - 0.354 Sig. 0.037). According to Table 3, then, if there is conflict between the team members, they will logically find it more difficult to reach agreement on how to do the task.

Table 3. Regression Model for Satisfaction

	SATISFACTION	PERSONAL CONFLICT	TASK CONFLICT	INTERDEPENDENCY
SATISFACTION	1	**-0.354***	**-0.526****	**0.797****
PERSONAL CONFLICT		1	**0.637****	-0.178
TASK CONFLICT			1	-0.287
INTERDEPENDENCY				1

sig. 0.05*
sig. 0.01**

5. DISCUSSION AND CONCLUSIONS

Figure 1 illustrates the results of our study. Note that the most highly correlated personality factors are agreeableness and extroversion. Figure 1 also highlights the importance of the group factor cohesion because of its relationships both to the above personality factors and to task characteristics.

Figure 1 reveals that there are several relationships between different personality factors, some of which are easier to understand and interpret than others. For example, the extroversion personality factor appears to be clearly related to openness and agreeableness. The logical explanation is that relationships will be more agreeable and fluid, with less tension between team members, in teams with average extroversion levels. In such a team environment, people will obviously find it easier to propose innovative methods for doing the work. Therefore, the quality of the work in such teams is likely to improve, as is the result (in this case the developed software product).

Team cohesion will tend to be stronger at average extroversion levels. These conditions in which the team is more united will perhaps support and boost the openness to experience and agreeableness factors among team members.

There are several points to be made about satisfaction (Figure 1). We find that the most satisfied students are precisely the ones that score higher on the agreeableness and conscientiousness factors. Molleman et al. [4] also discovered this positive relationship between satisfaction and conscientiousness. In other words, people feel more satisfied in teams where there is a more agreeable and/or conscientious environment. Satisfaction levels are also higher when the team can decide on how to develop and organize the work to be done (autonomy). Finally, the more conflict is perceived in the team, the less cohesion and satisfaction there is among its members.

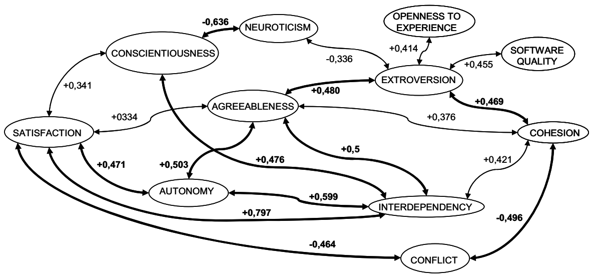


Figure 1. Personality-Group Behaviour-Task Characteristics-Quality/Satisfaction Correlations

As regards software quality, the only clear relationship found was to extroversion (Figure 1). The scores for some NEO-PI-R factors are outside the standardized values for the sample of this study. The results might possibly have been clearer if the sample had been more variable. This would to some extent explain the lack of statistically significant relationships to developed software quality identified in other research. However, the one relationship we did identify matches the results reported by Barrick and Mount [6], Barry and Stewart [5] and Barrick et al. [7]. While these works deal with different task types, they do have the common denominator of high interaction between team members and a unanimous vision of the goals to be achieved. These are also key characteristics for the agile development method. Therefore, average extroversion within teams is considered to improve the software product quality for this development method.

REFERENCES

1. M.I. Kellner, R.J. Madachy, D.M. Raffo.: Software Process Simulation Modelling: Why? What? How?. Journal of Systems and Software. Vol. 46, pp. 91-105, (1999).
2. J. Wynekoop, D. Walz.: Investigating traits of top performing software developers. Information Technology & People. Vol. 13(3), pp. 186-195, (2000).
3. S.T. Acuña and N. Juristo.: Assigning people to roles in software projects. Software: Practice and Experience. Vol. 34, pp. 675-696, (2004).
4. E. Molleman, A. Nauta, K.A. Jehn.: Person-Job Fit Applied to teamwork: A Multi-Level Approach. Small Group Research. Vol. 35, pp. 515-539, (2004).
5. B. Barry, G.L. Stewart.: Composition, process and performance in self-managed groups: The role of personality. Journal of Applied Psychology. Vol. 82, pp. 62-78, (1997).
6. M.R. Barrick, M.K. Mount.: The Big Five personality dimensions and job performance: A meta-analysis. Personnel Psychology. Vol. 44, pp. 1-26, (1991).
7. M.R. Barrick, G.L. Stewart, M.J. Neubert, M.K. Mount.: Relating Member Ability and Personality to Work Team Processes and Team Effectiveness. Journal of Applied Psychology. Vol. 83, pp. 377-391, (1998).
8. H.-L.Yang, J.-H. Tang.: Team structure and team performance in IS development: a social network perspective. Information & Management. Vol. 41, pp. 335-349, (2004).
9. S. Faraj, L. Sproull.: Coordinating expertise in software development teams. Management Science. Vol. 46(12), pp. 1554-1568, (2002).
10. W.S. Humphrey, M.D. Konrad.: Motivation and Process Improvement. In: Software Process Modeling. Springer, (2005).
11. Jr.P.T. Costa, R.R. McCrae.: NEO Personality Inventory. Psychological Assessment Resources, (1992).
12. T. DeMarco, T. Lister.: Peopleware: Productive Projects and Teams (2nd ed): New York: Dorset House, (1999).
13. B.W. Boehm, C. Abts, W.A. Brown, S. Chulani, B.K. Clark, E. Horowitz, R. Madachy, D.J. Reifer, B. Steece.: Software Cost Estimation with COCOMO II. Upper Saddle River: Prentice Hall PTR, (2000).
14. R.H. Rutherford.: Using personality inventories to help form teams for software engineering class projects. SIGCSE-Bulletin. Vol. 33(3), pp. 76–76, (2001).
15. R.P. Bostrom, K.M. Kaiser.: Personality differences within systems project teams: Implications for designing solving centers. Proceedings of the Eighteenth Annual ACM SIGCPR Conference. pp. 248–285, (1981).
16. L.T. Hardiman.: Personality types and software engineers. IEEE Computer. Vol. 301(10), pp 10–10, (1997).
17. K. White, R. Leifer.: Information systems development success: Perspectives from project team participants MIS Quarterly. Vol. 10(3), pp. 215–23, (1986).
18. G. Burdett, R-Y. Li.: A quantitative approach to the formation of workgroups. Proceedings of the ACM SIGCPR Conference. pp. 202–212, (1995).
19. A. Zakarian, A. Kusiak.: Forming teams: An analytical approach. IIE Transactions. Vol. 31, pp. 85–97, (1999).
20. W. Zuser, T. Grechening.: Reflecting skills and personality internally as means for team performance improvement. Proceedings of the 16th Conference on Software Engineering Education and Training, IEEE Computer Society, (2003).
21. B. Tuckman.: Developmental sequence in small groups. Psychological Bulletin. Vol. 63, pp. 384–399, (1965).
22. T.D. Cook, D.T. Campbell.: Quasi-Experimentation Design and Analysis Issues for the Field Settings. Boston, MA: Houghton Mifflin, (1979).
23. K. Beck.: Extreme Programming Explained: Embrace Change. Addison-Wesley, Reading, (1999).
24. D.L. Gladstein.: Groups in Context: A Model of Task Group Effectiveness. Administrative Science Quarterly. Vol. 29, pp. 499, (1984).
25. J.P. Stokes.: Toward an understanding of cohesion in personal change groups. International Journal of Group Psychotherapy. Vol. 33, pp. 449-467, (1983).
26. G. Van der Vegt, B. Emans, E. Van de Vliert.: Affective reactions to individual task interdependence in outcome interdependence groups. Personnel Psychology. Vol. 54, pp. 51-69, (2001).
27. E. Molleman.: The modalities of self-management: the "must", "may", "can" and "will" of local decision making. The International Journal of Operations and Production Management. Vol. 20, pp. 889-910, (2000).

Towards Constructing High-available Decentralized Systems via Self-adaptive Components

Xi Sun, Li Zhou, Lei Zhuang, Wenpin Jiao+, Hong Mei

Software Institute, School of Electronics Engineering and Computer Science, Peking University
Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education
Beijing 100871, P.R. China
{sunxi, zhouli04, zhuanglei04, jwp, meih} @sei.pku.edu.cn

ABSTRACT

In decentralized computing environments, systems are built mainly from components that are developed and maintained independently by different third-party providers. The executions and evolutions of components located on distributed sites are beyond the control of the system developers and the availabilities of those components are, to some extent, unpredictable because of their own tendencies and the unstable network. As a result, it is still a great challenge to construct high-available decentralized systems. In this paper, a self-adaptive component model is proposed to model those components distributed on the Internet and the running framework is described for constructing systems composed of self-adaptive components. Self-adaptive components can adjust their knowledge about the availabilities of the required services via learning from the feedback of historical invocations. Based on the knowledge, components can find the most appropriate service providers effectively and automatically. Experiments show that systems can always gain a high availability under the dynamic decentralized environment by using the approach.

1. INTRODUCTION

The emergence and popularization of new computing paradigms such as pervasive computing, grid computing, and service computing, make today's software to be increasingly "Internet-scale" and "service-oriented". Many Internet-based applications reuse different third-party components to improve their development productivity and quality. We call these systems as *decentralized systems* [1], which represents **systems that are built mainly from components that are developed and maintained independently by different providers, on hardware nodes that are not under the control of the system developers**. As more and more services and decentralized systems will certainly be deployed in the coming years, how to construct high-available decentralized systems is becoming a crucial challenge.

System availability can be defined as the degree to which a system is operational and accessible when required for use [2]. As situated in the unstable network environment, decentralized system's availability is always weakened by the underlying connectivity failures. Moreover, third-party services often evolve outside of the systems, join or quit the network without notifying others. As a result, service availability often keeps on changing at runtime and, to some extent, is unpredictable. On the other hand, the proliferation of services available on the Internet increases the candidate providers. To improve the overall system availability, research efforts [3] [4] have been put into clustering many function-identical services and binding most adequate service(s) automatically during the execution time.

Unfortunately, most of existing solutions are using a centralized architecture style, which may be unsuitable in a large, decentralized environment with a large number of independent and distributed components. In the web service community, Ran [5] presents an approach to storing quality information (e.g., availability) of web services into the UDDI registry. However, service availability depends on not only the service implementation but also the network status between service requestors and providers, and a service highly available to the UDDI server may provide a poor user-experience to the actual client. Another typical approach is to delegate customer requests to a service broker [6]. However, this approach needs the central broker to process a large number of client requests.

To cope with the uncertainty and dynamics in decentralized environments without a global coordinator, system components are required to approximate the current status of remote resources [1] (i.e., software entities constructing Internet-scale applications). That is, components should estimate the availabilities of service providers, and adjust the estimated values according to the service evolutions. Consequently, two important problems should be addressed at least: (1) given a collection of third-party services, how can the currently most available candidate be found effectively without a central information service? (2) When the quality of service or network connectivity changes, how can the system be aware of and adapt to the change automatically?

To address above problems, in this paper, we present a component framework to facilitate constructing high-available self-adaptive systems in the decentralized environment. The main contributions of this paper are: First,

+ Corresponding Author

we propose a self-adaptive component model to encapsulate the availabilities of remote components and resources as adaptable "knowledge". This model allows a component to dynamically estimate the availabilities of remote entities at runtime to adapt to the dynamic environment. Second, we design and implement an algorithm to reason about the availabilities of required services via learning from the feedback of historical invocations. This algorithm enables components to update their knowledge automatically. Finally, a running infrastructure is implemented to increase the feasibility of our approach.

The reminder of this paper is organized as follows. Section 2 proposes the self-adaptive component model, and describes the process of constructing systems based on self-adaptive components. Section 3 presents the algorithm for dynamically estimating and adjusting the availabilities of services in detail. Section 4 reports the experimental studies and analyzes the results. Section 5 compares with some related work. Finally, Section 6 concludes this paper and discusses our future work.

2. CONSTRUCTING HIGH-AVAILABLE SYSTEMS VIA SELF-ADAPTIVE COMPONENTS

In this section, we first give a definition to the self-adaptive component. Then, we illustrate the runtime structure of the components in our current implementation. We finally describe the system constructing process.

1. Self-adaptive component definition

First, the definitions of interface and service are provided for completeness.

An interface is defined by Szyperski as:

Definition 1. An *interface* is a set of named operations that can be invoked by the clients [7].

We define a service as a program that implements and publishes one or more interfaces. A service may also depend on zero or more required interfaces.

Definition 2. A *service* is defined as a triple as follows:

S = < name, PIs, RIs >, where:

- *name* is the unique name of service S;
- PIs is the set of provided interfaces of S;
- RIs is the set of required interfaces of S;

Self-adaptive components in our approach specify the services (including those they provide and they require) as interfaces explicitly. For each required interface, a set of service candidates should be given at the component deployment stage. In our current implementation, the candidates list can also be modified at runtime via component's reflective interface. Due to the space limit, the details of reflective interface are not included in this paper.

Definition 3. A *self-adaptive component* can be formally defined as a tuple as follows:

C = < *name*, PIs, RIs, D, *Beh* >, where

- *name* is the unique name of component C;
- PIs is the set of provided interfaces of C;
- RIs is the set of required interfaces of C;
- D is a relation to specify C's dependencies, where: $D = RIs \times (S \rightarrow \Re)$,
  That is to say, for every required interface, there exist one or more service candidates that implement this interface (S is the set of the services). Moreover, for every service, component has a variable (which is with range of $\Re$, the set of real numbers) to record the estimated value of the service's availability.
- *Beh* is the component's behavior to maintain the component's dependencies and to adjust those estimated values related to the depended services' availabilities. The component behavior is realized via an algorithm, which will be explained in depth in Section 3.

2. The runtime structure of self-adaptive components

When developing a self-adaptive component, developers should provide an implementation of the provided interfaces and a specification of the required interfaces separately. In the required interfaces specification, each service candidate should be specified with service type (e.g., web services, RESTful services [8]), address (e.g., URL), communication protocol (e.g., SOAP [9]), message structure, and etc.

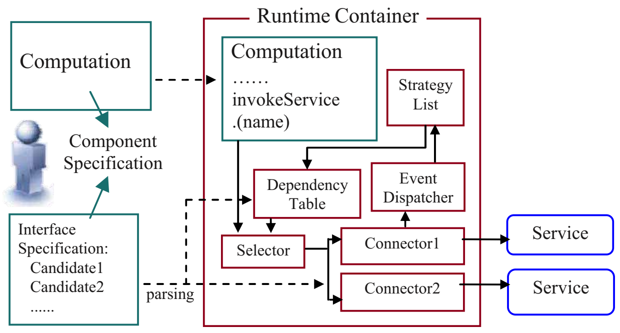


Figure 1 self-adaptive component: structure overview

Figure 1 shows a runtime self-adaptive component from the structural perspective. The component is running in a container, and the service candidate list of every required interface is stored in the container's dependency table. The component's invocations to the required interfaces are delegated to the container's selector module, which chooses one of the candidates and then requests the service via the candidate's connector. Our current implementation provides a tool to assist in generating the web service connector (i.e., the invocation stub) according to the given web service specification.

As shown in Figure 2, briefly, there are three phases when a self-adaptive component invokes a service:

1. At the initialization stage, for each required interface, self-adaptive component loads the list of service candidates that implement this interface. Then, the component designates a selection probability to each service candidate. Each service is initially assigned with the same probability.

2. When the component needs to invoke a service, the

selector module chooses a candidate according to the current specified probability distribution. (Steps 1-3 in Figure 2)

3. After the invocation, component adjusts the selection probabilities related to the service candidates according to the invocation feedback event (Steps 4-6 in Figure 2). The adjustment is determined by the learning algorithm, which is registered in the strategy list.

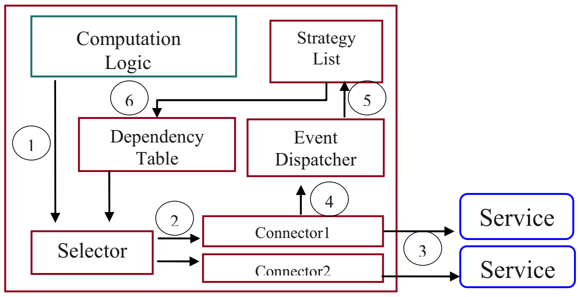


Figure 2 the behaviors of self-adaptive components

3. Self-adaptive component based systems.

Self-adaptive component-based systems are composed of many self-adaptive components and service candidates. Note that a self-adaptive component is also a service from the definitions in section 2.1; thus, the components composing the system can be organized in a layered structure, in which a self-adaptive component can be used as one service candidate of other components.

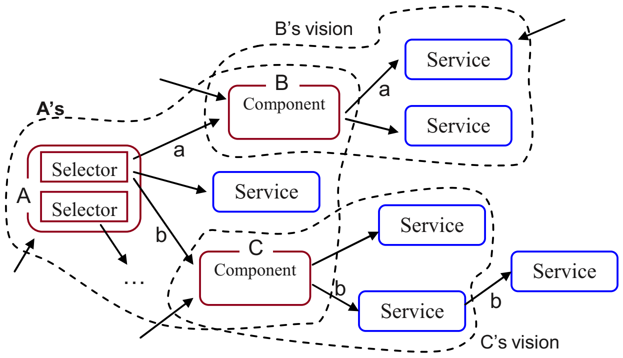


Figure 3 self-adaptive component based system

In decentralized environments, services (include the self-adaptive components) are running independently and continuously. Every service processes the requests from many different requestors, and may send request to other services. Furthermore, none of the services has a global view of the system: each self-adaptive component only knows some dependent services, in other words, a part of the system architecture. Figure 3 shows these components' local view of the environment.

During the execution time, which services participate in the current computation task is dependent on the self-adaptive components' selection results. As shown in Figure 3, a request for service to component A may lead to the invocation to component B during A's execution, and the path labeled with 'a' is the whole service process; to process another request, A may select component C and the process may be the path labeled with 'b' in the figure. Although the selection is dynamic, as the components process more invocations, the learning algorithm will enable them to find the most available service providers, and then the service process is increasingly stable.

In a word, the construction of a decentralized system is determined at runtime. Self-adaptive component-based systems are represented as coalitions of distributed entities which are self-organized to guarantee the high availability.

3. The Estimated Service Availability Adjustment Algorithm

Our adjustment algorithm borrows the idea of the simulated annealing algorithm [10], but has many important differences in addressing the requirements of finding the service with highest availability in dynamic environments.

The estimated availability value v_i of service candidate$_i$ is calculated in **formula 1** below.

$$v_i = e^{\frac{\alpha \times history}{temperature}} \quad (1)$$

There are two variables in formula (1), *history* and *temperature*. Variable *history* denotes the historical success rate of service candidate$_i$. The success rate is quantified as the ratio of the number of successful completed interactions between component and candidate$_i$ to the total number of attempted requests between them. A candidate performs better in the history will have a bigger probability to be chosen in the next invocation. We enlarge the influence of *history* by multiplying it by a constant α , which is designated to 60.0 in our current implementation.

Variable *temperature* denotes the degree of freedom to select among the candidates. As we known, even a highly available service may fail occasionally due to network failures or other problems; meanwhile, at the beginning, it's difficult for the component to differentiate the candidates based on insufficient historical invocation feedback. Therefore, a high *temperature* is given in the initial learning stage, thus the candidates will have similar estimated values (i.e., similar probability to be selected). As the component gains more experiences, the *temperature* is decreased continuously and the estimated value becomes more sensitive to the value of *history*. When the *temperature* drops to a minimum, the probability of the most available service candidate will be significantly different from those of the less available ones.

During the annealing process, *temperature* is adjusted as follows:

Formula 2: Formally, let '*T*' be the temperature and *minT* be its lower bound. Let *'minCT' and 'totalCT'* denote the minimum and the sum of the historical invocation times of service candidates, respectively. Let Δ be the incremental operator. Then,

$\Delta T_{anneal} = -Min\{T - minT,\ \Delta\lfloor totalCT / i \rfloor + \Delta minCT\}$ (2);

where *'i'* is the adjusting parameter. In our experiment, its value is set as 50, and *'minT'* is set as 1.

This formula represents that *temperature* is decreased every time when the increment of *totalCT* is over *i* times, which implies that the component has gained rich experience. We also decrease *temperature* when *minCT* is increased, which indicates that the component gains more availability information about the candidate with the least invocation times. The annealing process ends when *temperature* drops to *minT*.

On the other hand, a service candidate with a high success rate in previous invocations may become unavailable at runtime; meanwhile, the component may add new service candidate dynamically. In these cases, we should increase the *temperature* to provide other candidates (or the new services) with more opportunities. And it is called the '*heating process'*.

Formula 3: Formally, for a service candidate, let '*recentSR*' and *'hisSR'* denote the success rates of the recent '*j*' times of invocations and the historical invocations, respectively. During the heating process, *T* is adjusted as follows:

$$\Delta T_{heat} = \begin{cases} m \text{ (if } recentSR - hisSR > 0.1 \text{ and } T = minT\text{);} \\ n \text{ (if a new service is added);} \\ 0 \text{ (other cases).} \end{cases} \quad (3);$$

in this formula, *'m' and 'n'* are the heating parameters. In our experiment, *'j'* is set as 40, and values of *'m' and 'n'* are set as 20 and 50, respectively.

This formula represents that if the difference between *recentSR* and *hisSR* is more than 10% and current *temperature* has been decreased to the lower bound, the *temperature* should be increased by *m*; note that after the heating process, the changed candidate's old invocation records will be removed. If a new service is added into the candidate list, we will increase the *temperature* by *n*.

After an invocation, the component will receive a feedback that contains the invoked candidate and the invocation result (success or failure). Then the component will adjust its estimated value of candidate availability as follows:

```
Algorithm 1: adjust the candidates' estimated values when a new feedback is received.
1. If a new feedback is received, load the required service's candidate list from dependency table.
2. Add this feedback into the list of history records. Update the values of totalCT, minCT, recentSR, hisSR, etc.
3. Calculate $\Delta T_{anneal}$ and $\Delta T_{heat}$ based on formula (2), (3). Update the value of temperature.
4. Calculate the candidates' estimated values of availability based on formula (1).
5. Normalize these values so that the sum will keep being 1 to facilitate the selection; that is, $v_i = \frac{v_i}{\sum v_i}$
```

A Self-adaptive component can add a new service provider into its candidate list at runtime, and the candidates' values will be adjusted as follows:

```
Algorithm 2: adjust existing candidates' estimated values when a new provider is added.
1. Load the required service's candidate list from dependency table and add the new service provider.
2. Reset minCT to 1.
3. Calculate $\Delta T_{heat}$ based on formula (3). Update the value of temperature.
4. Recalculate the candidates' estimated values of availability based on formula (1).
```

As the historical records become rich and the *temperature* is gradually decreased, the component will have the highest probability to select the most available candidate. The correctness of our algorithm can be proven by using the Law of large numbers [11] in Statistics. Due to space limit, the proof is not discussed in this paper.

4. EXPERIMENTAL STUDY

Our experimental study is divided into two stages. First, we evaluate our learning algorithm by comparing it with some general algorithms; in the experiment, we use a component with several service candidates that may change their availability dynamically. Then, we build a system with multiple self-adaptive components to observe the overall availability of the system. For the requirements of experiments, we programmed several simulated services, whose availability can be changed by us at runtime.

1. The evaluation of the learning algorithm

Figure 4 shows the self-adaptive component and the services used in our first experiment. For simplicity, we assume the self-adaptive component's computation is always available (i.e., the availability is 100%), whereas the simulated services may throw an exception according to a specified probability.

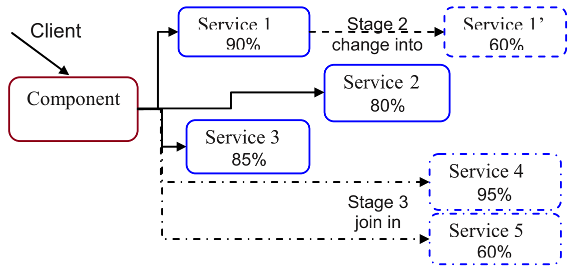


Figure 4 the experiment to evaluate the learning algorithm

In this experiment, a client program will generate service requests to the component continuously at a fixed speed (About 200 requests / per minute). The component will select one of the function-identical services to delegate a part of its tasks; after eight minutes, we change the service$_1$'s availability to 60% to simulate a network problem (stage 2);

and after another eight minutes, we add two different new services to the component's candidate list (stage 3).

Two general algorithms are also implemented to compare with our learning algorithm (named as LA). The algorithm named *test-and-choose* (T&C) is to select every service candidate n times first, and then choose the service with highest availability. Another algorithm named *recent-successful-choice* (RSC) is to select a service randomly, and then choose the same one at the next time if this invocation is successful; otherwise (the invocation is failed), the component delegates the following request to another service candidate randomly.

Figure 5 shows the obtained experiment results, which have been averaged over fifty runs.

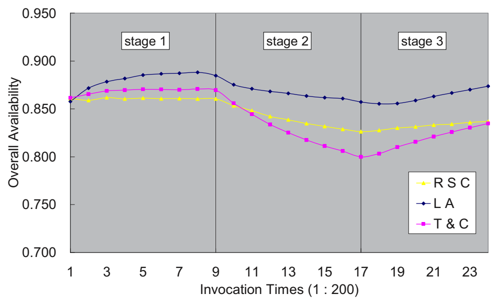


Figure 5 the results of Experiment 1

As shown in Figure 5, at the stage 1, the component using the learning algorithm finds the high quality candidate fast and achieves highest availability; the T&C algorithm may find the best candidate (the success rate depends on the test times n, which is set as 50 in our experiments), but it has to spend a period of time in invoking each candidate many times to make decision. At the stage 2, although both LA and RSC can switch to another candidate quickly, the LA algorithm can find the new highest available candidate (i.e. Service3) and send most requests to it, thus gains a better availability. On the other hand, the T&C algorithm cannot detect the change of service availability, which may occur frequently in a dynamic environment. Experiment results also show that although RSC algorithm has a quick response to invocation failures by sending following requests to another candidate, this strategy cannot recognize the best candidate so that system cannot obtain the theoretically maximal availability after the environment becomes stable later. Therefore, we believe that the LA algorithm is effective; based on our approach, systems can gain a high availability under the dynamic and decentralized environment.

2. The evaluation of self-adaptive component-based systems

Experiment 2 is to test the overall availability of a self-adaptive component-based system. In figure 6, there are three self-adaptive components and six services in the test environment. As mentioned in section 2.3, services need to process requests from different clients, and every request will have an influence on the self-adaptive components' decision-making. To simulate the real environment, for each self-adaptive component, we place a client to send requests simultaneously. The component A is our test's target.

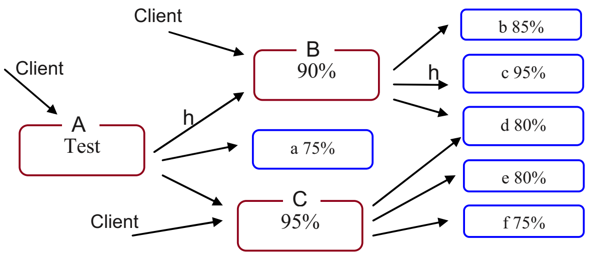


Figure 6 the experiment to evaluate the overall availability of a self-adaptive component based system

At the early stage, component B and C have not found their best dependent service candidates; as a result, their own availabilities are not stable, and from the perspective of component A, the availabilities of component B and C keep changing at runtime. As component B and C gain more experiences from invoking their candidates to finish the tasks delegated from component A and other clients, these self-adaptive components will recognize their best candidates based on the learning algorithm and send most of requests to them. Consequently, their availabilities reach to their maximal values and become stable. Meantime, component A will be able to find its most available candidate. Figure 7 shows the experiment result, which represents that the system finds the best invocation process quickly (Note that the path labeled by 'h' in Figure 6 is the only path whose overall availability exceeds 80%), and with the run times increasing, the availability of the system becomes higher and gradually tends to the theoretically maximal value.

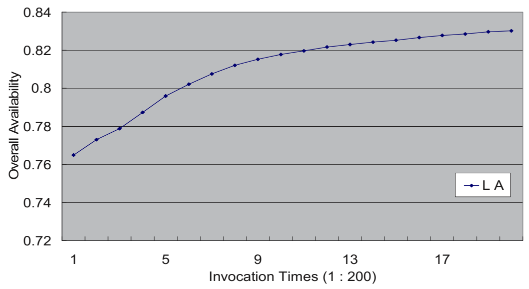


Figure 7 the results of Experiment 2

3. Discussion

So far, we have given an effective algorithm to find the most available service provider from many third-party candidates. As situated in a dynamic environment and unknown service providers, a self-adaptive component has to take a period of time to learn from the feedback of invocations.

To further reduce the learning cost, one way is to "learn from others", that is, to enrich a component instance's experience by reusing other instances' invocation results. In our current work, we are extending our component framework to support a mechanism that enables the component instances to share knowledge among each other. In the improved framework, for each self-adaptive component type, an Instance Manager will record all the active component instances and their invocation times, and when a new component instance is created, it will be initialized with the most "experienced" instance's invocation history. In other words, this new instance will perform as good as the trained instances without a learning curve in its early stage.

5. Related work

Many research efforts have focused on how to improve the system availability at runtime. In some *distributed systems* (e.g., systems built on the internal services within an enterprise boundary), system components are deployed throughout a group of machines, and the lifecycles of components are controlled by a centralized management server. The availability of this kind of systems can be improved via system redeployment [12]. The redeployment process mainly includes the following steps: first, the management server monitors and calculates the reliability of the underlying network; then, the server determines the optimal deployment plan based on the components' availability, their interaction frequencies, the reliability of network connectivity, etc; finally, the server redeploys the system via component migration. However, in *decentralized systems*, redeployment techniques are unsuitable because components deployed in third-party provider's machine cannot be migrated. Moreover, a service's availability may change at runtime, whereas existing redeployment algorithms often consider it as a static value.

Availability is an important aspect of Quality of Service (QoS). Several related work on QoS has been done in the area of service selection. Zeng et al. [13] presents a platform which addresses the issue of selecting web services for the purpose of their composition in a way that maximizes user satisfaction expressed as utility functions over QoS attributes. As mentioned in Section 1, this centralized planner may be unavailable in dynamic environments where it is not always possible to access to the server. Moreover, the planner may become a potential bottleneck when the number of the services grows up.

Several existing research projects use machine learning techniques to adapt systems to the changing environments. Doshi et al. [14] use Markov decision process (MDP) to model web service workflow composition. They interleave MDP-based workflow generation and Bayesian model learning to produce robust workflows. In that approach, the optional services are predefined and cannot be modified at runtime. Dowling [15] proposed an adaptive component framework named K-Component, which facilitates building autonomic systems by collaborative reinforcement learning. In this model, adaptation actions of the adaptive components are fixed after the compile stage, thus how to deal with new coming services is not addressed.

6. Conclusion and future work

In this paper, we proposed an approach to constructing high-available decentralized systems in a dynamic environment. We introduced our self-adaptive component model and gave a detailed description of the self-adaptive component-based systems. We also explained the learning algorithm in depth and validated its effectiveness by several experiments

For the future, we plan to further investigate the self-adaptive component model and the algorithm to concern multiple system quality attributes (such as security, response time) for the purpose of building high-quality decentralized systems.

Acknowledgment

The research was sponsored by the National Grand Fundamental Research 973 Program of China under Grant No.2005CB321805 and the National Nature Science Foundation of China under Grant No.90412011, 90612011.

References

[1] R. Khare and R. N. Taylor, "Extending the Representational State Transfer (REST) architectural style for decentralized systems," in *Proceedings of 26th International Conference on Software Engineering*, 2004, pp. 428-437.
[2] IEEE Standard Computer Dictionary: *IEEE Standard Computer Glossaries*. New York. 1990
[3] G. Huang, L. Zhou, X. Z. Liu, H. Mei, and S. C. Cheung, "Performance Aware Service Pool in Dependable Service Oriented Architecture," *Journal of Computer Science and Technology,* vol. 21, pp. 565-573, 2006.
[4] B. Benatallah, Q. Sheng, and M. Dumas, "The Self-Serv environment for Web services composition," *IEEE Internet Computing,* vol. 7(1), pp. 40-48, 2003.
[5] S. Ran, "A model for web services discovery with QoS," *ACM SIGCOM Exchanges,* vol. 4, pp. 1-10, 2003.
[6] K. Sycara, M. Paolucci, J. Soudry, and N. Srinivasan, "Dynamic discovery and coordination of agent-based semantic Web services," *IEEE Internet Computing,* vol. 8(3), pp. 66-73, 2004.
[7] C. Szyperski, *Component Software*. Addison-Wesley, 1998. ISBN: 0-201-17888-5
[8] RESTful Service: http://www.answers.com/topic/representational-state-transfer
[9] SOAP Specification: http://www.w3.org/TR/soap/
[10] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Prentice-Hall, 1995.
[11] J. L. Devore, *Probability and statistics for engineering and the sciences*. Duxbury Press Belmont, 1995.
[12] M. Mikic-Rakic, S. Malek, and N. Medvidovic, "Improving Availability in Large, Distributed, Component-Based Systems via Redeployment," *Proceedings of 3rd International Working Conference on Component Deployment (CD 2005), Lecture Notes in Computer Science,* vol. 3798, pp. 83-98, 2005.
[13] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-aware middleware for Web services composition," *IEEE Transactions on Software Engineering,* vol. 30, pp. 311-327, May 2004.
[14] P. Doshi, R. Goodwin, R. Akkiraju, and K. Verma, "Dynamic Workflow Composition Using Markov Decision Processes," *International Journal of Web Services Research,* vol. 2, pp. 1-17, Jan-March 2005.
[15] J. Dowling, "The Decentralised Coordination of Self-Adaptive Components for Autonomic Distributed Systems," University of Dublin, Trinity College, phd thesis,October 2004.

SAFES: A Static Analysis for Field Security in Java Components

Aiwu Shi and Gleb Naumovich
Department of Computer and Information Science
Polytechnic University
6 MetroTech Center, Brooklyn, NY 11201, USA
ashi01@cis.poly.edu, gleb@poly.edu

Abstract

In this paper, we address the security problems of vulnerable Java components by means of a combined static analysis. More precisely, we use escape analysis relying on points-to and dependence analysis to determine the information leaks of security-sensitive fields in the components under analysis, and apply mutability analysis to figure out the mutability of the fields in the analyzed components. Finally, our analysis approach combines both escape and mutability information to discover the security problems in the components and reports to the developers for reviewing and fixing them.

`SAFES`, which implements our static analysis technique, has been developed to statically detect field security in Java components. Using the tool, we have experimented on a set of publicly-available J2EE applications and explored the trade-offs between the precision and performance with a variety of underlying analysis techniques as well as a couple of different implementations for mutability analysis. On the benchmarks examined, the results of our study show the effectiveness and efficiency with few false positives.

1 Introduction

In this paper, we present a novel static analysis technique for detecting security threats to confidentiality and integrity in Java components. Our technique builds upon work done in escape analysis and mutability analysis and uses a combination of both analysis techniques targeted at object fields in Java components. The key insights behind the development of the technique go as follows: (1) whether the information controlled by the fields in a Java component is accessible beyond the boundary of the component; (2) whether an untrusted client can modify confidential data encapsulated in the component. As a result, we first use escape analysis, which combines existing points-to analysis and dependence analysis, to reason about the accessibility of the fields in a component. Then, along with the escape information, our technique employs mutability analysis for checking whether the fields are modifiable outside the domain of the component. The potentially accessible and (or) modifiable fields found by our analysis approach to violate the security policy are reported for the component developers to fix the security vulnerabilities.

We have implemented the static analysis technique in a prototype tool, `SAFES`, a `Static Analysis for Field sEcurity in java componentS`, and used the tool to evaluate the technique on a set of benchmark applications.

The main contributions of this work to the field of static analysis for security are as follows:

1. A combined static analysis for Java components, aimed at detecting security vulnerabilities in the components.

2. A taxonomy of software vulnerabilities, addressed by the notions of escape and mutability analysis.

3. `SAFES`, a prototype tool that implements our proposed analysis technique, written in pure Java and based on some existing frameworks.

4. An experimental validation of SAFES's ability to detect the security flaws, showing the effectiveness and the efficiency of the technique.

The remainder of this paper is organized as follows. Section 2 describes software vulnerability categories with real examples. Section 3 presents the novel static analysis combining escape analysis and mutability analysis. We introduce the `SAFES` architecture and implementation in Section 4. In Section 5, we run `SAFES` on a set of publicly-available benchmark applications, and validate the precision and efficiency of our tool and analysis technique. Section 6 reviews related work. Finally, we conclude the paper in Section 7.

```
public class NBA{    public final class NBA{
 ......               ......
 public long data;    private long data;
 public int  size;    private int  size;
 public Class type;   private Class type;
 }                    }

Before (JMF 2.1.1c)    After (JMF 2.1.1e)
```

Figure 1. Information Leak Example: JMF

2 Software Vulnerability Categories with Examples

We classify software vulnerabilities into three general categories in this paper.

- `Information Leaks`, which occur when an interaction between untrusted clients and trusted software components returns sensitive information that should be inaccessible to clients. Therefore, this sort of vulnerabilities violate the confidentiality policy of information security.
- `Modification Access Violations`, which occur when the untrusted clients invoke accessible methods to interact with the trusted component and are able to modify sensitive data controlled by the component. As a result, data integrity of information security is compromised by this category of vulnerabilities.
- `System Resource Leaks`, which occur when the trusted component allows untrusted clients to perform sensitive operations upon the resources of the underlying computer system, e.g., executing arbitrary commands in a system shell.

Our work in this paper focuses on addressing the first two vulnerability categories using a combined static analysis technique.

Example 1: (Information Leaks: JMF API) The Java Media Framework API (JMF) extends the J2SE for multimedia developers by providing a powerful toolkit to develop scalable, cross-platform technology. The public JMF 2.1.1c class `com.sun.media.NBA` exposes a public pointer to a physical memory by means of `public, long` field `data`, so untrusted applets may read the system memory and crash the JVM or gain unauthorized privileges [9]. The vulnerability is fixed in JMF 2.1.1e or later, as presented in Figure 1.

To prevent leaking of secret data from new methods, the creation of subclasses is forbidden in the fixed version by declaring class with `final`. In addition, three `public` fields are changed into `private` in order to avoid direct access to the sensitive data by untrusted parties.

```
// Before (JDK 1.42_04)
public class org.apache.xpath.compiler.
       FunctionTable{
 public static org.apache.xpath.compiler.
    FuncLoader[] m_functions;
 ......
}

// After (JDK 1.42_05)
public class org.apache.xpath.compiler.
       FunctionTable{
 private static org.apache.xpath.compiler.
    FuncLoader[] m_functions;
 ......
}
```

Figure 2. Modification Access Violation Example: XSLT

Example 2: (Modification Access Violation: XSLT Parser) The XSLT (eXtensible Stylesheet Language Transformations) parser embedded in Sun JDK is directly taken from the Apache Xalan [1]. In JDK 1.42_04, malicious objects may be inserted into the non-final static array `m_functions` of public class `org.apache.xpath.compiler.FunctionTable` in the XSLT parser to allow the untrusted applet to escalate privileges [10]. The bug is fixed in JDK 1.42_05 or later, as presented in Figure 2.

The `private` modifier of the field `m_functions` prohibits malicious codes to modify the table consisting of the built-in functions in the XSLT parser and enhances functionality of the parser component to the level needed for running the component securely in JVM.

3 Combined Static Analysis

To reason about potential information leaks from a component, we present the escape analysis, which employs existing points-to analysis for reference type fields (or variables) and dependence analysis technique for primitive type fields (or variables), to compute the accessibility of the fields in the component and detect whether a security-sensitive field is accessible in some way from outside the component. Specifically, the traditional data and control dependence relationships are extended to better suit our security analysis under the boundary of a software component. The detailed algorithms for detecting potential leaks of sensitive information beyond the domain of a component are proposed in our previous work [15] and implemented in the Escape Analyzer analysis engine of our prototype tool.

```
DetermineSecurity(f)

Inputs: field f to be analyzed, ImmutableFields, EscapeFields
Output: classification of field f as Safe, Visible, Modifiable
Steps:

    if $f \in$ ImmutableFields
        if $f \in$ EscapeFields
            return Visible
        else
            return Safe
        end if
    else
        return Modifiable
    end if
```

Figure 3. An Algorithm to Determine the Security of a Field

To address the modification access violation problem, an extended mutability analysis [15] relying on escape information from the escape analysis technique described above is proposed to determine whether a security-sensitive field can be modified outside the analyzed component. Our mutability analysis is largely based on the earlier work in [14] and nevertheless modifies some definitions and algorithms for improving the analysis precision in the context of components.

With the escape information and field (or class) mutability in hand, we combine the analysis results and present an algorithm to perform the static analysis for security in Figure 3. Sets *ImmutableFields* and *EscapeFields* denote the set of fields that have been classified immutable in the components under analysis (CUA) and the set of fields that escape outside the CUA, respectively. Three levels of security of a given field in the CUA are defined in our analysis, i.e., `Safe`, `Visible` and `Modifiable`. As the names mean, the `Safe` fields are safe since the value of the fields does not escape the CUA and remains unmodifiable from outside the CUA; the `Visible` fields are moderately dangerous since the value of the fields can escape the CUA but cannot be modified outside the CUA; the `Modifiable` fields are highly dangerous since the value of the fields can escape and be modified outside the CUA.

4 SAFES Architecture and Implementation

We have implemented our combined static analysis technique in a prototype tool, called `SAFES`, to statically detect security problems in Java components. The following sections illustrate the architecture of SAFES and describe the implementation details for the analysis engines in SAFES.

4.1 SAFES Architecture

Figure 4 shows the architecture of SAFES, which consists of three major analysis engines, i.e., Escape Analyzer, Mutability Analyzer and Security Analyzer. The inputs to SAFES are the Java codes to be analyzed as well as a user specification. SAFES, similar to some static analysis tools [2, 3, 5] which require a set of criteria or rules as input (or annotating in source programs), also needs a user specification by the developers (or testers) to specify the security-sensitive fields in the CUA.

Through the three analyzers sequentially, a security result is reported by SAFES for security analysts to review the analyzed programs and fix the security problems. In terms of the three levels of security classification presented above, the security report details each security-sensitive field about its escapability and mutability as well as the causes and traces to a certain degree.

4.2 Analysis Engines

The basic analysis engine, Escape Analyzer, performs the escape analysis proposed in our work [15] relying on as inputs points-to and dependence information, which are computed from points-to and dependence sub-analyzer, respectively. The escape tuples output by Escape Analyzer imply the escape type, e.g., `Reference-`, `Data-`, `Control-escape` proposed in [15], and simple escape trace for the security-sensitive fields in the CUA.

Mutability Analyzer takes as input escape information from the preceding Escape Analyzer and implements the proposed mutability analysis algorithm for computing mutability results of the fields (and classes) concerned with the security in the CUA. Both a general iterative implementation and an optimized Graph-based implementation (similar to topological sort solution) for the mutability analysis algorithm are developed to evaluate the efficiency of our technique, as experimented in the evaluation later.

The final analysis engine, Security Analyzer, concludes the static analysis for field security in Java components with the analysis results from the prior two analyzers. It implements the algorithm presented in Figure 3 and determines the security level of a given security-sensitive field in the CUA based on the escape and mutability information. Besides the security levels, the security result reported from Security Analyzer shows the causes of the security problems and a few traces to the root in the analyzed codes. With the security result in hand, security analysts can review the source codes and fix the security bugs displayed in the result.

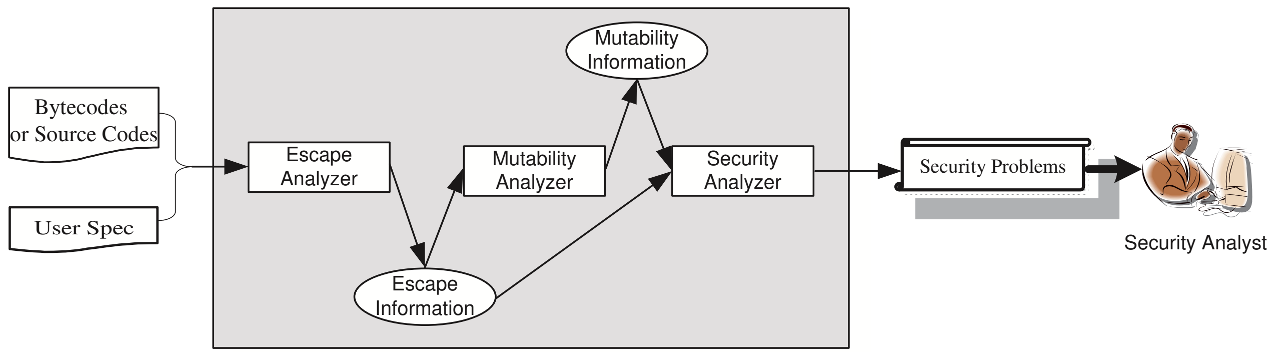


Figure 4. SAFES Architecture

5 Evaluation

5.1 Experiment Setup

The applications are chosen as our benchmarks as follows: `OrderApp` [11], `CustomerApp` [11], `DukesBank` [7], `RMS` [6] and `PetStore` [8]. For each application, Table 1 presents:

- The overall size of the EJB bytecodes analyzed in the corresponding J2EE applications in Column 2.
- The total number of the class files analyzed (including the loaded library classes) and the number of class files defined in the EJB components in Columns 3 and 4, respectively.
- The total number of reachable methods in each application's call graph and the number of methods declared in the EJB components in Columns 5 and 6, respectively.
- The number of reference type and primitive type fields declared in the EJB components in the last two columns.

We used a Pentium 4 3.06 GHz Dell Dimension machine with 1.43 GB RAM and Windows XP Professional Version 2002, SP2. We ran the experiments inside Eclipse V3.1.1, using Sun JDK 1.5.0_05.

5.2 Comparison of Escape Analysis Implementations

We employ two points-to analysis implementations with a simple Soot-based implementation for dependence analysis to evaluate the precision of our tool. Our points-to analysis is implemented with Spark framework of Soot. In the experiments, two implementation options we chose are `ot-otf-fs` and `ot-cha-fb` with the hybrid set implementation, which are detailed in [13]. `ot` means respecting declared types, `otf` and `cha` represent the call graph construction through on-the-fly and CHA-based, respectively. `fs` and `fb` describe field-sensitive and field-based, respectively.

Table 2 shows the experimental results with our default user specification, which conservatively treats all fields in the CUA as security-sensitive. Columns 5, 6 and 7, labeled as `M(FP)`, `V(FP)` and `S(FP)`, report the number of fields in the corresponding security level and the number of false positives in the parentheses, where `M, V, S` mean the security level of `Modifiable`, `Visible` and `Safe`, respectively. False positives are the fields that are mistakenly classified as the corresponding security level. In our study, the false positives are manually checked with a lot of reviews.

In all cases, `SAFES` reports few false positives. Between the two points-to implementation options, `ot-otf-fs` shows better precision, which corresponds to the report in [13]. According to Column 3 of Table 2, the combination with `ot-otf-fs` as points-to analysis runs faster for each benchmark since on-the-fly call graph construction reaches much fewer methods compared to CHA-based one. The memory usage of our experiments takes on the same order as time cost. As a result, on the experimental results reported in Table 2, `ot-otf-fs` is the better points-to analysis implementation with higher precision and performance.

5.3 Comparison of Mutability Analysis Implementations

We evaluated the performance of mutability analysis with two implementations, such as general iterative implementation and optimized graph-based implementation. The details for each algorithm implementation are presented in

Name	Size (KB)	#Classes Total	#Classes App.	#Methods Total	#Methods App.	#Fields Ref.	#Fields Pri.
OrderApp	43	4988	37	5286	260	32	14
CustomerApp	44	4988	35	5341	213	32	10
DukesBank	53	4976	39	5485	314	48	1
RMS	58	4987	23	5479	158	83	7
PetStore	231	5306	121	6555	1006	343	32

Table 1. Benchmark Characteristics

Benchmark	PT Analysis	Time (Sec.)	Mem (MB)	M(FP)	V(FP)	S(FP)
OrderApp						
	ot-otf-fs	130	354	26(0)	12(1)	8(0)
	ot-cha-fb	440	903	27(1)	12(1)	7(0)
CustomerApp						
	ot-otf-fs	131	335	27(0)	11(0)	4(0)
	ot-cha-fb	380	856	27(0)	12(1)	3(0)
DukesBank						
	ot-otf-fs	124	309	36 (1)	7(0)	6(0)
	ot-cha-fb	445	899	36 (1)	10(3)	3(0)
RMS						
	ot-otf-fs	99	300	78(0)	10(0)	2(0)
	ot-cha-fb	397	870	78(0)	10(0)	2(0)
PetStore						
	ot-otf-fs	476	521	95(3)	239(2)	41(0)
	ot-cha-fb	1160	989	95(3)	245(8)	35(0)

Table 2. Experimental results of comparing the escape analysis implementations

our earlier work [15]. Both implementations have identical precision in the sense that they compute the same mutability information for each application. For each benchmark, we ran each algorithm 3 times and averaged the run time and memory cost for each version. The experimental results are reported in Table 3.

As for the running time, the graph-based implementation ran much faster than iterative implementation on the benchmarks examined, which proves that a nice order of scan is the key to improve the performance of algorithm. Otherwise, the iterative implementation consumed a little more memory than the graph-based one.

5.4 Discussion

In our study, the precision of our analysis result is sensitive to the precision of underlying analysis engines, especially, Points-to analysis. Comparing the experimental results in Table 2, `ot-otf-fs` points-to analysis reports the best precision and performance on our benchmark applications. In contrast, excessively conservative treatments in `ot-cha-fb` points-to analysis produces more false positives as illustrated.

The experiments in mutability analysis definitely prove that our optimized graph-based implementation ran faster than general iterative implementation. Because of the interdependence between field mutability and class mutability, the execution order of classes is critical to performance of the mutability analysis algorithm. We map the optimal order solution problem in the mutability analysis to a graph problem, which is similar to a topological sort but graphs with cycles may be considered.

At first glance, graph-based implementation needs to keep a graph in memory and thus seems to need more execution memory. Conversely, our study shows that it ran with a little less memory. We believe that this can be explained by the fact that our graph-based algorithm deletes nodes and edges of the graph representation as analysis proceeds, which makes some objects unreachable and frees memory by Java garbage collection mechanism constantly.

Our combined static analysis approach is heavily dependent on escape analysis, whose information result is in turn input into consecutive mutability analyzer as depicted in Figure 4. In our evaluation, escape analyzer costs much more time than other analyzers (e.g., mutability analyzer, or security analyzer) in each execuation. It is not surprising because the exhaustive field-sensitive Andersen's points-to analysis from Spark framework in our tool is much more

Benchmark	Iterative		Optimized	
	Time (msec)	Mem (Mb)	Time(msec)	Mem (Mb)
OrderApp	620	380	578	373
CustomerApp	766	350	750	343
DukesBank	578	328	531	310
RMS	1438	323	1031	322
PetStore	3619	552	2001	528

Table 3. Comparison between two kinds of mutability analysis implementations

complicated, as compared to other analyzer engines. Recent demand-driven points-to analysis [16] might be promising to efficiently meet the time challenges.

6 Related Work

Static analysis tools and specialized checkers such as FindBugs [3] report useful information about code violations. However, the type of violations are largely limited to errors that do not require detailed program analysis. ESC/Java2, the extended static checker for Java version 2, attempts to find common run-time errors by program verification techniques and its formal annotations. Unlike SAFES, ESC/Java2 needs programmers' assistance to annotate their programs with specially formatted comments called pragmas, and then verifies the correctness of the JML(Java Modeling Language)-annotated Java programs. JLint [4] analyzes Java bytecodes and performs syntactic checks and data flow analysis for finding bugs. Also, JLint builds the lock graph to find inconsistencies and synchronization problems. PMD [5] performs syntactic checks on Java source codes to look for potential problems such as dead code and duplicate code. These bugs from PMD are largely due to stylistic conventions, some of which are possibly bad programming style, but not real bugs. Unlike other bug-finding tools mentioned above, Bandera [12] is a verification toolset, which includes optional slicing and abstraction phases, followed by model checking tools.

7 Conclusion

In this paper, we have presented SAFES, a static security analysis tool, which automatically detects field security problems in Java components. SAFES combines escape and mutability analysis techniques to find two categories of vulnerabilities, i.e., information leak and modification access violation. In our study, SAFES can correctly identify potential security problems on our benchmark applications with few false positives, making our tool useful for real-world applications. In our future work, we will focus on helping to fix the security problems found by the analysis tool, and developing the guidelines for developers to fix them.

Acknowledgments

We want to thank Phyllis Frankl, David Chays and the anonymous reviewers for commenting on an earlier version.

References

[1] The apache xalan project. *http://xalan.apache.org/*.

[2] ESC/Java2-the extended static checker for Java version 2. *http://secure.ucd.ie/products/opensource/ESCJava2/*.

[3] Findbugs-find bugs in Java programs. *http://findbugs.sourceforge.net/*.

[4] Jlint. *http://artho.com/jlint/*.

[5] PMD. *http://pmd.sourceforge.net/*.

[6] Sun microsystems industry and partners engineering team, resource management system. *http://java.sun.com/developer/technicalArticles/J2EE/rms/*.

[7] Sun microsystems, J2EE 1.4 tutorial. *http://java.sun.com/j2ee/1.4/download.html#tutorial/*.

[8] Sun microsystems, Java petstore. *http://java.sun.com/developer/releases/petstore/*.

[9] Sun alert id: 54760. *http://sunsolve.sun.com/search/document.do?assetkey=1-26-54760-1*, 2003.

[10] Sun alert id: 57613. *http://sunsolve.sun.com/search/document.do?assetkey=1-26-57613-1*, 2004.

[11] G. Anderson and P. Anderson. Enterprise Javabeans component architecture: Designing and coding enterprise applications. *Sun Microsystems Press Series*.

[12] J. C. Corbett, M. B. Dwyer, J. Hatcliff, S. Laubach, C. S. Pasareanu, Robby, and H. Zheng. Bandera: Extracting finite-state models from java source code. *Proceeding of the 22nd International Conference on Software Engineering*, pages 439–448, June 2000.

[13] O. Lhoták. Spark: a flexible points-to analysis framework for Java. *Master's thesis*, McGill University, Dec 2002.

[14] S. Porat, M. Biberstein, L. Koved, and B. Mendelson. Automatic detection of immutable fields in Java. *Proceedings of IBM Centre for Advanced Studies Conference (CASCON)*, page 10, 2000.

[15] A. Shi and G. Naumovich. Static analysis of computing escapability and mutability for Java components. *Fifth IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'05)*, Sep 2005.

[16] M. Sridharan, D. Gopan, L. Shan, and R. Bodik. Demand-driven points-to analysis for Java. *OOPSLA'05*, Oct 2005.

Reuse of Database Access Layer Components in JEE Product Lines: Limitations and a Possible Solution (Case Study)

Ding Peng, Stan Jarzabek, Damith C. Rajapakse
Department of Computer Science
School of Computing,
National University of Singapore
ding_peng@alumni.nus.edu.sg;
stan@comp.nus.edu.sg;damith@comp.nus.edu.sg

Hongyu Zhang
Tsinghua University
China
hongyu@mail.tsinghua.edu.cn

Abstract

We set up an experiment to evaluate JEE as a platform for product line development. While JEE provides many useful mechanisms for reuse of common services/components, still we found that systematic across-the-board reuse in application domain-specific areas was hard. The main difficulty was the lack of a mechanism to represent groups of similar components in a generic, adaptable form. Such similar components arise as the number of variant features of a product line grows, and we need to accommodate legal combinations of variant features in components of a product line architecture. Such uncontrolled growth of similar component versions hinders productivity of reuse-based development and raises maintenance costs. In the paper, we study the manifestation of this problem in the JEE™ database access layer. Interactive Development Environments such as NetBeans or JBuilder speed up the development process, but they do not address the source of the problem, which is the lack of mechanisms to design generic components capable of accommodating variant features in various combinations. We filled this gap with a "mixed strategy" solution based on generative programming technique of XVCL applied on top of JEE. In the paper, we highlight the nature of the problems we encountered and our solution.

1. Introduction

JEE™(Java Enterprise Edition) is a widely accepted standard and platform for Java based enterprise applications. Rapid application development, reuse and flexible deployment of applications are crucial in today's enterprises. Component platforms such as JEE help industries to achieve these qualities. Another important industry trend is the product line approach, whereby productivity is boosted by systematic reuse across a family of similar software systems. Because of much synergy in targets, JEE, already widely accepted in industries, is a viable option to consider for product lines.

In this paper, we describe an experiment in which we explored this option. We worked with a product line that we developed in an earlier project – a Computer-Aided Dispatch (CAD) system family [1]. We completely re-designed the previously developed product line architecture to take best advantage of the JEE platform and its reuse mechanisms. With understanding of the issues specific to the new platform, we added some new variant features[2] of CAD product line to facilitate better evaluation. We included JBuilder™ into the toolbox for the project.

We have positive and negative findings. JEE offers many useful mechanisms for reuse in the area of common services for component-based systems, such as component communication and management of components' lifetime. Normally, a developer would have to switch his attention between supporting such services and writing the actual application code. There would be much repetition of similar service code across components and mutual cross-cutting of the application code and service code. JEE component infrastructure avoids this problem in which it is very supportive to product line goals.

However, we found little support for systematic reuse in application domain-specific areas. For product lines whose members differ in functional features, such as our CAD systems, this is certainly a problem that needs to be solved. In an earlier paper, we discussed the problem and solution in the JEE presentation and business logic system layers [12]. In this paper, we illustrate problems encountered in the database access layer (DAL) and analyze their sources. We outline a solution based on a "mixed strategy" approach in which we enhance JEE with generic design capabilities of XVCL (XML-based Variant Configuration Language) [3]. The JEE/XVCL solution allows us to represent DAL components as generic, adaptable meta-components. Meta-components

[1] A project in collaboration with ST Electronics Pte Ltd., under the Singapore-Ontario joint research project.
[2] Variant features are product line requirements that vary across product line members.
[3] http://xvcl.comp.nus.edu.sg

can be customized to produce specific DAL components accommodating any legal combination of variant features. Specifications of such customizations are in both human- and machine-readable form. These mechanisms are sufficient to keep the number and complexity of DAL components under control in the product line situation.

2. Brief introduction to JEE

JEE simplifies enterprise application development by providing a standardized architecture that helps developers reuse common service components. JEE takes advantage of many features of the Java 2 Platform, Standard Edition, such as "Write Once, Run Anywhere^{TM}" portability, JDBC^{TM} API for database access, and a security model that protects data even in internet applications. Building on this base, JEE adds full support for Enterprise JavaBeans^{TM} (EJB) components, Java Servlets API, JavaServer Pages^{TM} and XML technology. Tools such as NetBeans and JBuidler provide visual development environment and template-like mechanism for automatic generation of JEE/EJB components. JBuilder can be integrated with application server such as BEA Weblogic, Tomcat, etc.

3. CAD product line

We briefly describe the domain of our experimentation: Computer Aided Dispatch (CAD for short) systems are mission-critical systems that are used by police, fire & rescue, health service, port operations and others. Figure 1 depicts a basic operational scenario and roles of a CAD system for Police.

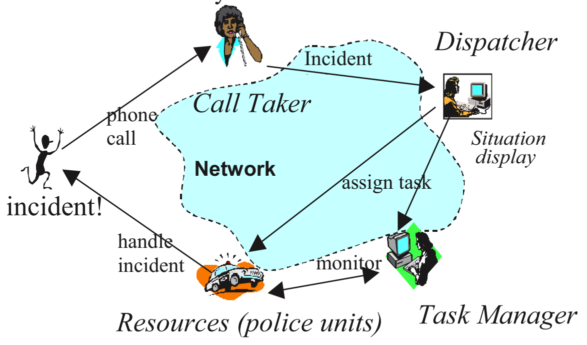


Figure 1. A CAD system for police

Once a *Caller* reports an incident, a *Call Taker* captures the details about the incident and the *Caller*, and creates a *Task* for this incident. The *Dispatcher* selects suitable *Resources* for un-dispatched *Tasks* and dispatches the *Resources* to execute the *Task*. The *Resources* execute the instructions given and reports to the *Task Manager*. The *Task Manager* actively monitors and updates the *Tasks* until the case is closed. The *Task Manager* closes a *Task* when there are no more activities associated with the *Task* in the context of CAD system.

At the basic operational level, all CAD systems are similar. However, the specific context of operation results in many variations on the basic operational scheme. We depict general CAD variant features in Figure 2 and JEE data access layer (DAL) variant features in Figure 3.

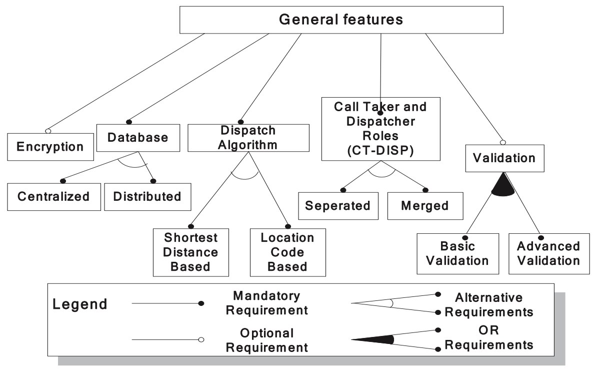


Figure 2. General CAD features

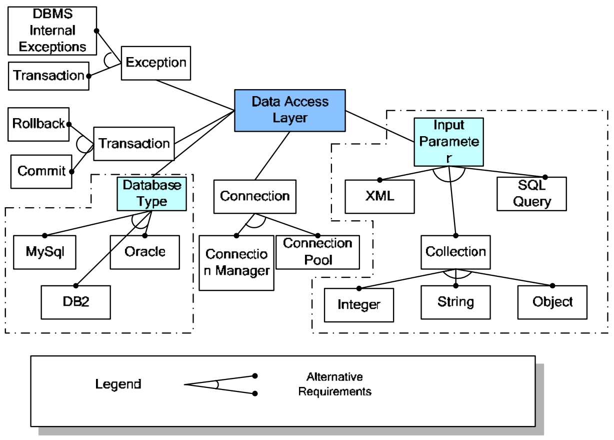


Figure 3. Data access layer (DAL) features

4. CAD product line with JEE

We developed the CAD product line architecture (CAD-PLA, for short) on JEE according to a blueprint of a product line lifecycle developed in our earlier projects [13][14], as follows: Having analyzed the CAD domain, we scoped the product line by selecting variant features for the project. Then, we defined a default CAD system – a typical system in CAD domain. We designed logical runtime architecture for the default CAD on JEE. We designed CAD-PLA and its components for ease of customization in the view of variants selected in domain analysis. We implemented CAD-PLA as a collection of component configurations [6] within a stable architecture [5]. We explored EJB/JEE platform and tools (JBuilder) to maximize reuse.

4.1. Domain analysis and a default CAD system

We used UML models such as use case, class, sequence, activity and deployment diagrams, with provisions for modeling variant features [8], to model CAD domain. *Task* (Figure 4) is a central concept in CAD systems: A *Task* is created as soon as the *Call Taker* has received the information about the incident. The *Task* is closed once the handling of the incident has been completed. The *Task*'s lifecycle spans major CAD components and all the major roles interact by means of a *Task*. We defined a default CAD system around the *Task* concept, as a natural bridge to a generic CAD-PLA.

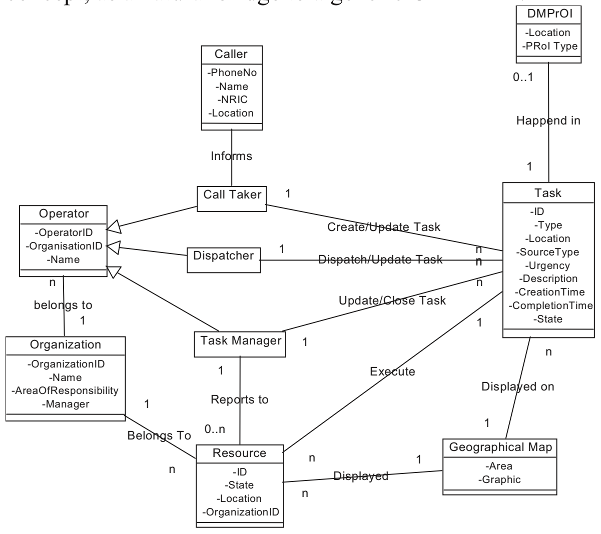


Figure 4. CAD class diagram

4.2. A JEE CAD-PLA

We skip the intermediate step of building a runtime architecture for the default CAD system and describe the final, refined-for-reuse, JEE CAD-PLA. CAD-PLA was based on a standard five-layer JEE model (Figure 5).

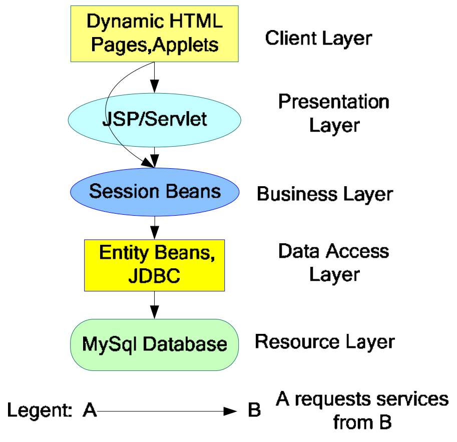


Figure 5. The five-layer JEE architecture

Presentation layer components used JSP to generate and format responses to the clients. Enterprise beans encapsulated the application logic on the server side. A Session Bean represented an activity (or process) operating on business objects and an Entity Bean provided a persistent storage mechanism for business objects. Many business objects corresponded to classes in a class diagram of Figure 4. There are two types of persistence of Entity Beans, namely bean-managed (BMP) and container-managed (CMP) persistence. With BMP, a developer writes the Entity Bean code to access a database. With CMP, the EJB container automatically generates the necessary database access calls. We applied both BMP and CMP in our system. Typically, each Entity Bean has a corresponding table in a relational database where bean instances are stored. We used MySQL 4.0 as the Database server.

One of the main performance problems associated with EJBs is the amount of network traffic needed to access attributes of the bean. The standard solution for alleviating the network delays is to return all the attributes in one method call, using a simple data-holder class. In JEE, this is achieved by the Value Object Pattern [5] and we applied it in our architecture. To achieve database independence, we applied the Data Access Object (DAO) Pattern which separates Entity Beans from low-level database access code. We designed a generic evaluator component to further separate DAO details from the query generation and evaluation code. We also used *MVC*, *Template*, *Composite* and many other recommended general and JEE-specific design patterns.

5. Evaluation of the JEE DAL for CAD-PL

The five-layer architecture model has several advantages. Firstly, rather than building isolated components, it allows us to build modules comprising a collection of client and server components, forming meaningful parts of an application. These components communicate through standardized, abstract interfaces, and when combined together form a complete application. Sometimes, change can be conveniently propagated from one bean to other related beans. Suppose we change *Task* bean from basic validation to advanced validation. *Create Task* process and *Dispatch Task* process beans will change their behavior automatically to reflect advanced validation.

Tools such as Jbuilder™ provide templates to facilitate auto generation of EJB codes, which speeds up development. For example, to create a *Task* bean, a developer typically enters attributes such as *Task* name, or *Task* location into a template, and the tool creates methods such as findByPrimaryKey, setTaskName, getTaskName, setLocation, getLocation, etc. automatically. Reusability is achieved due to EJB standard specification.

The key issue in product line (PL) approach is to customize components of a product line architecture (PLA) to accommodate variant features required in the specific system, PL member, that we wish to build. The ease of selecting PLA component configurations and their customization determines the economic benefit of the PL approach [6]. Therefore, we analyzed the impact of variant features on CAD domain model, and JEE CAD-PLA components. We found that variants had a profound impact on CAD. Understanding the impact of variants on a domain model is a good starting point for tracing the impact of variants on PLA and components' code. For example, *Caller/Task* validation method and *Call Taker* and *Dispatch* roles affected use cases such as *Create Task*, *View Task*, *Create Caller*, and many others. Class and sequence diagrams were also affected by these variants. As UML provisions for modeling variant features are limited, we used extensions described in [8] to model CAD product line.

At times, the impact of variants on the PLA and components' code was quite drastic, and required us to change allocation of functions across components, component communication patterns, and/or component interfaces. Certain variants (or their combinations) required us to include/exclude certain components to/from a PLA. We say that a variant has *architectural (or non-local) impact* on the PLA, if the chain of customizations triggered by the variant spreads through the system rather than being confined to a small number of components. Variants that affected multiple CAD models most often also had such architectural, non-local impact on JEE CAD-PLA.

In our case study, most of the general variants had architectural, non-local impact on CAD. For example, an option under one variant is that *Dispatcher* and *Call Taker* roles are played by the same person. In such a case, components related to *Call Taker* and *Dispatch Task*, both user interface and business logic, must be re-formulated and re-designed into single components to reflect new requirements for system operation. In most cases, addressing a variant also requires us to modify component implementation.

We have discussed handling general CAD variant features in a CAD-PLA in other papers [12][13][14]. In this paper, we focus on the database access layer (DAL) features (Figure 3). Changes of a database and database access codes may have profound impact on application-level CAD components and vice versa. JEE uses DAO pattern to shield application-level code from possible changes in the database. DAL components, built around DAO, receive database requests (queries) from the business logic and presentation layers. DAL receives result of the query first and then passes it to the caller in the required format which is either input parameter or output parameter.

CAD-PLA must cater for many CAD systems that differ in general features (Figure 2). General variants often affect the database structure, database access codes. The choice of general features of CAD triggers the need for specific variant features that affect DAL components (Figure 3). Table 1 shows some of the Entity Bean DAL components (listed in the first column) affected by variant features (listed in the first row).

Table 1. DAL components

	DB Type	Connection Pool	Input Parameter	Bean Type
Caller Bean	X	X	X	X
Task Bean	X	X	X	X
Organization Bean	X	X	X	X
Area Bean	X	X	X	X
Resource Bean	X	X	X	X
Location Bean	X	X	X	X
Command Bean	X	X	X	X

Any member of CAD product line needs Entity Beans implementing some combination of features, as shown in Figure 6. Strictly speaking, each path from the top to the bottom in Figure 6 denotes a unique Entity Bean, that may be needed in some CAD system. Suppose we address four different databases, then we have four choices at Level 1. We have two choices at Level 2; seven choices at Level 3; and two choices at Level 4. Therefore, we may have as many as 112 possible Entity Beans that arise from CAD variant features. Here, our analysis covers only a rather simplified situation. In reality, transaction attribute, exception handling and many other variants need be taken into account which is likely to result in thousands possible Entity Beans.

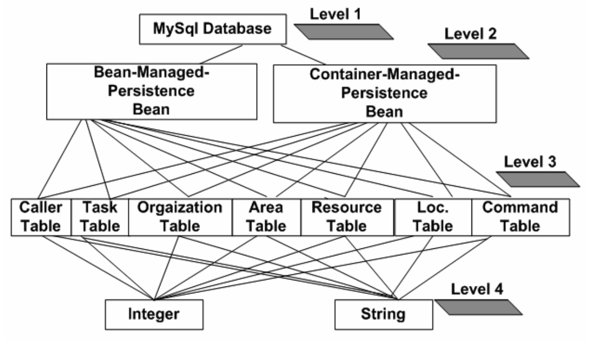


Figure 6. Features affecting DAL components

Entity Beans implementing various combinations of variant features are similar to each other, in their interfaces, the way they represent underlying tables, and in implementation details. Consider *Task* bean and *Caller* bean. *Caller* bean needs the caller phone information, location information etc. *Task* bean needs the task creation date, task area information. The classes implementing the two beans are the same with the exception of attribute names/types, and small differences in algorithmic details. Unfortunately, JEE does not provide any mechanism to abstract this kind of similarity pattern into a generic component for reuse.

Component explosion problem affects all the software layers, not only DAL. We observed the same situation in JEE presentation and business logic component layers of JEE Web Portals [12]. The problem is not limited to JEE. Other OO and component-based technologies lack strong enough mechanisms to address the problem at the root level, or even to deal with its symptoms. Studies indicate that as many as ten thousand versions of a component may arise during evolution of industrial product lines [6].

6. JEE/XVCL "mixed strategy" solution

"Mixed strategy" approach avoids the problem of component explosion by means of generic design: Whenever we observe enough similarity among a group of components affected by variant features, rather than multiplying component versions, we apply generative programming technique of XVCL to build a generic, adaptable meta-component. Along with a generic component, we keep specifications of how to produce concrete components for required combinations of variant features. Such specifications can be examined by a developer, and also executed by the XVCL Processor to produce custom components on demand.

We described XVCL mechanisms in other papers, so here we only outline the general structure of the solution.

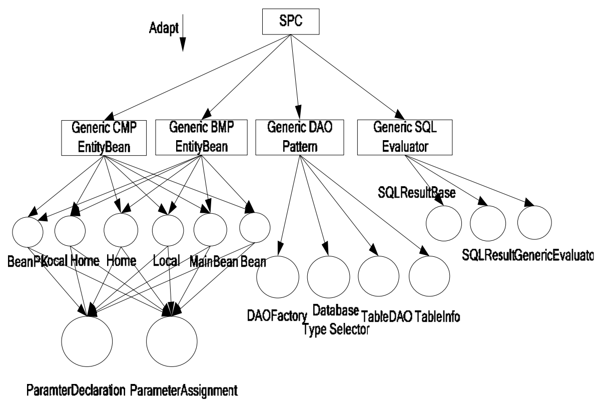


Figure 7. JEE/XVCL CAD-PL

Our JEE/XVCL "mixed strategy" solution addressed four groups of similar components for which we built generic meta-components. They are shown in rectangle boxes of Figure 7 as CMP Entity Beans, BMP Entity Beans, DAO Pattern, and SQL Evaluator. Below the rectangle boxes, there are generic building blocks from which the XVCL Processor could synthesize, after possible customizations, concrete classes or Entity Beans. The top-most meta-component, called SPC, contained specifications describing how to customize lower level meta-components to produce concrete classes or Entity beans for any required combination of features.

7. Evaluation of results

In the default CAD built on JEE, DAL included 42 Entity Bean classes and 19 DAO classes, comprising the total of 5811 LOC (without blank lines and comments). The JEE/XVCL "mixed strategy" solution for DAL consisted of 20 meta-components, comprising the total of 2354 LOC, counting both JEE and XVCL code. These 20 meta-components were generic and adaptable, capable of accommodating anticipated range of variant features.

The reduction of physical and conceptual size of the solution had a positive impact on implementing changes. We conducted an experiment to compare the effort involved in changing DAL components in each of the JEE and JEE/XVCL representations. We implemented new combinations of variant features in both representations. We considered variants *DB Type*, *Connection Pool*, *Input Parameter* and *Bean Type*. We measured the effort in terms of the number of modification points, modified classes, and total number of lines of code (LOC) in Entity Beans that we needed modify to accommodate the change.

In Table 2 and Table 3, we show the number of modification points in DAL in each of the JEE and JEE/XVCL "mixed strategy" solutions. Modification points are either in class attributes (A), or methods (M) in each of the 42 classes. X stands for DB Type, Y – for Connection Pool, Z – for Input Parameter, and P – for Bean Type.

Table 2. Modification points (JEE)

	Entity Bean		DAO		Value Object		Total
	A	M	A	M	A	M	
X	0	0	2	38	0	0	40
Y	0	0	3	38	0	0	41
Z	7	56	7	14	14	21	119
P	24	20	4	38	11	24	122

Table 3. Modification points (JEE/XVCL)

	Entity Bean		DAO		Value Object		Total
	A	M	A	M	A	M	
X	0	0	1	3	0	0	4
Y	0	0	1	4	0	0	5
Z	1	8	1	4	1	3	18
P	3	4	1	6	1	4	19

8. Conclusions

In the product line situation, variant features need be implemented into software components, resulting in an explosion of component versions. In the paper, we focused on the problem of explosion of component versions in the JEE database access layer (DAL), caused by feature combinations.

In JEE solution, many similar components had to be implemented. Despite striking similarities, groups of similar components had to be repeatedly implemented. We addressed the problem with a generative technique of XVCL. The JEE/XVCL "mixed strategy" solution could represent each of the groups of similar DAL components in unique, generic, adaptable form. Not only did JEE/XVCL representation collapse the physical and conceptual size of DAL representation, but it also reduced the effort to implement changes to DAL.

While the "mixed strategy" solution solves the problem of component version explosion and makes reuse easier, it is not without pitfalls. It is easier to understand a concrete program than a meta-program. It is also difficult to validate a meta-program as we can derive many concrete programs from it. The current form of XVCL is an assembly language for generic design via parameterization. XVCL's explicit and direct articulation is the source of its expressive power, but it also adds a certain amount of complexity to the problem. However, benefits of being able to deal with issues of genericity at the meta-level plane seem to outweigh the cost of the added complexity. These benefits include ease of reuse with adaptation, the overall reduction of conceptual complexity and size of the solution, improved traceability and changeability. We are currently collecting empirical and analytical evidence to support the above hypothesis. Our lab studies and two projects by our industry partner ST Electronics confirm our expectations [9].

In the future work, we plan to extend empirical studies on JEE and .NET platforms to further substantiate the results reported in this paper. We plan to study the impact of pattern-driven development on reuse in the application domain-specific areas of component-based systems. We believe there may be good opportunities for "mixed strategy" to improve productivity in application of patterns and in maintaining software developed in pattern-driven way. Finally, we plan to work on tools helping in detecting similarity patterns in legacy code, for possible re-engineering into "mixed strategy" solutions [2].

References

[1] Bassett, P. Framing software reuse - lessons from real world, Yourdon Press, Prentice Hall, 1997

[2] Basit, A.H. and Jarzabek, S. "Detecting Higher-level Similarity Patterns in Programs," *ESEC-FSE'05, European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering*, ACM Press, September 2005, Lisbon, pp. 156-165

[3] Basit, H.A., Rajapakse, D.C., and Jarzabek, S. "Beyond Generics: Meta-Level Parameterization For Effective Generic Programming," *Proc. 17th Int. Conf. on Software Engineering and Knowledge Engineering*, *SEKE'05*, Taipei, July 2005

[4] Basit, H.A., Rajapakse, D.C., and Jarzabek, S. "Beyond Templates: a Study of Clones in the STL and Some General Implications," *Int. Conf. Software Engineering, ICSE'05*, St. Louis, May 2005, pp. 451-459

[5] Clements, P. and Northrop, L. *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2002

[6] Deelstra, S., Sinnema, M. and Bosch, J. "Experiences in Software Product Families: Problems and Issues during Product Derivation," *Proc. Software Product Lines Conference*, *SPLC3*, Boston, Aug. 2004, LNCS 3154, Springer-Verlag, pp. 165-182

[7] Jarzabek, S. and Li, S. "Eliminating Redundancies with a "Composition with Adaptation" Meta-programming Technique," *Proc. ESEC-FSE'03, European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering*, ACM Press, September 2003, Helsinki, pp. 237-246; paper received ACM Distinguished Paper award

[8] Jarzabek, S. and Zhang, H. "XML-based Method and Tool for Handling Variant Requirements in Domain Models", *Proc. 5th International Symposium on Requirements Engineering, RE'01*, August 2001, Toronto, Canada, pp. 166-173

[9] Pettersson, U., and Jarzabek, S. "An Industrial Application of a Reuse Technique to a Web Portal Product Line," accepted for *ESEC-FSE'05, European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering*, ACM Press, September 2005, Lisbon, pp. 326-335

[10]Wong, T.W., Jarzabek, S., Myat Swe, S., Shen, R. and Zhang, H.Y. "XML Implementation of Frame Processor," *Proc. ACM Symposium on Software Reusability, SSR'01,* Toronto, Canada, May 2001, pp. 164-172

[11]Introducing Enterprise Java Beans, Justin Couch, Daniel H. Stenberg. Java™ 2 Enterprise Edition Bible pp. 818-830

[12]Yang, J. and Jarzabek, S. "Applying a Generative Technique for Enhanced Reuse on JEE Platform," *4th Int. Conf. on Generative Prog. and Component Eng., GPCE'05*, Sep 29 - Oct 1, 2005, Tallinn, Estonia, pp. pp. 237-255

[13]Zhang, H. and Jarzabek, S "An XVCL-based Approach to Software Product Line Development," *Conf. on Software Engineering and Knowledge Engineering,SEKE'03,* San Francisco, July 2003, pp. 267-275

[14]Zhang, H. and Jarzabek, S. "A Mechanism for Handling Variants in Software Product Lines," special issue on Software Variability Management of Elsevier's journal *Science of Computer Programming,* Volume 53, Issue 3, Dec. 2004, pp. 255-436

Design of Wrapper for Self-Management of COTS Components

Michael E. Shin
Department of Computer Science
Texas Tech University
Lubbock, TX 79409-3104
(806) 742-3527
Michael.Shin@ttu.edu

Fernando Paniagua
Department of Computer Science
Texas Tech University
Lubbock, TX 79409-3104
(806) 742-3527
Fernando.Paniagua@ttu.edu

Abstract

This paper describes an approach to the design of wrapper for self-managing COTS (commercial off-the-shelf) components. Each wrapper for COTS components encapsulates the properties of self-management – detection, reconfiguration, and repair. A COTS component deals with application perspectives, whereas the wrapper handles self-management perspectives, separately from the application perspectives. Each wrapper for self-managing COTS components is structured into several objects in support of detection, reconfiguration, and repair of the anomalous COTS component. The approach suggested in this paper is applied to the distributed elevator system consisting of COTS components.

1. Introduction

COTS (commercial off-the-shelf) component-based software technologies for concurrent and distributed systems have been enhanced for a long time. Nowadays, more and more commercial, industrial, military, medical, and consumer application systems are developed using COTS components, instead of developing from scratch. However, the COTS-component based systems may still have design faults or come across unanticipated events resulting in system failures. Some software system failures result in loss of business and, at the worst, loss of human life.

Traditionally, software fault-tolerant approaches [Kopetz94, Torres-Pomales00, Kim00, Guerra02] have been used in concurrent and distributed systems to ensure that software faults do not cause system failures. The mechanisms used in the approaches rely on software diversity or redundancy; however, it may not be practical for two or more different development teams to design and implement (separately) various versions of a single software system using the same specification, whereas replicates of a software system can result in the same system failures under the same execution conditions.

As an alternative, self-management (or self-healing) [Koopman03, IMB03, Dashofy02, Garlan02, Garlan03, Shin06] of COTS component-based concurrent and distributed systems is considered in which anomalies of systems (i.e., design faults or unanticipated events) are autonomously detected and repaired at runtime by a self-managing mechanism. For this, COTS components may need to be changed to support the properties of self-management in a system, but the change to COTS components are not always possible due to COTS vendor's protecting intellectual property.

This paper describes an approach to the design of a wrapper [DeLine99] for self-managing COTS components. Self-managed COTS-components need to be capable of detection of anomalies in COTS components, reconfiguration, and repair of anomalous COTS components. A wrapper for self-managing COTS components is structured into several objects in support of the self-management mechanism.

The approach described in this paper is applicable for soft real-time COTS component-based concurrent and distributed application systems, rather than hard real-time systems that should provide each service within its tight deadline. This is because the time required for repairing anomalous objects may go beyond the deadline specified for a service in hard real-time systems. An earlier paper [Shin06] describes the software architecture for self-managed COTS component-based systems. Other previous work [Shin05a, Shin05b] describes the design of reliable software components using the internal structure of components.

2. Related work

Related work addresses approaches to making COTS components more reliable. The architecture of mediators and obligations [Thomas00] makes it possible to build dependable systems including COTS components. Mediators are software entities that detect faults by monitoring interactions between client and server components, enforcing client and server obligations. A mediator for a COTS component can provide multiple functions including fault management, acceptance tests for inputs and outputs, and system monitoring. A mediator for COTS components is designed for client/server systems.

A fault tolerant COTS component [Guerra03, Anderson03, Guerra04] has been proposed by adding protective wrappers to COTS components to design dependable software architecture. A fault tolerant COTS component is structured into normal activity and abnormal activity. A COTS component is encapsulated in the normal activity, which detects errors using upper and lower detectors. The errors detected by the normal activity are resolved by error handlers in the abnormal activity.

Protective wrappers for COTS components can also mediate the input to and output from COTS components to make the system more dependable [Meulen05]. The specifications of inputs to and outputs from COTS components are defined as preconditions and postconditions, which are used by protective wrappers to detect the outbound values of specifications for the input and output. The wrappers adjust the outbound values to reasonable values in order to avoid behavior undefined for COTS components or the system.

Although several approaches have been suggested for making COTS components reliable, these approaches support partially the requirements of wrapper for self-managing COTS components.

3. Requirements of wrapper for self-managing COTS components

A wrapper for self-managing COTS components needs to encapsulate a COTS component so as to control the interaction between COTS components. In the design of a wrapper for self-managing COTS components, the following are required:

(1) Detection. Each wrapper for COTS components needs to be capable of monitoring a COTS component to detect anomalies that may result in system failures. A COTS component may contain malfunctions that can be caused by such anomalies in the component as design errors, unfair resource allocation, or inappropriate controlling of concurrency among threads in the component. These anomalies within a COTS component can make the systems unreliable and unavailable.

(2) Reconfiguration. COTS components need to be reconfigured at run-time by their wrappers to minimize the impact from the anomalies detected. An anomalous component needs to be isolated from other healthy components in the system. As the anomalous component is back to normal after the repair, wrappers also need to reconfigure the system to provide full functionality of the system.

(3) Repair. The wrappers of COTS components need to repair anomalies in the COTS components. Some anomalies in a COTS component are critical to the whole system, which may lead to system failures. The repair of anomalies in COTS components is to protect the system from system failures.

In addition, each COTS component needs to be used as sold by COTS component vendors. That is, a COTS component is considered a black box whose internal details are hidden except for its original interface to services the component provides and requires.

4. Design of wrapper for self-managing COTS components

A wrapper for self-managing COTS components is structured into several objects (Fig. 1) to support the properties of self-management of COTS components.

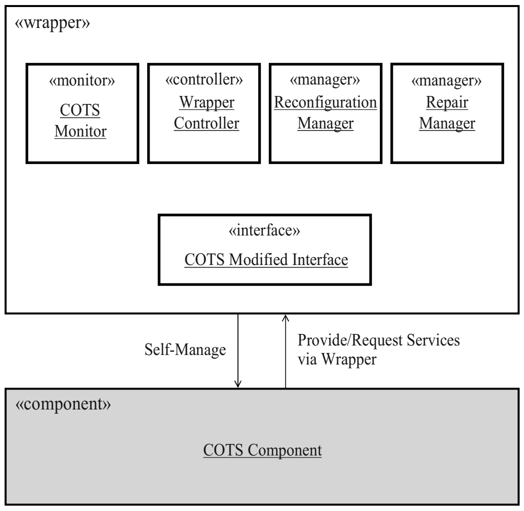

«wrapper»
«monitor» COTS Monitor
«controller» Wrapper Controller
«manager» Reconfiguration Manager
«manager» Repair Manager
«interface» COTS Modified Interface
Self-Manage
Provide/Request Services via Wrapper
«component»
COTS Component

Fig. 1 Wrapper Architecture for Self-Managed COTS Component

4.1. COTS Modified Interface

The COTS Modified Interface is a linker between a wrapper and its COTS component. The interface to an original COTS component is wrapped up by the COTS Modified Interface through which the COTS component provides services to and/or requires them from other components. The COTS Modified Interface captures the messages passing between the COTS component and other COTS components. The messages are delivered to the COTS Monitor, which detects anomalies in the COTS component using the messages.

The COTS Modified Interface maintains the status of each operation provided by a COTS component. An operation is “unblocked” if it performs its functionality normally, whereas an operation is marked “blocked” if either the operation is anomalous (referred to as malfunction block) or the operation requires anomalous operation in other COTS component (referred to as dependency block). An operation can be blocked during the repair of anomalous COTS component as well (referred to as repair block). When a COTS component (service requestor) requests an operation from other COTS component (service provider), the COTS Modified Interface for the service provider COTS component checks the status of the operation requested, and then it allows the operation to be performed if the operation is unblocked.

The COTS Modified Interface is involved in the dynamic reconfiguration of the anomalous COTS component before and after the repair. The details are described in section 4.3.

4.2 COTS Monitor

The operations provided by a COTS component are monitored by the COTS Monitor, which encapsulates statecharts describing the specifications of each operation of the COTS component. When an operation in a service provider COTS component is invoked by a service requestor COTS component, the statechart for the operation makes a transition from the “Idle” state to the “Performing the Operation” state. Some operation in a COTS component may need services from other COTS components to complete its own service. The statechart for an operation also describe these dependencies among the operations in COTS components.

The statecharts for each operation of a COTS component, encapsulated in the COTS Monitor, are executed by the messages from the COTS Modified Interface. The interface to a COTS component is invoked by the COTS Modified Interface, which is invoked by service requestor COTS components. The COTS Modified interface is an intermediary in interaction between a COTS component and its service requestor/provider COTS components, notifying the COTS Monitor when the COTS components communicate with each other. The COTS Monitor uses the notification messages to check the specifications of operations so that it determines when and which operation in a COTS component is anomalous.

Fig. 2 depicts the self-management of a COTS component using the message communication diagram of UML [Booch05, Rumbaugh05], in which the message sequence A1 through A8 describes the monitoring of the Operation1 of the COTS component that is invoked by an input device such as an elevator button. When the Operation1 is invoked by an input device (message A1), the COTS Modified Interface checks the status of the Operation1. If the Operation1 is blocked, it is not allowed to be called by the input device. However, if the Operation1 is unblocked, the COTS Modified Interface notifies the COTS Monitor that the Operation1 has been invoked (message A2), and calls the Operation1 in the COTS Component (message A3). Similarly, when the Operation1 needs other OperationK from a different COTS component (message A5 and A7), the COTS Modified Interface notifies the COTS Monitor (message A6 and A8).

The COTS Monitor presumes that an operation provided by a COTS component is anomalous if the expected notification messages have not arrived within a reasonable time interval. Each state in a statechart for an operation, encapsulated in the COTS Monitor, should make transitions within an expected time interval once the operation is invoked. Otherwise, the COTS Monitor reports the anomaly of the operation to the Wrapper Controller, which in turn takes an appropriate action against the anomalous operation. In Fig. 2, the COTS Monitor notifies the Wrapper Controller of the anomaly of the Operation1 (i.e., “Notify Failure” in Fig. 2) if it did not receive a notification message from the COTS Modified Interface within a predefined time interval.

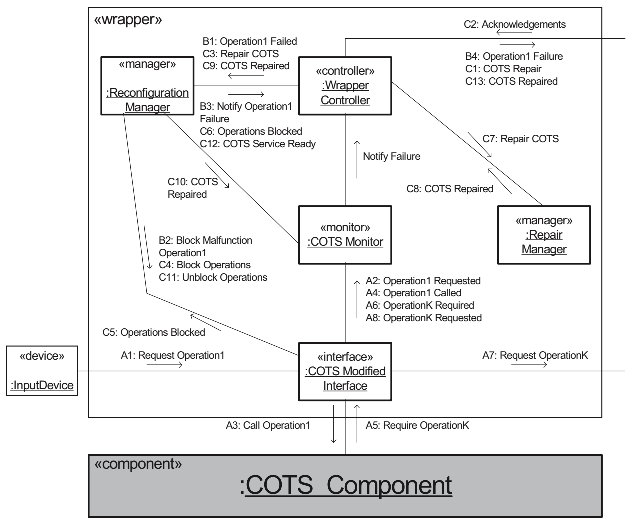


Fig. 2 Self-Management of COTS Component

4.3 Reconfiguration Manager

The reconfiguration of a self-managed COTS component is performed by the Reconfiguration Manager in a wrapper for the COTS component. The COTS components are reconfigured by blocking or unblocking the interfaces to each operation of COTS components, which are encapsulated in the COTS Modified Interface.

Anomalous operations are blocked so that other components cannot request the operations paralyzed. An operation is blocked either if the operation is anomalous, or if the operation requires anomalous operations in different COTS components. The status of an operation is changed to "Unblocked" as the operation resumes its service normally.

The Reconfiguration Manager has information associated with the reconfiguration of a COTS component. The information includes the status of both each operation in a COTS component and its associated operations (i.e., callee and caller operations) in other COTS components. An operation may call some operations (i.e., callee operations) in other components, or may be called by some operations (i.e., caller operations) in other components. The status of each operation (including its callee and caller operations) is classified into "Unblock", "Malfunction block", "Dependency block", and "Repair block". In addition, the Reconfiguration Manager contains the dependency relationship among operations in a COTS component and in other COTS components. This relationship is used to generate reconfiguration plans against anomalous operations both within a COTS component and in different COTS components.

Fig. 2 depicts the reconfiguration of COTS components against anomalous Operation1 in a COTS component. When an anomaly in the Operation1 has been detected, the Reconfiguration Manager is notified the status of the Operation1 by the COTS Monitor through the Wrapper Controller (message B1). With the notification, the Reconfiguration Manager updates the status of the operation, and generates a reconfiguration plan against the anomalous Operation1. Based on the plan, the Reconfiguration Manager sends a request to the COTS Modified Interface to block the anomalous Operation1 as "malfunction" (message B2), and then the COTS Modified Interface updates the status for this operation from "unblock" to "malfunction block". The Reconfiguration Manager sends a failure notification to the neighboring components (messages B3 and B4) through the Wrapper Controller as well.

4.4 Repair Manager

An anomalous operation of a COTS component is repaired by the Repair Manager by means of re-initialization, re-installation, or modification of input to the component. The anomalies in a COTS component are repaired at the level of a component, not at the level of each operation of a component. This is because a COTS component is a black box whose internal details are unknown enough to heal each operation provided by the COTS component. An anomalous COTS component can be reinitialized if the component provides an operation for initializing the component state. An anomalous COTS component can be also reinstalled using the same version of the COTS component or replaced with a variant of the component if it is available. The variants of a component can be different versions of the component, which may either be implemented in a different way or provide different performances. Some input to a COTS component may need to be modified to fix anomalous operations in the COTS component.

A COTS component should be repaired immediately when critical operations in the component come to be anomalous. Each operation in a COTS component can be classified into either a non-critical or critical operation. For example, in an elevator system with multiple elevators, an operation that controls the elevator direction lamp at a floor can be a non-critical operation, whereas an operation scheduling multiple elevators can be a critical operation. Non-critical operations in a COTS component may not result in system failures although the COTS component may provide degraded services. Non-critical operations may be fixed at the regular maintenance instead of immediate repair of the component. This is to avoid the interruption of services provided by a COTS component that are not affected from the anomalous, non-critical operations. However, critical operations should be healed immediately although the component repair may result in the service interruption.

Fig. 2 depicts the repair of an anomalous Operation1 in the case where the Operation1 is a critical operation to the system. The repair is performed by means of re-installation of the COTS component - message sequence C1 through C13. After isolating neighboring components from the anomalous component (messages C1 and C2), all operations in the COTS component being repaired are blocked to stop their services (messages C3 through C6) during the repair. Then the Repair Manager reinstalls the COTS component (messages C7 and C8). As the COTS component is repaired to provide service again, the COTS Monitor is reinitialized (messages C9 and C10) and all operations are unblocked (message C11). Finally the neighboring components are notified the readiness of the COTS component resuming services (messages C12 and C13).

4.5 Wrapper Controller

The Wrapper Controller coordinates the other objects in a wrapper to control the self-management of a COTS component. The Wrapper Controllers in wrappers for COTS components co-operate with each other to reconfigure the system at run-time against anomalies detected, so that the neighboring COTS components have the minimal impact from anomalous COTS components.

The Wrapper Controller in a wrapper for a COTS component determines whether an operation in a COTS component should be repaired immediately or not. The Wrapper Controller maintains information on operation type such as critical or non-critical operations. With the notification messages about anomalous operations from the COTS Monitor, the Wrapper Controller makes decisions about what actions will be taken against the operations - either just blocking anomalous operations, or both blocking and repairing the whole COTS component immediately.

5. Implementation

Part of the wrapper designed in this paper has been implemented and applied to the distributed elevator system with multiple elevators [Gomaa00] including three Elevator, ten Floor, and one Scheduler COTS components. The COTS Modified Interface, COTS Monitor, and Wrapper Controller objects were implemented using Java programming language, while the Reconfiguration Manager and Repair Manager are under development.

The Scheduler COTS component schedules three elevators in response to the elevator requests from floor components. The Scheduler Modified Interface contains a Scheduler Operation Status Table to trace the status of each operation provided by the Scheduler COTS component. If an operation of the Scheduler COTS component is unblocked, the operation can be called by the Scheduler COTS Modified Interface.

When the Floor Modified Interface requests an elevator from the Scheduler Modified Interface, the Scheduler Modified Interface notifies this event to the Scheduler Monitor by sending the "Service Request Arrived" message, and then the Scheduler Modified Interface calls the "ServiceRequest" operation in the Scheduler COTS component. The statechart for "ServiceRequest" operation in the Scheduler Monitor makes a transition by the notification message - from Idle state to the state "Waiting for Service Request Called". After calling the "ServiceRequest" operation, the Scheduler Modified Interface notifies the Scheduler Monitor of "Service Request Called" message. Then the Monitor starts timing the operation while it waits for the next notification message, "Scheduler Request Arrived" from the Scheduler Modified Interface (The "Scheduler Request" message is delivered to one of three Elevator Modified Interfaces so that an elevator is sent to a floor requesting an elevator). The "ServiceRequest" operation is anomalous if the notification message, "Schedule Request Arrived", does not arrive within an expected time interval.

When the "ServiceRequest" operation is detected as an anomalous operation, the Scheduler Wrapper Controller determines whether the operation is critical or non-critical using a Scheduler Operation Class Table. This operation is critical for the Elevator system because it receives requests for elevators from floors. Elevators cannot be sent to floors if this operation is anomalous. Thus the Scheduler COTS component needs to be repaired immediately.

The Scheduler Reconfiguration Manager generates a reconfiguration plan against the anomalous "ServiceRequest" operation using the Scheduler Reconfiguration Table (Table 1) and Scheduler/Callee/Caller Operation Status Table. When the critical "ServiceRequest" operation is anomalous, the Reconfiguration Plan Manager changes the status of all operations in the Scheduler Modified Interface to "repair block". It also notifies the Floor Modified Interface (Caller component in Table 1) so that the "FloorButtonRequest" operation in the Floor COTS component cannot call the anomalous "ServiceRequest" operation. The Scheduler Repair Manager starts repairing the Scheduler COTS component.

Scheduler Reconfiguration Table				
Operation	Callee		Caller	
	Operation	Component	Operation	Component
ServiceRequest	ScheduleRequest	Elevator 1	FloorButtonRequest	Floor
	ScheduleRequest	Elevator 2		
	ScheduleRequest	Elevator 3		
UpdateStatusPlan			ElevatorButton Request	Elevator #
			ScheduleRequest	Elevator #
UpdateElevator Status			ElevatorButton Request	Elevator #
			ScheduleRequest	Elevator #

Table 1. Scheduler Reconfiguration Table

6. Conclusions

This paper has described the design of a wrapper for self-managing COTS components. Each COTS component is structured into a self-managed COTS component using its own wrapper. A wrapper supporting the self-management of a COTS component is composed of several objects – COTS Modified Interface, COTS Monitor, Wrapper Controller, Reconfiguration Manager, and Repair Manager. These objects are designed to realize the requirements of a wrapper for self-managing COTS components. Each wrapper is capable of detecting anomalies in the COTS component, reconfiguring the COTS component against the anomalies detected, and repairing the anomalous COTS component.

This research can be extended to further research. The capability of decision mechanism in a COTS Monitor needs to be extended to detect anomalies of external devices (such as door or motor in the distributed elevator system) handled by a COTS component. Currently, a COTS Monitor determines if an operation of the COTS component is anomalous or not. However, the anomalies of an operation may result from the anomalies in external devices.

References

[Anderson03] Tom Anderson, Mei Feng, Steve Riddle, Alexander Romanovsky, Protective Wrapper Development: A Case Study, in Proceedings of the 2nd International Conference on COTS-Based Software Systems (ICCBSS 2003), Ottawa, Canada, 10-13 February 2003.

[Booch05] G. Booch, J. Rumbaugh, I. Jacobson, "The Unified Modeling Language User Guide", Second Edition, Addison Wesley, Reading MA, 2005.

[Dashofy02] Eric M. Dashofy, Andre van der Hoek, and Richard N. Taylor, "Towards Architecture-based Self-Healing Systems," Workshop on Self-healing systems, Proceedings of the first workshop on Self-healing systems, Charleston, SC, November18-19, 2002.

[DeLine99] R.DeLine, "A Catalog of Techniques for Resolving Packaging Mismatch," In Proceedings 5th Symposium on Software Reusability, Los Angeles, CA. May 1999, pp. 44-53.

[Garlan02] David Garlan and Bradley Schmerl, "Model-based Adaptation for Self-Healing Systems," Workshop on Self-healing systems, Proceedings of the first workshop on Self-healing systems, Charleston, SC, November18-19, 2002.

[Garlan03] David Garlan, Shang-Wen Cheng, and Bradley Schmerl, "Increasing System Dependability through Architecture-based Self-repair," in Architecting Dependable Systems. R. de Lemos, C. Gacek, A. Romanovsky (Eds), Springer-Verlag, 2003.

[Gomaa00] Hassan Gomaa, "Designing Concurrent, Distributed, and Real-Time Applications with UML," Addison-Wesley, 2000.

[Guerra02] Paulo Asterio de C. Guerra and Rogerio de Lemos, "An Idealized Fault-Tolerant Architectural Component," Workshop on Architecting Dependable Systems, ICSE'02 International Conference on Software Engineering, Orlando, FL, May 25, 2002.

[Guerra03] Paulo Asterio de C. Guerra, Alexander Romanovsky, Rogerio de Lemos, Integrating COTS Software Components into Dependable Software Architectures, in Proceedings of the 6th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2003), Hakodate, Hokkaido, Japan, 14-16 May 2003, pp. 139-142.

[Guerra04] Paulo Asterio de C. Guerra, Cecília Mary F. Rubira, Alexander Romanovsky, Rogério de Lemos, "A Dependable Architecture for COTS-Based Software Systems using Protective Wrappers," in Architecting Dependable Systems ADS II LNCS 3069. October 2004, pp. 147-170.

[IBM03] IBM, "An architectural blueprint for autonomic computing," IBM and autonomic computing, April 2003.

[Kim00] Kim, K.H. and Subbaraman, C., "The PSTR/SNS Scheme for Real-Time Fault Tolerance via Active Object Replication and Network Surveillance," IEEE Trans. on Knowledge and Data Engr., Vol.12, No.2, Mar./April 2000, pp.145-159.

[Koopman03] Philip Koopman, "Elements of the Self-Healing System Problem Space," Workshop on Software Architectures for Dependable Systems (WADS2003), ICSE'03 International Conference on Software Engineering, Portland, Oregon, May 3-11, 2003.

[Kopetz94] Hermann Kopetz and Gunter Grundteidl, "TTP-A Protocol for Fault-Tolerant Real-Time Systems," IEEE Computer, pages 14-23, January 1994.

[Meulen05] Meine van der Meulen, Steve Riddle, Lorenzo Strigini, Nigel Jefferson, Protective Wrapping of Off-the-Shelf Components, in Proceedings of the COTS-Based Software Systems: 4th International Conference, ICCBSS 2005, Bilbao, Spain, February 7-11, 2005.

[Rumbaugh05] J. Rumbaugh, G. Booch, I. Jacobson, "The Unified Modeling Language Reference Manual," Second Edition, Addison Wesley, Reading MA, 2005.

[Shin05a] Michael E. Shin, "Self-Healing Component in Robust Software Architecture for Concurrent and Distributed Systems," Science of Computer Programming, Vol. 57, No. 1, 2005, pp 27-44.

[Shin05b] Michael E. Shin and Daniel Cooke, "Connector-Based Self-Healing Mechanism for Components of a Reliable System," Workshop on Design and Evolution of Autonomic Application Software (A Workshop of ICSE2005), St. Louis, Missouri, USA, May 21, 2005.

[Shin06] Michael E. Shin and Fernando Paniagua, "Self-Management of COTS Component-Based Systems Using Wrappers," 30th Annual International Computer Software and Applications Conference (COMPSAC 2006), Chicago, September 17-21, 2006.

[Thomas00] V. Thomas, SuriKumar Kareti, Walter Heimerdinger, Sunondo Ghosh, "Mediators and obligations: An architecture for building dependable systems containing COTS software components," Proceedings Workshop on Dependable System Middleware and Group Communication (DSMGC 2000), Nuremberg, Germany, October 2000.

[Torres-Pomales00] Wilfredo Torres-Pomales, "Software Fault Tolerance: A Tutorial," NASA/TM-2000-210616, October 2000.

QoS-Optimized Integration of Embedded Software Components with Multiple Modes of Execution [*]

Zonghua Gu
Department of Computer Science and Engineering
Hong Kong University of Science and Technology
Hong Kong, China

Qingxu Deng
Department of Computer Science
Northeastern University
Shenyang, China

Abstract

Component-Based Software Engineering (CBSE) is an effective approach for tackling the increasing complexity of large-scale embedded software development projects. Software components often have multiple modes of execution, each characterized by a set of resource requirements such as maximum stack and heap memory size, worst-case execution time, etc, and one or more Quality of Service (QoS) values that measure its benefit. There is typically a tradeoff relationship between the resource requirements and utility value. In this paper, we address the problem of optimizing system utility when composing multiple software components into a complete system, each having a set of discrete modes of execution with different resource requirements and utility values. We present an optimal branch-and-bound algorithm and another fast heuristic algorithm for solving the optimization problem.

1 Introduction

Embedded software components are often *modal*, each having several modes of execution, and each mode having a different set of QoS value and resource consumption characteristics. For example, a software component in an embedded control system may have two possible runtime modes: one implementing a sophisticated control algorithm with good control performance but has high resource requirements in terms of execution time and memory size, and the other implementing a simplistic control algorithm with below-average control performance but has low resource requirements. In this paper, we assume that we can assign an application-specific *utility value* to each mode to quantify its QoS. For example, in control theory, the control performance can be measured with a composite index number that combines the steady-state error, maximum overshoot, settling time and rise time.

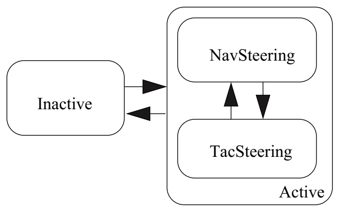


Figure 1. A modal software component *Steering* **with two sub-modes of execution when it is active:** *NavSteering* **(Navigational Steering) and** *TacSteering* **(Tactical Steering).**

As a more concrete example, Figure 1 shows a modal component taken from Avionics Mission Computing [1] with two high-level modes of execution: *active* and *inactive*. The *active* mode further has two sub-modes within it, each offering the same input/output interface but with different resource consumption and performance characteristics. When the component is in the *TacSteering* mode, it produces results with high accuracy but also has high CPU requirements; when it is in the *NavSteering* mode, it produces results with lower accuracy but has lower CPU requirements. The component may switch to a different mode at runtime depending on runtime resource availability. This can be implemented by having multiple implementations for each component, loading all of them into memory at system startup, and switching among them dynamically at runtime.

As a demonstration of the modal component in ac-

[*]This work is partially supported by the Cultivation Fund of the Key Scientific and Technical Innovation Project (No 706016), Ministry of Education, China and the Competitive Earmarked Research Grant (No 613506), Research Grants Council, Hong Kong.

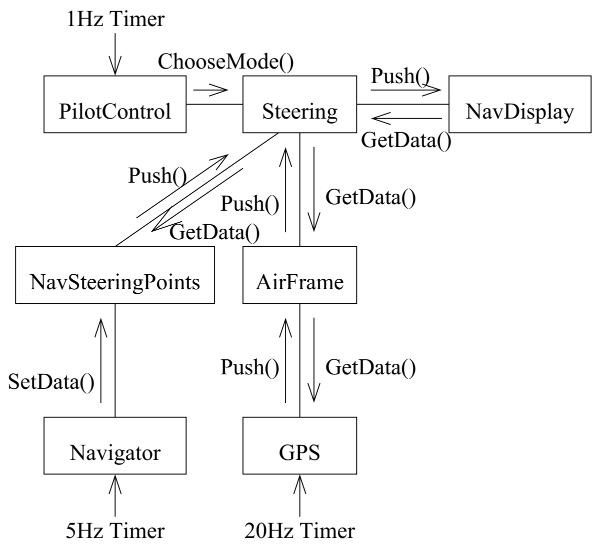


Figure 2. An application scenario.

tion, Figure 2 shows an application scenario that contains the modal *Steering* component. The behavior of this scenario can be described in three stages depending on the mode of the *Steering* component, which can switch between three modes, i.e., *Inactive*, *NavSteering* and *TacSteering*, at runtime:

1. The *GPS* component wakes up upon a 50ms (20Hz) interval timeout and issues a *Data Available* event. The *Airframe* component receives the event and issues its own *Data Available* event. The *Steering* component receives this event, updates its internal state, but does not issue events since it is initially inactive.

2. *PilotControl* issues a command that enables the *NavSteering* mode of the *Steering* component. Now when the *Steering* component receives the *Airframe*'s *Data Available* event, it updates its state and issues a *Data Available* event, causing the *NavDisplay* to display the navigational steering data obtained from the *NavSteeringPoints* component.

3. *PilotControl* issues a command that enables the *TacSteering* mode of the *Steering* component. From this point on, the *NavDisplay* component displays the tactical steering data instead of the navigational steering data.

Embedded systems used to be designed in a fairly static fashion for the sake of predictability, with a fixed set of tasks run at a static cyclic schedule. However, modern embedded systems are required to handle more dynamic situations, where unexpected tasks may arrive and leave the system at runtime, and system resource availability may also change dynamically due to runtime load variations and hardware faults. This requires the engineer to add flexibility and adaptivity to runtime execution, in order to protect the system from overload by trading off resource requirements for QoS and achieve graceful degradation. Existing work on dynamic runtime adaptation is typically programmed in an *ad hoc* manner, and reconfiguration code is tangled with application functional code. This breaks component modularity, a key benefit of CBSE, and hinders system maintenance and evolution. We propose an approach of predefining a set of *system modes*, each of which is a combination of component modes designed and optimized offline at design time according to some optimization criteria. In each mode, a different set of components are active, and each active component may be in one of its sub-modes depending on the current active system mode. At runtime, one of the system modes is activated as the initial default mode. The system may change modes during execution to accommodate dynamic resource availability. The runtime middleware monitors runtime conditions and dynamically switches between system modes based on runtime resource availability by changing component modes. Its role is similar to a transaction manager in database systems, and coordinates system-wide mode changes while maintaining global consistency. This approach allows the designer to use time-consuming optimization techniques to predefine and optimize system modes at offline design time, yet incurs low overhead for performing dynamic mode changes at runtime. Figure 3 shows the overall workflow.

Possible resource constraints include total memory size, CPU utilization (percentage of time when the CPU is busy), network bandwidth, etc. For example, for a system with n tasks scheduled in a rate monotonic fashion, the upper bound of the total CPU utilization in order to guarantee schedulability is $n(2^{1/n} - 1)$. We solve the optimization problem to obtain a set of system modes, each characterized by a tuple [*CPU utilization, total memory size, system utility*] expressing the tradeoff relationships between the available resources and the achievable system utility. At runtime, the middleware adaptively switches among different system modes to cope with varying system load and resource availability, e.g., when the system is lightly loaded, then upgrade to a system mode with high utility; when overload occurs, downgrade to a system mode with low utility by producing lower quality results. In this paper, we focus on the design stage, where the engineer

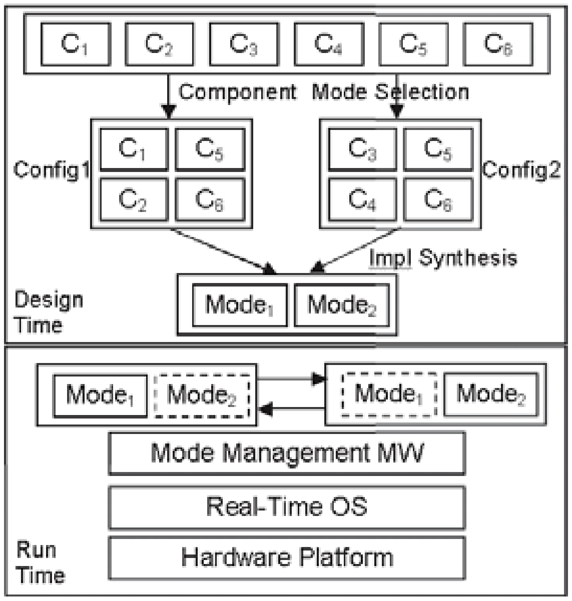


Figure 3. Overall workflow of the proposed methodology. A dotted box denotes an inactive mode.

selects a mode for each component in the application to form multiple system modes, each optimized for a different design objective, such as high performance or low resource requirements.

The rest of this paper is structured as follows: we present the problem formulation in Section 2, then present two optimization techniques in Section 3, one optimal algorithm based on exhaustive search and another fast heuristic algorithm. We present performance evaluation results in Section 4. We discuss related work in Section 5, and draw conclusions and discuss future work in Section 6.

2 Problem Formulation

An embedded software application consists of a set of real-time tasks, each containing a set of software components. Here we assume that each software component belongs to at most one task, and it is not possible for multiple tasks to share one component, as discussed in [2]. There are n software components in the system $(C_1, C_2, \ldots, C_n)$, taking into account components in all tasks. The available system resources are given by a vector $\mathbf{R} = (R_1, \ldots, R_m)$. Each component C_i, where $i = (1, \ldots, n)$, has l_i possible execution modes C_{ij}, where $j = (1, \ldots, l_i)$, each of which has a set of resource requirements $\mathbf{r_{ij}} = (r_{ij1}, \ldots, r_{ijm})$, and a user-defined utility metric v_{ij}. (There may be multiple utility metrics in general, but we only consider the case with a single utility metric here.) There may be *mutual exclusion* constraints, e.g., no more than one component among a set of components is active in any system mode, and *mutual attraction* constraints, e.g., a set of components are always active simultaneously, or none of them are active in any system mode. The component integrator must define a small set of system modes such that each satisfies resource constraints and optimizes the utility metrics. For example, one system mode may be *high-performance* with high resource consumption, while another may be *degraded-performance* with minimal resource consumption, designed to cope with the situation where certain faults may have occurred and reduced the system resource availability.

Assuming each component mode is associated with a set of resource requirements and a utility value, the optimization problem can be formulated as *finding the combination of component modes with the largest total utility value given total cost constraints.* A special case is when all constraints are mutual attraction, and the system has a few fixed system-level modes of certain component mode combinations without any freedom for design space exploration. In this paper, we assume that there is some flexibility in choosing component modes, hence there is a need for searching through a potentially large design space.

Suppose we have two resource constraints, CPU utilization U and total memory size M; one optimization objective, an application-specific utility value V, with a monotonic tradeoff relationships V grows larger with larger U and M. We can formulate the optimization problem in several different ways:

1. Given an upper bound on U and an upper bound on M, maximize V.

2. Given a lower bound on V and an upper bound on M, minimize U.

3. Given a lower bound on V and an upper bound on U, minimize M.

4. Optimize U, M and V jointly to obtain a *Pareto* optimal curve [1] using Evolutionary Algorithms [3].

In other words, we can optimize one objective while treating the others as constraints, or we can jointly optimize all objectives simultaneously. The formulation of joint multi-objective optimization is more general

[1]A Pareto optimal curve for a multi-objective optimization problem contains all the solutions where no solution is inferior or superior to any other solution for all objectives.

than single objective optimization, since the single-objective solutions can be easily obtained from the Pareto optimal curve. In this paper, we focus on the 1st problem formulation by using both exhaustive search and fast heuristic algorithms.

3 Solving the Optimization Problem

Branch-and-bound is widely used for solving combinatorial optimization problems. It works by constructing a search tree, and each node in the tree represents a solution where some variables are assigned and others are free. A node is expanded by assigning value to a free variable, e.g., expanding a node by assigning value to a binary variable x generates two nodes: one with $x = 0$ and another with $x = 1$.

For our optimization problem, each leaf node in the search tree is a complete solution represented by a vector $\mathbf{x} = \{x_{ij}\}$, where $i = 1, \ldots, n$ and $j = 1, \ldots, l_i$. For component C_i, $x_{ij} = 1$ means that mode j of component i is active, and $x_{ij} = 0$ means that it is inactive. we use another bit vector $\mathbf{b} = (b_1, \ldots, b_n)$ to represent the status of mode assignment for components $(C_1, \ldots, C_n)$. $b_i = 1$ indicates that C_i's mode has already been assigned, and $b_i = 0$ indicates that C_i's mode has not been assigned. The algorithm has the following key steps: start with the initial state where none of the component modes have been assigned. Find the tree node N with the largest upper bound on utility value. If the bit vector $\mathbf{b}$ for node N contains all 1's, then all component modes have been assigned. Record a solution and backtrack. Otherwise, the bit vector $\mathbf{b}$ contains at least one 0, then expand node N by generating l_j children nodes to represent all possible mode assignments for a component C_{ij} whose mode has not been assigned. Prune any generated tree node that violates resource constraints and backtrack. Iterate until all component modes have been assigned, or no feasible solution is found.

Since combinatorial optimization problems are NP-Hard in general, running time of the branch-and-bound algorithm grows exponentially with the number of components. Suppose an application consists of 3 components, and each component has 2 execution modes, then we have $2^3 = 8$ possible system modes. In order to scale up to larger models, we next present a heuristic algorithm for solving the optimization problem. The algorithm has the following key steps:

1. Start with the system mode where each component is in the mode with the smallest utility value. If this system mode does not satisfy memory size or utilization constraints, then there is no feasible solution, and the algorithm terminates.

2. Find the additional resources $\mathbf{r} = (r_1, \ldots, r_m)$ needed to upgrade each component to the next higher utility value, and choose the component mode upgrade that provides the largest utility gain divided by additional resource demand.

3. If a feasible component mode upgrade cannot be found, then return the current solution and terminate. Otherwise, go to step 2.

In the above description, we start from a feasible solution with lowest utility value and gradually upgrade each component to a higher utility mode until the resource bound is violated. We then choose the solution that satisfies the resource bounds and has the largest utility. It is also possible to start from an infeasible solution with highest utility value and gradually downgrade each component to a lower utility mode until the resource bound is satisfied. The former approach may converge to the solution faster when resources are scarce, and the latter approach may converge faster when resources are abundant.

4 Performance Evaluation

To evaluate the performance of our optimization algorithms, we generate sets of modal components, and assign random values of CPU utilization and utility to each component mode. We assume that CPU is the bottleneck resource and do not consider memory size constraints, therefore the resource vector $\mathbf{r}$ becomes a single scalar value. We optimize the total system utility given different upper bounds on CPU utilization. The optimization algorithms are run on a Pentium PC with 3.2 Ghz CPU and 1GB memory. Table 4 shows the performance results. N_c denotes total number of components; M_c denotes number of modes per component; UB denotes the utilization bound; V_{BB} denotes the maximum utility achieved with branch-and-bound algorithm. When the utilization bound is 1.0, we scale and normalize the maximum utility to be 1 for easy comparison. V_{HEU} denotes the maximum utility achieved with the heuristic algorithm; T_{BB} denotes the branch-and-bound algorithm running time; T_{HEU} denotes the heuristic algorithm running time. For the heuristic algorithm, we start from the system mode where each component is assigned the mode with lowest utility value, and scale up gradually. As we can see, running time of the branch-and-bound algorithm increases sharply with the number of components and number of modes per component, while running time of the heuristic algorithm stays more or less constant. Optimization results of the heuristic algorithm are remarkably close to the true optimal calculated with

N_c	M_c	UB	V_{BB}	V_{HEU}	$T_{BB}(s)$	$T_{HEU}(s)$
5	2	1.0	1.0	0.99	1.7	0.21
5	2	0.8	0.91	0.88	1.4	0.15
5	2	0.6	0.78	0.74	0.9	0.11
10	3	1.0	1.0	0.94	34.2	0.94
10	3	0.8	0.86	0.82	27.3	0.88
10	3	0.6	0.67	0.64	20.8	0.71
15	3	1.0	1.0	0.95	403.2	1.13
15	3	0.8	0.89	0.84	356.1	0.97
15	3	0.6	0.59	0.53	316.2	0.85
20	4	1.0	1.0	0.97	2598.1	1.41
20	4	0.8	0.91	0.87	2109.1	0.99
20	4	0.6	0.64	0.62	1600.5	0.91
20	10	1.0	1.0	0.89	9374.0	1.70

Table 1. Performance evaluation results.

branch-and-bound, as we can see by comparing the columns labeled V_{BB} and V_{HEU}. Note that the algorithm running time gets smaller when the utilization bound is reduced, which results in reduction of design space size. The final result is a set of system modes, each consisting of a collection of component modes and characterized by a tuple [*CPU utilization, system utility*]. At runtime, different system modes may be activated depending on runtime resource availability.

5 Related Work

Most existing work on CBSE focuses on the issue of component integration based on *functional criteria*, i.e., how to choose a set of components that implement certain functionality and are compatible with each other. Our focus is on component integration based on *non-functional criteria*, where the designer must choose among multiple component modes to form system modes that offer similar functionality, but have different resource requirements and performance/utility attributes.

Some researchers have addressed the problem of embedded software component integration. Sedigh-Ali [4] presented a graph-based model for component-based software development with optimization objectives such as system reliability, complexity metric and cost constraints, but no details on solving the problem were provided. Neema *et al* [5] developed a design space exploration tool (DESERT) based on Boolean Decision Diagrams (BDD) data structure. The tool starts with a set of design templates, each template consisting of several possible implementation alternatives, and synthesizes fully specified models that meet selected design constraints. However, the DESERT approach allows pruning of design space based on design constraints, but does not offer the capability of optimization. That is, it gives the designer a set of implementation alternatives that all satisfy design constraints, but does not give a ranking of the alternatives based on one or more optimization objectives. Wandeler *et al* [6] introduced a component system for interface-based design of systems with mixed Fixed Priority, Rate Monotonic and Earliest Deadline First scheduling. Henzinger *et al* [7] presented an assume-guarantee interface algebra for real-time components that supports both the incremental addition of new components and the independent stepwise refinement of existing components. Gößler *et al* [8] proposed a framework for component-based modeling using an abstract layered model for components, which ensures correctness by construction of a system from properties of its interaction model and of its components. The properties considered include global deadlock-freedom, individual deadlock-freedom of components, and interaction safety. However, optimization of QoS under resource constraints is not considered.

Abdelzaher *et al* [9] proposed a model for quality-of-service (QoS) negotiation in building real-time services with the goal of guaranteeing predictable performance under specified load and failure conditions, and ensuring graceful degradation when these conditions are violated. Since their algorithm was designed to operate at runtime, an efficient greedy heuristic is adopted to tradeoff computation time with solution optimality. Our approach is based on offline optimization of multiple pre-defined system modes and online switching among them, therefore we can afford to use time-consuming optimization techniques such as branch-and-bound to achieve more optimal results.

6 Conclusions and Future Work

In this paper, we address the problem of optimized integration of embedded software components with multiple execution modes. Even though we have used memory size and CPU utilization as resource constraints, and a user-defined utility metric as optimization objective, our problem formulation is quite general and abstract and can be easily extended to address other resource and utility function definitions. As an example, *Critical Scaling Factor* (CSF) [10, 2] was proposed to be a metric that measures the sensitivity of a real-time system to variations of task execution times. It is defined as the largest coefficient by which execution time of all tasks can be simultaneously multiplied while preserving feasibility. For example, if a system has CSF of 1.17, then if we multiply the execution time of all tasks by a number $n \leq 1.17$, the system would still be schedulable. However, any

$n > 1.17$ would render the system unschedulable. A larger CSF value means that the system is more robust to timing faults caused by inaccuracies in execution time estimation. A system is schedulable if and only if its CSF $n \geq 1$. Therefore, it is desirable to adopt scheduling algorithms and task timing attribute assignments to maximize the CSF. We can treat CSF as another optimization objective in addition to one or more utility metrics. One way to solve the multi-objective optimization problem is by converting it to a single-objective one using a weighted sum of the multiple objectives. However, this approach is often not desirable since the assignment of weights can be arbitrary when combining multiple unrelated metrics. Another approach is to formulate a multi-objective optimization problem and solve it using evolutionary algorithms [3].

References

[1] Z. Gu, S. Wang, S. Kodase, and K. G. Shin, "An End-to-End Tool Chain for Multi-View Modeling and Analysis of Avionics Mission Computing Software," in *IEEE Real-Time Systems Symposium (RTSS)*, 2003, pp. 78–81.

[2] Z. Gu and K. G. Shin, "Synthesis of real-time implementation from component-based software models," in *IEEE Real-Time Systems Symposium (RTSS)*, 2005.

[3] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler, "PISA — a platform and programming language independent interface for search algorithms," in *Evolutionary Multi-Criterion Optimization (EMO 2003)*, ser. Lecture Notes in Computer Science, C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, Eds. Berlin: Springer, 2003, pp. 494 – 508.

[4] S. Sedigh-Ali and A. Ghafoor, "A graph-based model for component-based software development." in *WORDS*, 2005, pp. 254–262.

[5] S. Neema, J. Sztipanovits, G. Karsai, and K. Butts, "Constraint-based design-space exploration and model synthesis." in *EMSOFT*, 2003, pp. 290–305.

[6] E. Wandeler and L. Thiele, "Interface-based design of real-time systems with hierarchical scheduling." in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2006, pp. 243–252.

[7] T. A. Henzinger and S. Matic, "An interface algebra for real-time components." in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2006, pp. 253–266.

[8] G. Gößler and J. Sifakis, "Composition for component-based modeling." *Sci. Comput. Program.*, vol. 55, no. 1-3, pp. 161–183, 2005.

[9] T. F. Abdelzaher, E. M. Atkins, and K. G. Shin, "QoS Negotiation in Real-Time Systems and Its Application to Automated Flight Control." *IEEE Trans. Computers*, vol. 49, no. 11, pp. 1170–1183, 2000.

[10] S. Vestal, "Fixed-priority sensitivity analysis for linear compute time models," *IEEE Trans. Software Eng.*, vol. 20, pp. 308–317, 1994.

A C++ Framework for Developing Component Based Software Supporting Dynamic Unanticipated Evolution

Andre Rodrigues, Hyggo Almeida, and Angelo Perkusich
Embedded Systems and Pervasive Computing Lab,
Electrical Engineering and Informatics Center, Federal University of Campina Grande
Postal Code 10.105 - 58109-970, Campina Grande, PB, Brazil
felipe.andre@gmail.com, hyggo@dsc.ufcg.edu.br, perkusic@dee.ufcg.edu.br

Abstract

In this paper we present a C++ component-based framework, called CCF, for developing software with support to dynamic unanticipated evolution. The CCF is an implementation of the COMPOR Component Model Specification, which provides mechanisms to manage software runtime evolution, even for unpredicted changes. We describe the framework design and main implementation issues. To exemplify the use of the proposed framework, we describe the development of an application for encoding/decoding files for a compact internat tablet.

1. Introduction

In 1980, a study published by Bennet Lientz pointed out evolution activities as responsible for 50% of the total software cost [8]. In a more recent study, presented by Len Erlikh in 2000, evolution activities in software development increased to 90% [5]. Motivated by this scenario, several engineering approaches have been proposed to reduce the software evolution cost. One of theses approaches is the Component Based Development (CBD)[2], which provides means to reduce the impact of requirement changes in the application design and code, by establishing well defined interfaces and preparing the architecture for anticipated changes.

However, the impact of software evolution on both the design and the existing code is more significant when the changes in the software requirements have not been anticipated. The problem occurs when part of the software which had not been prepared for changes needs to be modified. Managing unanticipated evolution is more difficult in some domains, in which the software execution cannot be interrupted due to financial or security reasons, such as telecommunications, banking systems, military systems, hospital systems, etc. For these cases, an infrastructure for dynamic unanticipated evolution is necessary.

Within this context, the *COMPOR Component Model Specification* (CMS) provides mechanisms for developing component based software with support to dynamic unanticipated evolution [1]. Following this specification, application components can be added, removed, and changed without stopping the system execution.

In this article we present a C++ based framework for developing component based software supporting dynamic unanticipated evolution called CCF (C++ Component Framework). This framework implements the CMS [1], providing an API to develop C++ applications with support to unanticipated changes, even at runtime. We describe the framework design and main implementation issues.

We also present the implementation of an adaptable image compression application based on DjVu [7]. Such an application, named DjVu Compactor, is implemented using the CCF for a Linux based mobile device. The choice to implement it in a mobile device is to make clear the needs and advantages of the dynamic evolution feature when developing application for computational devices with limited memory and storage capacity, such as mobile phones and compact internet tablets.

The remainder of this paper is organized as follows. In Section 2, we describe the COMPOR Component Model Specification. In Section 3, we present the proposed framework. Section 4 describes the case study. Section 5 discusses related works. Finally, in Section 6, we present the concluding remarks.

2. Component Model Specification (CMS)

The COMPOR Component Model Specification is based on a hierarchical structure, where leaf-nodes represent **functional components** and the other nodes represent **containers**. Functional components provide, through services and events, the system functionalities and do not have child components. Containers are software entities that implement no application-specific functionalities. The containers work as "folders" storing the child components, which may be functional components or other containers. Moreover, they manage the services and events provided by their children [1].

Functional components are made available by inserting them into containers. Therefore, at least one container, named root container, must exist. The containers store two tables, one for provided services and other for events of interest to their child components. When a component is inserted into a given container, both the *provided services* and the *events of interest* tables of each container up to the root of the hierarchy are updated. After that, any component of the hierarchy can access the services provided by the new component as well as notify events of interest to it. The removal and updating processes are similar to the insertion one. Figure 1 shows the component deployment process and its steps are detailed as follows.

1. The Component X that implements the "calculate" service is inserted in Container 2.

2. The Container 2 updates its service table redefining the services provided by its child components.

3. The Container 2 requests its parent container to update its service table.

4. The Container 1 updates its service table redefining the services provided by its child components.

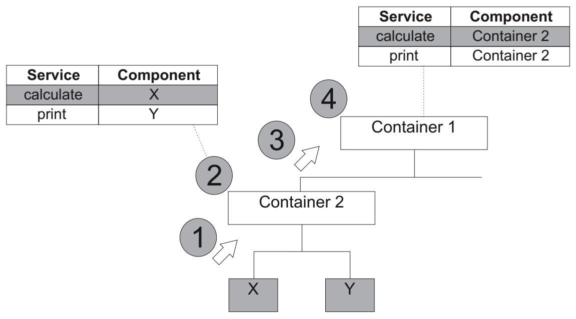


Figure 1. Component deployment.

After the component "X" deployment, any component of the hierarchy can access its services as well as notify it when some event of its interest occurs, even without having an explicit reference for "X". Observe that a component has only the reference to its parent container.

The interaction among components can be performed based on services or events. In the service based interaction a functional component can invoke any service provided by another component of the hierarchy, even if it belongs to a different container. On the other hand, the event based interaction focuses on the announcement of a state change in some functional component to all the interested ones. In both cases there is no explicit reference among components.

2.1. Service based interaction

When a service request occurs, the service provider must be found. Thus, the request of the "save" service by a component "X" triggers a search in the hierarchy for a component "K" that implements the "save" service. Such a process is presented in Figure 2 and the steps are detailed as follows.

1. The Component X requests the execution of the "save" service to its parent container.

2. The Container 2 verifies, according to its service table, that none of its child components implements the service "save".

3. The Container 2 forwards the request to its parent container.

4. The Container 1 verifies, according to its service table, that one of its child components implements the "save" service(Container 3). For Container 1, the Container 3 is the component that provides the service.

5. The Container 1 forwards the request to Container 3.

6. The Container 3 does not implement the service but has a reference to the real provider of the service - Component "K".

7. The Container 3 forwards the request to Component "K".

8. The Component "K" executes the "save" service and returns the result.

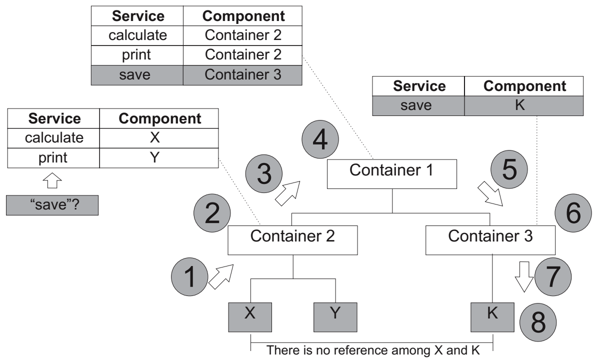


Figure 2. Service based interaction.

It is important to observe that there is no reference between the requester component ("X") and the provider component("K"). Thus, the component that implements the "save" service can be changed without modifying the rest of the structure.

2.2. Event based interaction

When an event is announced by a given functional component, all the components in the hierarchy of the application that are related to the event must be notified. The interaction based on events is also implemented by containers, and thus there are no direct references among functional components. This process is shown in Figure 3 and the steps are detailed as follows.

1. The component X announces an event named "Event A".
2. The announcement is directly received by its parent container (Container 2), which verifies if any of its child components have to be notified about the event, by inspecting the event table.
3. Container 2 forwards the event to the interested components, in this case only the component Y.
4. Container 2 then forwards the event to its parent container (Container 1).
5. Container 1, according to its event table, forwards the event to those interested on it, except the one that announced the event (Container 2). Since Container 1 is the root of the hierarchy, there is no parent container to forward the event. Thus, the event is only forwarded to Container 3.
6. Container 3 forwards the event according to its event table that in this case is component K.

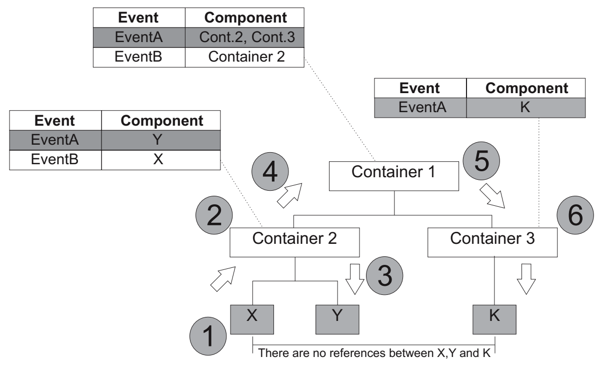
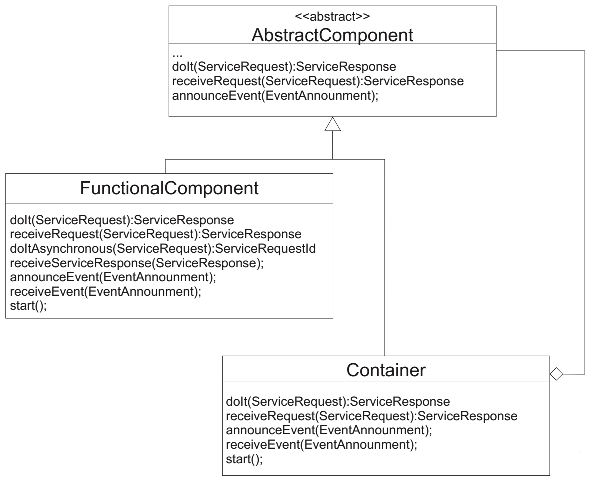


Figure 3. Event based interaction.

As can be seen in Figure 3, there are no references between the component that announced the event (X) and those interested on it (Y and K).

3. C++ Component Framework

The C++ Component Framework (CCF) is a C++ implementation of the CMS. The CCF design is based on the Composite design pattern [6], which is used to compose objects in tree structures in order to represent part-whole hierarchies. The CCF main methods are described in the class diagram depicted in Figure 4. The functional components and containers are instances of the classes `FunctionalComponent` and `Container`, respectively. The abstract class `AbstractComponent` allows composing complex elements from simple elements, through recursive composition [6]. Thus, the containers do not know if their children are functional components or other containers. The `AbstractComponent` class defines the methods that must be overwritten by the subclasses. Therefore, the `FunctionalComponent` and `Container` classes implement in different ways the extended methods, for both service and event interaction models.



<<abstract>>
AbstractComponent
...
doIt(ServiceRequest):ServiceResponse
receiveRequest(ServiceRequest):ServiceResponse
announceEvent(EventAnnounment);
FunctionalComponent
doIt(ServiceRequest):ServiceResponse
receiveRequest(ServiceRequest):ServiceResponse
doItAsynchronous(ServiceRequest):ServiceRequestId
receiveServiceResponse(ServiceResponse);
announceEvent(EventAnnounment);
receiveEvent(EventAnnounment);
start();
Container
doIt(ServiceRequest):ServiceResponse
receiveRequest(ServiceRequest):ServiceResponse
announceEvent(EventAnnounment);
receiveEvent(EventAnnounment);
start();

Figure 4. CCF simplified class diagram.

There are two methods to implement the service based interaction: `doIt` and `receiveRequest`. These methods are responsible to find and execute a service request following the hierarchy. The `doIt` method forwards the service request, in a bottom-up way, until reaching the container that stores the reference to the provider. When this occurs, the `receiveRequest` method is invoked to forward the requisition, in a top-down way, until arriving the functional component that implements the service (Figure 5). Both methods receive, as parameter, a

ServiceRequest object that encapsulates the name and the necessary parameters for the service execution. The return of both doIt and receiveRequest methods is a ServiceResponse object that stores the service result, when the execution occurs successfully, or stores the exception, when the execution fails.

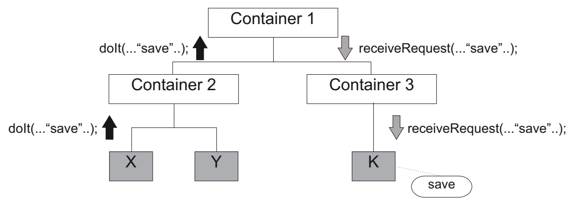


Figure 5. The doIt and receiveRequest methods.

The service based interaction can also occur asynchronously. For that, a component requests a service execution through the doItAsync method and receives a request identifier (ServiceRequestId). Then, a new thread is started to invoke the doIt method. In this way, while all the hierarchy is visited by the doIt method to find the service provider, the main thread can continue its execution normally. When the doIt method execution ends, the response (ServiceResponse) is forwarded to the component that had requested the service by calling the receiveAsyncServiceResponse method. As the ServiceResponse stores the request identifier, the component that invoked doItAsync method can identify which request a given response refers to.

The event announcement is similar to the asynchronous service invocation. The announceEvent method starts a new thread where the announced event is forwarded to all containers, in a bottom-up way. When a container identifies that some child component has interest in the announced event, the receiveEvent method is invoked, in a top-down way, until the notification reaches the interested components.

The asynchronous service interaction and the event interaction are based on the *ActiveObject* [12] design pattern. Such a pattern is implemented using the Pthreads API [10]. This API allows encapsulating the processing of the new thread within a function. Pthreads has been chosen because it is an IEEE POSIX standard and presents implementations in several platforms, such as Windows, Linux, Solaris etc.

FunctionalComponent class contains attributes of type ProvidedSevice and EventOfInterest. The ProvidedSevice and EventOfInterest classes have an attribute to store a reference to the method that will be invoked when the service request or the event announcement occurs. The storage and recovery of these methods is performed through computational reflection [3]. As the C++ language does not have a native reflection mechanism, the CCF uses the Seal Reflex library [11]. Such a library makes possible a C++ program to recover information about a class and its members as well as to create a class instance through its name. Furthermore, methods can be referenced and invoked through information on their signatures. Figure 6 shows the process that allows the computational reflection in a CCF component. The steps related to such a process are detailed as follows.

1. The genreflex tool is used to generate the dictionary, MyComponentRflx.cpp, with information about the class described in the MyComponent.h file.

2. The shared library, MyComponentRflx.so, with the dictionary information is generated.

3. The functional component shared library, MyComponent.so, is generated.

4. The shared libraries previously generated are loaded in the CCF. After that, the computational reflection can be performed in the inserted functional component.

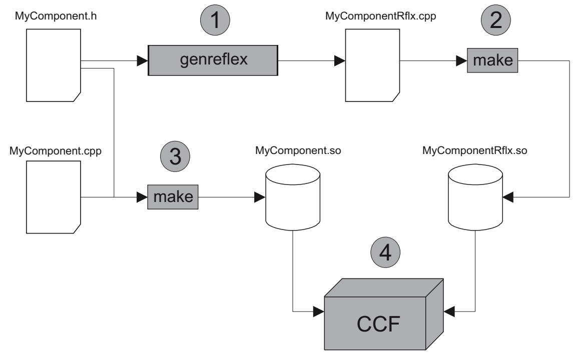


Figure 6. Computational reflection in CCF

Moreover, the CCF provides a mechanism for starting and stopping the components execution. The initialization and finalization of the components are implemented by the start and stop methods, respectively.

4. Application: DjVu Compactor

DjVu [7] is a image compression technology capable of replacing formats such as PDF, PS, and TIFF, with advantages in terms of size and transmission efficiency for scanned documents, digital documents and high resolution photographs. It is an open format, although proprietary, whose specifications are available under GPL license. Comparing with the PDF format, DjVu files are about 3 to 8 times smaller than PDF files and can be visualized without the need for downloading the entire document, therefore the parts of documents that are displayed on the screen are decompressed on-the-fly. The technology was developed by AT&T Laboratories and, today, an open source implementation of the decoders and part of the encoders, called DjVuLibre[1], is available.

The DjVuLibre project makes available various implementations for both encoders and decoders. Two examples are: (i) the *bzz*, which is a general purpose encoder/decoder, similar to *bzip2*, and; (ii) the *cjb2*, which is an image encoder. Although several other enconder/decorders are available, they are not all used at the same time. Also, other ones can be developed whenever new file formats will be specified.

In order to provide an on-demand loading/unloading of encoders/decoders, an application called DjVu Compactor has been developed using the CCF. The DjVu Compactor provides a structure for insertion and removal at runtime of DjVu encoders/decoders implementations. Thus, the user can tune the Djvu Compactor to its needs. Figure 7 illustrates the component hierarchy of DjVu Compactor. The screens shots shown in Figure 8 illustrate the *bzz* component removal and the *cjb2* component addition.

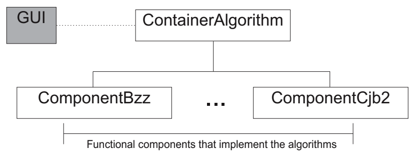


Figure 7. DjVu Compactor components hierarchy

The on-demand dynamic evolution feature is very important for computational devices with limited memory and storage capacity, such as mobile phones and compact internet tablets. In this context, the implementations of CCF and DjVu Compactor have been ported to the MAEMO platform[2]. MAEMO is a Linux based open platform to create applications and it is available for the Nokia Internet Tablet products, such as Nokia N800 and Nokia 770. This way, it is possible to develop other applications for such a platform with on-demand dynamic evolution feature.

5. Related Work

In the context of infrastructures for supporting dynamic unanticipated software evolution in C++, we highlight the Balboa composition environment [4]. It has three parts: a component integration language (CIL); a set of C++ component libraries; and a set of split-level interfaces (SLIs) to link the two. The Balboa uses a custom interpreted language (CIL) for both instantiating and connecting the components. After that, a C++ component can be manipulated through an SLI wrapper, which is implemented with the BALBOA interface definition language (BIDL), or interact directly with other components. In contrast with CCF, the Balboa uses three different languages to implement the dynamic evolution. Moreover, the Balboa performance using SLI is impacted, since the language must be interpreted.

The framework proposed in [13] uses the idea of proxy to eliminate references between C++ implementation modules. It is divided into two parts. The first part is the dynamic configuration service, which includes a module-proxy. The dynamic modules, which consist of module-implementations, constitute the second part. The module-implementations are the upgradeable C++ components, which can be disabled, enabled, loaded, unloaded, and hot swapped. The module-proxy is used as a port forwarding all the incoming requests to the right module-implementation. For this work, there is no mechanism to provide recursive composition and dynamic composition is only allowed for the part of dynamic modules.

The programming framework for autonomic applications called Accord[9] also allows composition of C++ elements at runtime through definition of rules. The composition manager injects interaction rules defined by users into corresponding element manager which then executes the rules to appropriately configure the C++ element and establish interaction with other elements. Although the Accord allows replacing of C++ elements, the elements being replaced do not answer the requests nor store them to forward to the new elements. Furthermore, recursive composition is not possible in Accord.

6. Concluding Remarks

This paper presented a C++ framework for developing component based software supporting dynamic unanticipated evolution, called CCF. CCF is an implementation of the COMPOR Component Model Specification (CMS). Such a specification provides mechanisms to make unpredicted changes in applications, even at runtime.

We described the framework design and the implementation issues. An application implemented in C++ using the proposed framework, called DjVu Compactor, was also presented. CCF allows DjVu Compactor to be evolved at runtime, more specifically, it makes possible to insert and remove DjVu encoders/decoders implementations on the fly.

As future work, we plan to develop applications in the domain of pervasive computing. More specifically, we are

[1]http://djvu.sourceforge.net/

[2]http://www.maemo.org/

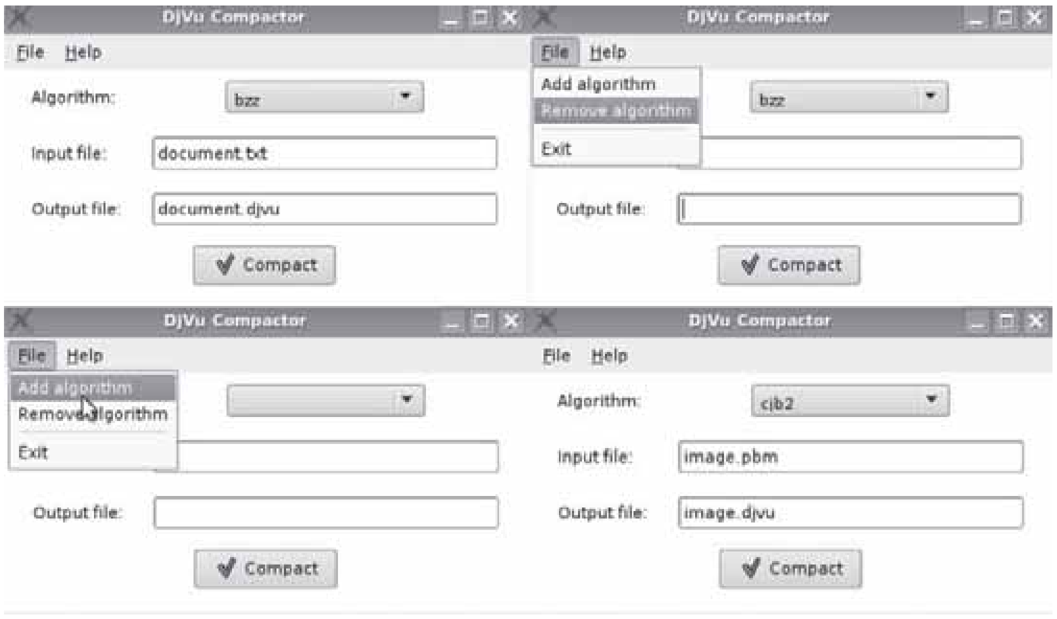


Figure 8. DjVu Compactor dynamic evolution

working on a C++ component based middleware for providing services in pervasive environments.

References

[1] H. Almeida, G. Ferreira, E. Loureiro, A. Perkusich, and E. Costa. A Component Model to Support Dynamic Unanticipated Software Evolution. In *Proceedings of International Conference on Software Engineering and Knowledge Engineering*, volume 18, pages 262–267, San Francisco, USA, 2006.

[2] A. W. Brown. *Large-Scale, Component Based Development*. Prentice Hall, 2000.

[3] W. Cazzola, R. J. Stroud, and F. Tisato. *Reflection and Software Engineering*. Springer, 2000.

[4] F. Doucet, S. Shukla, R. Gupta, and M. Otsuka. An Environment for Dynamic Component Composition for Efficient Co-Design. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 736–743, Paris, France, 2002. IEEE Computer Society.

[5] L. Erlikh. Leveraging Legacy System Dollars for E-Business. *IEEE IT Professional*, 2(3):17–23, 2000.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, 1995.

[7] P. Haffner, L. Bottou, P. G. Howard, and Y. LeCun. DjVu: Analyzing and Compressing Scanned Documents for Internet Distribution. In *Proceedings of the Fifth International Conference on Document Analysis and Recognition*, Washington, DC, USA, 1999. IEEE Computer Society.

[8] B. P. Lientz and E. B. Swanson. *Software Maintenance Management*. Addison-Wesley, 1980.

[9] H. Liu and M. Parashar. Accord: a programming framework for autonomic applications. In *Systems, Man and Cybernetics, Part C: Applications and Reviews, IEEE Transactions*, volume 36, pages 341–352. IEEE Computer Society, 2006.

[10] B. Nichols, D. Buttlar, and J. P. Farrell. *Pthreads Programming: A POSIX Standard for Better Multiprocessing*. O'Reilly & Associates, 1996.

[11] S. Roiser. The SEAL C++ Reflection System. In *Computing in High Energy and Nuclear Physics (CHEP) Conference*, Interlaken, Switzerland, 2004.

[12] J. Vlissides, J. Coplien, and N. Kerth. *Pattern Languages of Program Design 2*. Addison-Wesley, 1996.

[13] L. Yu, G. C. Shoja, H. A. Mller, and A. Srinivasan. A Framework for Live Software Upgrade. In *Proceedings of the 13 th International Symposium on Software Reliability Engineering (ISSRE)*, pages 149–158, Annapolis, MD, USA, 2002. IEEE Computer Society.

Representing Design Rationale to support Reuse

Adriana Pereira de Medeiros and Daniel Schwabe

Dept. of Informatics, PUC-Rio
Rua Marquês de São Vicente 225,
22453-900, Rio de Janeiro - RJ, Brasil
{adri, dschwabe}@inf.puc-rio.br

ABSTRACT

Many engineering areas benefit from design reuse, and it is widely accepted that software engineering should be no exception. However, design reuse requires knowledge about the "why" and "how" an artifact was designed the way it was. Design Rationale provides such information. This paper presents Kuaba, a new design rationale representation approach, and the architecture of an integrated design environment to support recording design rationale, as well as use of design rationale to support reuse of model-based designs, particularly software design. The Kuaba approach uses a formal representation language and takes advantage of the formal semantics of artifacts, as defined by the design methods, for representing design rationale. This representation approach enables a new type of design reuse, where existing artifacts rationales are integrated and re-employed in designing a new artifact.

1. INTRODUCTION

Design reuse means reusing the knowledge invested in a design, so that the proposed solutions for an artifact can be reused in designing other artifacts. Therefore, for reusing an existing design it is necessary to know "why" and "how" the produced artifacts were designed. Design Rationale (DR) [5] provides such information, since it records the reasons behind the design decisions, including also design alternatives considered and eventually rejected, the tradeoffs evaluated and the argumentation that led to the decisions. Reusing design without understanding DR may cause defects, since the modifications may destroy the carefully elaborated solutions in the existing design.

Although DR has a potential value for the design reuse, it has not been very used during software design. One of the reasons, despite much research, is the time consumption and the cost generally required for the capture and representation of DR. Another reason is the lack of a representation approach that enables the development of an integrated tool that can function as the capture, representation and use of design rationale part of the software design process.

It is fair to say that reuse at the highest abstraction level occurs when rationales are integrated and re-employed when designing a new artifact. This type of reuse can be achieved by a formally defined DR representation, which integrates the formal semantics provided by the meta-model that describes the artifacts being designed. Starting with existing artifacts, the designer can analyze their rationales and decide to integrate or extend them to get a more complete design solution, generating a new DR. From this point of view, both software maintenance and evolution can be considered simply as a continuation of a previous design process, captured in a given DR.

This paper first briefly describes the Kuaba[1] approach [7], a new DR representation approach, and how it integrates a DR representation model with the formal semantics of the artifacts provided by the design meta-models for recording DR. Next, we present the conceptual architecture of the integrated design environment that is being built to support designers through processing of formal DR representations. Finally, we conclude by discussing related work and drawing some conclusions.

2. THE KUABA APPROACH

Although there are several different approaches for representing DR, such as IBIS [4] and DRL [6], most of them generate incomplete or informal representations, not enabling the effective DR use in the design of new artifacts. Furthermore, when applying them to formally defined artifacts (such as software artifacts), their informality prevents automatically taking into consideration alternatives prescribed by the design methods, as well as incorporating their restrictions. In other words, it is not possible to leverage the semantics of the artifact provided by the formal model that describes it.

For many design domains, particularly in software design, the artifacts (models) are built according to the semantics provided by the meta-model of the modeling language or design method used to guide the design process. We call this special kind of design domain "model-based design", in which design can be seen as an instantiation process of a

[1] "Kuaba" means "knowledge" in Tupy-guarany, the language of one of the native peoples in Brazil.

meta-model. This meta-model represents the formal models that describe the modeled artifacts and provide semantic descriptions which allow reasoning over the artifacts being produced. An example of such a formal model is the meta-model of the Unified Model Language (UML) [10].

The Kuaba approach proposes the use of the artifact's formal semantics, as provided by the meta-model of the design methods (or modeling language). This is achieved through the instantiation of the DR representation model described by the Kuaba ontology [8]. This approach allows the creation of a semi-automated process to represent DR. In this process, the integrated design environment uses the formal model for the artifact being designed to suggest design options at each step in the design, and to record the corresponding choices made by the designer, using the Kuaba ontology vocabulary. This vocabulary is described in F-logic [3], a formal language for ontology specification, which permits the definition of a set of rules and computable operations to support the reuse of designs by the processing and integration of their rationales. Figure 1 shows the elements of this vocabulary, using an UML-like notation to help visualization; some relations and constraints were omitted to simplify the presentation.

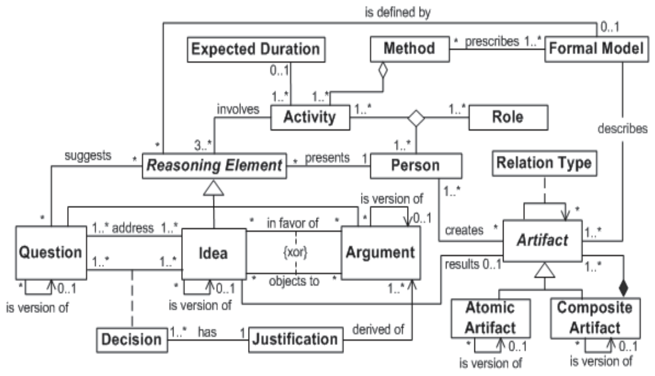


Figure 1. Kuaba Ontology Vocabulary

Briefly described, the Kuaba ontology vocabulary represents the reasoning elements used by the designers during the design; the decisions made by them; information about the artifacts that results from these decisions; and information about the formal models used to specify the produced artifacts. The reasoning elements represent the design questions that the designer should deal with, the possible solution ideas for these problems and the arguments against or in favor of the presented ideas. In Kuaba, some of these elements (questions and ideas) are instantiated according to the artifact formal model prescribed by the (software) design method.

2.1 The Kuaba Approach in Use – An Example

Consider the following scenario: a designer needs to model the navigation that will be used by users in a Web application of a CD Store and decides to use the Object Oriented Hypermedia Design Method (OOHDM) [11] to guide his design. When the designer chooses this design method, he indirectly determines the formal model(s) that will be used to describe the artifacts. This formal model specifies, to a great extent, the questions and ideas that the designer can propose, since they are pre-defined by this model. Figure 2 shows part of the formal model prescribed by the OOHDM method to specify the navigation of a Web application.

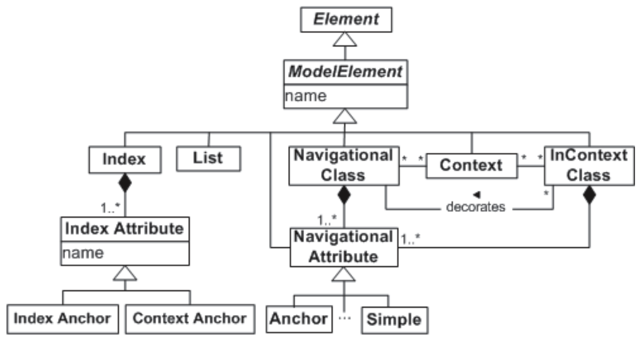


Figure 2. Partial OOHDM formal model for Navigation

According to the OOHDM formal model, an element in a navigation model can be an index, a list, a navigational class, a context or an InContext class. Contexts define the set of navigation objects that will be explored by the user at each moment. These objects are defined based on the navigational classes related to the context. The access to a context object can be achieved through an index or through an anchor. An index is a set of ordered objects, where each object has at least an attribute of the type anchor (*seletor*). For example, the "*CDs of the artist Daniela Mercury*" context represents the set of CDs that will be accessible to the user, after he has selected his favorite artist (*Daniela Mercury*) in a index. Figure 3 shows the context schema that represents this navigation. Note that the "*CDs by Artist*" context includes a group of possible contexts, whose objects are defined by the name of the artist selected by the user in the "*Artists*" index.

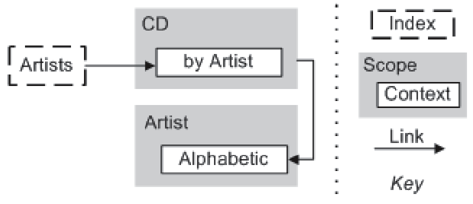
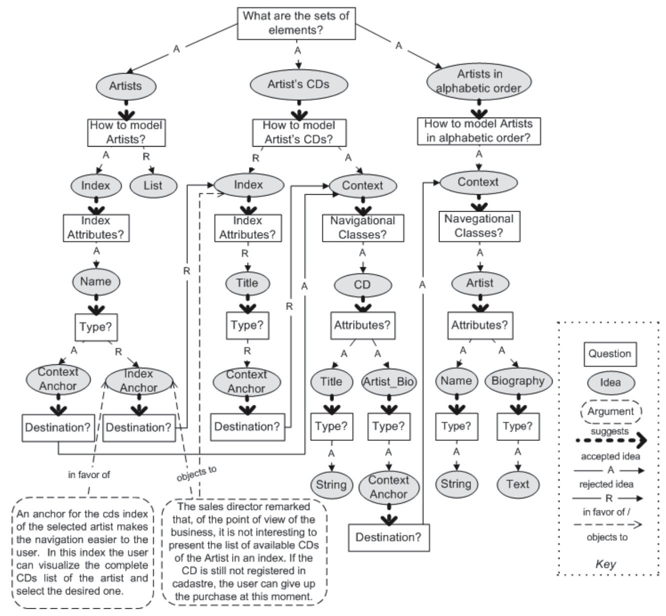


Figure 3. Context Schema Example.

Figure 4 shows the portion of the DR regarding the alternatives and decisions that resulted in the artifact shown in Figure 3. It is a graphical representation we have created to help visualizing instances of the Kuaba ontology (DRs). In this representation, the root node is an initial question (represented as rectangle), "*What are the sets of elements?*", which is addressed by the ideas "*Artists*", "*Artist's CDs*" and "*Artist in alphabetic order*", represented as ellipses. Notice that these values are determined by the designer's knowledge of the domain, or were extracted from the DR of a previous phase, requirements elicitation, which is not addressed in this paper.

Once these initial ideas for the sets of elements have been established, the designer must decide how each one of them will be modeled using the primitives of the OOHDM method to make up his final navigation model. This next step is represented in Figure 4 by the "suggests" relation, which determines questions of the type "*How to model 'set'?*" entailed by ideas. The possible ideas that address these questions are determined by the OOHDM formal model (Figure 2) for defining navigation models – sets of elements can essentially be modeled as an index, a context, or a list. Accordingly, the OOHDM formal model is instantiated into the "*Index*" and "*Context*" ideas linked to the "*How to model Artist's CDs?*" question.


An anchor for the cds index of the selected artist makes the navigation easier to the user. In this index the user can visualize the complete CDs list of the artist and select the desired one.

The sales director remarked that, of the point of view of the business, it is not interesting to present the list of available CDs of the Artist in an index. If the CD is still not registered in cadastre, the user can give up the purchase at this moment.

Figure 4. DR representing the navigation in the Figure 3

Observe that when the designer considers the ideas of modeling the sets "*Artists*" and "*Artist's CDs*" as indices, the questions "*Index Attributes?*" are immediately suggested. This occurs because an index must be associated to one or more attributes, according to the OOHDM formal model. These questions are addressed by the ideas that correspond to the respective attributes of these indices. Notice that these attributes can be of type "*Index Anchor*" or "*Context Anchor*" with different destinations, depending on the navigation that the designer desires. The proposed ideas for the type of the attributes *"Name"* and "*Title"* of the indices "*Artists*" and "*Artist's Cds*", respectively, and the ones that address the question "*Destination?*" in Figure 4, define two different solutions for the navigation design in the Web application being developed. The first solution leads from the "*Artists*" index, through the "*Name*" anchor, to the "*Artist's CDs*" context. This alternative was accepted by the designer (labels "A" on the arrows between questions and ideas), corresponding to the solution represented in the context schema in Figure 3. Observing the rationale of this solution (Figure 4), we can perceive that the designer decided to include the "*Artist_Bio*" attribute of type "*Context Anchor*" in the CD navigational class. This anchor allows the user to navigate from the information of a CD to the artist's biography in the "*Artists in alphabetic order*" context. In the second navigation solution, rejected by designer (labels "R"), the user selects the desired artist in the "*Artists*" index to access a list of CDs of this artist, in the "*Artist's Cds*" index. Then, he could select the desired CD in this index to access its detailed information, in the "*Artist's Cds*" context. Observe that the idea "*Artist's CDs*" had two design alternatives (index and context) and both could have been accepted by the designer as solutions to model this set.

This example illustrates how the formal model "drives" the instantiation of the Kuaba ontology, recording the DR. This formal model is used by the support environment proposed in this work to suggest to the designer the possible Kuaba ontology instances that must be defined at each step of the design process. After defining the design options for each set, the designer has to record the arguments for and against each option (dashed rectangles in Figure 4), and decide which design solution will be used in the final artifact.

3. DESIGN REUSE WITH RATIONALE

Formalizing DR representation using the model defined by Kuaba enriched with the artifact formal model semantics allows a new kind of design reuse. This reuse is possible through the integration of existing DRs to start a new artifact design. This integration involves matching Kuaba ontology instances (DRs) to compose a more complete design solution. Figure 5 shows the conceptual architecture of an integrated design environment that supports recording DR, as well as use of DR to support design reuse.

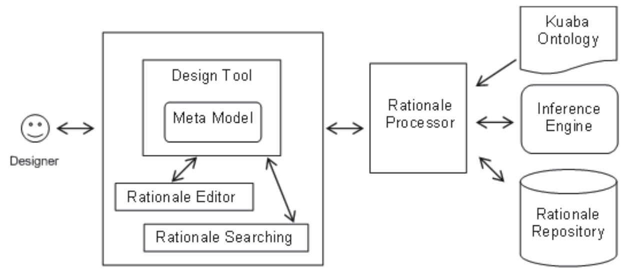


Figure 5. Architecture of an Environment Supporting DR

Since most software design support tools already use some kind of formal description of the artifacts being designed, we propose to extend them to allow their integration with the DR "processor" that is capable of processing representations using Kuaba. The extension enriches the design tools by adding two layers to support the editing and searching of DR. In the editing layer, the designer informs the arguments for and against the design alternatives considered, and the justifications for the decisions made. In the search layer, he searches existing designs with their rationales, formulates questions about the designs found, and starts the integration of rationales. In this layer the designer can also graphically visualize the rationale of the artifact being designed, or the rationale of the designs being reused in his design. These layers are being developed initially for HyperDE [9], a support environment to the hypermedia applications design using the OOHDM method.

In the proposed architecture, the design tool transfers the design options and the rationale information provided by the designer to the rationale processor, responsible for creating the DR representations and processing the rationales integration, when requested by the designer. The representation and the integration of DR involve different types of operations. Currently, the rationale processor implements the complete union of two DR representations. This operation consists of a set of rules implemented in the Flora-2 language [2] that translates F-Logic in tabled Prolog code and processes this code in the XSB [12] deductive system.

3.1 Integrating Design Rationale

Consider again the scenario in which the designer needs to model the navigation of a Web application for a CD store. Since the online (CD) stores domain is common in software design, the designer decides to perform a search for existing designs, trying to find similar artifacts, before he begins a new design. As a result, he finds the artifacts shown in Figures 3 and 6. These artifacts represent different design solutions to model the navigation using OOHDM.

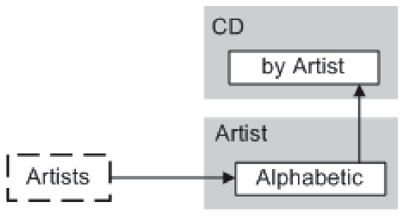


Figure 6. Context schema of a navigation solution

Analyzing the artifacts found, the designer notices that in the navigation solution of the second artifact (Figure 6), initially the user selects the favorite artist in the "*Artists*" index to access information of this artist in the "*Artist Alphabetic*" context. From this information he can navigate to the "*CDs by Artist*" context to have access to the detailed information of that CD. Observing the rationale of this navigation solution (Figure 7) we can notice that its author decided to include the "*Artist's CD*" InContext class for the "*Artists Alphabetic*" context. This InContext class defines an attribute of the type "*Context Anchor*" which will be accessible to the user only when he visualizes an artist's information in the context "*Artists Alphabetic*".

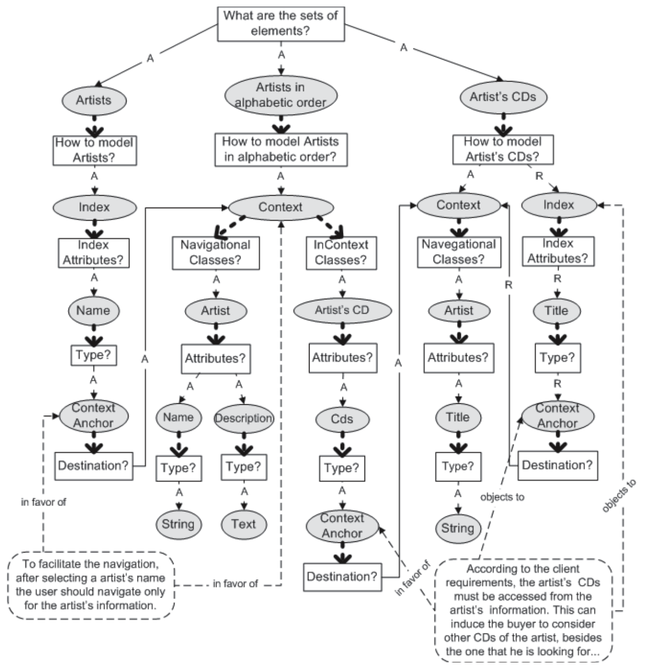

What are the sets of elements?
A
Artists
Artists in alphabetic order
Artist's CDs
How to model Artists?
How to model Artists in alphabetic order?
How to model Artist's CDs?
R
Index
Context
Context
Index
Index Attributes?
Navigational Classes?
InContext Classes?
Navegational Classes?
Index Attributes?
Name
Artist
Artist's CD
Artist
Title
Type?
Attributes?
Attributes?
Attributes?
Type?
Context Anchor
Name
Description
Cds
Title
Context Anchor
Destination?
Type?
Type?
Type?
Type?
Destination?
in favor of
String
Text
Context Anchor
String
objects to
objects to
To facilitate the navigation, after selecting a artist's name the user should navigate only for the artist's information.
in favor of
Destination?
in favor of
According to the client requirements, the artist's CDs must be accessed from the artist's information. This can induce the buyer to consider other CDs of the artist, besides the one that he is looking for...

Figure 7. DR representing the navigation in the Figure 6

After analyzing the DRs of these artifacts, shown partially in Figures 4 and 7, the designer decides to integrate the navigation solutions used by other designers to start his design with a bigger set of options. The integration of these different solutions is possible only because the artifacts (Figures 3 and 6) are built based on the same type of formal model (OOHDM navigation) to design the same domain (CD store) and their rationales are represented as instances of the same ontology (Kuaba).

The integration of two DR representations involves: the definition of the representation that will be used as the basis for the integrated DR; the equality specification for the domain ideas and questions of the two representations; and the union of the sub-trees of elements (equivalent or not) of these ideas. The base representation definition and the equality specification are performed by the designer. Based on this information, the rationale processor applies equivalence rules and performs operations to process the DRs. Figure 8 shows the integrated DR, which has the DR shown in Figure 4 as base. Observe that the arguments presented by the designer to the "*Context Anchor*" idea in the DR shown in Figure 7 are copied to the sub-tree of its equivalent idea in the base representation. The idea of modeling the "*Artist's CD*" InContext class used in the navigation solution shown in Figure 6 is also copied.

In integrated DR, all navigation options considered in the artifacts shown in Figures 3 and 6 were joined into one unique DR representation. Notice that the decisions made for the reused artifacts are not incorporated to this representation. This reflects the fact that this integrated DR represents a new design, in which the designer can do modifications and make new decisions according to his objectives.

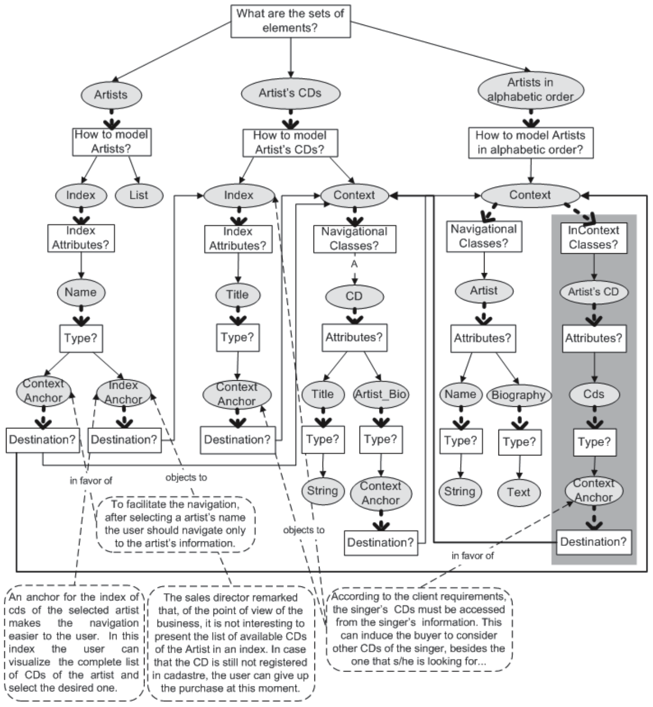

What are the sets of elements?
Artists
Artist's CDs
Artists in alphabetic order
How to model Artists?
How to model Artist's CDs?
How to model Artists in alphabetic order?
Index
List
Index
Context
Context
Index Attributes?
Index Attributes?
Navigational Classes?
Navigational Classes?
InContext Classes?
A
Name
Title
CD
Artist
Artist's CD
Type?
Type?
Attributes?
Attributes?
Attributes?
Context Anchor
Index Anchor
Context Anchor
Title
Artist_Bio
Name
Biography
Cds
Destination?
Destination?
Destination?
Type?
Type?
Type?
Type?
Type?
in favor of
objects to
String
Context Anchor
String
Text
Context Anchor
To facilitate the navigation, after selecting a artist's name the user should navigate only to the artist's information.
objects to
Destination?
Destination?
in favor of
An anchor for the index of cds of the selected artist makes the navigation easier to the user. In this index the user can visualize the complete list of CDs of the artist and select the desired one.
The sales director remarked that, of the point of view of the business, it is not interesting to present the list of available CDs of the Artist in an index. In case that the CD is still not registered in cadastre, the user can give up the purchase at this moment.
According to the client requirements, the singer's CDs must be accessed from the singer's information. This can induce the buyer to consider other CDs of the singer, besides the one that s/he is looking for...

Figure 8. Example of an integrated DR

4. RELATED WORK

Considering the DR approaches for software design, Kuaba is similar to the Potts and Bruns model, that was extended by [6] in the creation of DRL. Both integrate the DR representation with software design methods semantics. But, they differ in the way they use this semantics. In Potts and Bruns, the generic model entities are refined to accommodate a particular design method's vocabulary for deriving new artifacts. In the Kuaba approach the meta-model elements prescribed by the design method is used in the instantiation of the reasoning elements.

The SEURAT (Software Engineering Using RATionale) system [1] has architecture similar to the architecture of the integrated support environment presented in this work. However, SEURAT supports the use of rationale only to identify inconsistencies during the software maintenance process. Moreover, SEURAT does not consider computable operations over the DR to support reuse.

5. CONCLUSION

To permit a more effective use of DR to support design reuse, we have proposed the use of the artifacts formal semantics integrated to the representation model defined in Kuaba ontology for representing DR. Design reuse is supported by the processing of the recorded DR in the new artifacts production, using an integrated design environment. Using the artifacts formal models in the DR representation, the Kuaba approach makes the rationale more specific according to the software design methods and permits new uses for DR, such as the DR integration for designing new artifacts. We believe that the use of formal models of artifacts can also facilitate the DR capture, since it permits automating part of generation of DR representations. Therefore, the large amount of data produced in DR representations of actual designs is significantly hidden from the designer through the use of automated support.

REFERENCES

1. Burge, J., Brown, D.C. An Integrated Approach for Software Design Checking Using Rationale, Design Computing and Cognition 2004, Kluwer Academic Publishers (2004), 557-576.
2. Flora-2 Language. http://flora.sourceforge.net/
3. Kifer, M., Lausen, G.: F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance and Scheme. ACM SIGMOD May (1989) 134-146
4. Kunz, W., Rittel, H. W. J.: Issues as Elements of Information Systems. Institute of Urban and Regional Development Working Paper 131, Univ of California, Berkeley, CA, (1970)
5. Lee, J. Design Rationale Systems: Understanding the Issues. IEEE Expert Volume 12, No. 13, 1997, 78-85
6. Lee, J. Extending the Potts and Bruns Model for Recording Design Rationale. In Proceedings of the 13th International Conference on Software Engineering, Austin, TX (1991) 114-125
7. Medeiros, A. P. Kuaba: An Approach for Representation of Design Rationale for the Reuse of Model-based Designs. Rio de Janeiro, 2006. 149p. DSc. Thesis – Dept. of Informatics, PUC-Rio.
8. Medeiros, A. P., Schwabe, D., Feijó, B.: Kuaba Ontology: Design Rationale Representation and Reuse in Model-Based Designs. Proc. ER 2005, LNCS 3716, 241-255.
9. Nunes, D. A. and Schwabe, D. Rapid prototyping of web applications combining domain specific languages and model driven design. In Proceedings of the WWW, 2006. ACM Press, New York, NY, 889-890
10. OMG: Unified Modeling Language Specification version 1.5. March (2003)
11. Schwabe, D., Rossi, G. An object-oriented approach to Web-based application design, Theory and Practice of Object Systems (TAPOS), 1998, 207-225.
12. XSB System. http://xsb.sourceforge.net/

Telling Stories about System Use: Capturing Collective Tacit Knowledge for System Maintenance

Adriana Cristina de Oliveira[1], Renata Mendes de Araujo[2], Marcos R.S. Borges[1]
[1]*Graduate Program in Informatics, IM&NCE, UFRJ, Brazil*
adricris@gmail.com; mborges@nce.ufrj.br
[2]*Department of Applied Informatics, UNIRIO, Brazil*
renata.araujo@uniriotec.br

Abstract

One of the great system engineering challenges has been how to understand user needs. The evolution of software requirements methodologies and techniques has been motivated by the belief that understanding business, client and user needs is a key activity for successfull system deployment and use. During maintenance, system users are again the most suitable actors able to recognize special kind of information about system use, usually hidden from the system maintenance team. Benefits and problems of using the system in ther daily activities, opinions, suggestions of improvement, beliefs, 'shortcuts' for greater productivity while using the system etc, usually remain as user tacit knowledge. This article presents the use of Group Storytelling as a knowledge capture and sharing technique, intending to use the colaborative creation of stories about the use of information systems as a way to assist software maintenance team activities.

Keywords: system maintenance, knowledge capture, group story telling.

1. Introduction

Let us start this paper by telling you a brief story: *'Lucy is a teacher at the Computing Department of the Somewhere University. The department has a supporting team that takes care of the IT infrastructure, including mail servers and accounts. The department and the supporting team invest greatly on open source technology, having installed a version of a well-known webmail for researcher and student use.*

Lucy faces many problems using the webmail, and has frequently reported them to the supporting team by means of the helpdesk system available for that purpose. One of these problems, however, seemed very atypical, with the supporting team not being able to understand and solve the problem, as nobody else in the Department had reported it and the cause was not apparent. What the supporting team did not know was that other users had reported to Lucy they had faced the same problem once. What the supporting team did not know was that Lucy was probably one of the most frequent webmail users, - all day long - differently from the majority of users, who used it much occasionally. Her chances of facing unusual problems were much higher. Something about her daily working context, about the way she used the tool and about her interaction with her colleagues, could bring the supporting teams valuable information about how to recognize and handle the problem as well as improve system use. How could the supporting team collect all this information?"

Systems users increasingly build knowledge about their experiences on using the system which is not often documented [1]. Maintenance teams, in turn, deal with system errors reports and suggestions for functionality evolution and adaptation. However, they lack understanding about the way systems are actually used, about the impacts that they have on users' daily work, and about how users overcome problems and share among themselves, ways for improving system use in their daily work. Maintenance teams usually focus on dealing with errors, changing code, testing and updating system documentation and do not have the opportunity to observe and obtain knowledge about the way systems are used.

This paper suggests the use of the Group Storytelling technique for knowledge capture as an approach for the collective creation of information systems use stories in organizations. It also suggests how maintenance teams can use these stories to support their activities.

The paper is organized as follows: Section 2 presents a summary of related work about approaches for knowledge management in maintenance activities; where the need for addressing the tacit knowledge of system users is put into evidence. Section 3 further discusses the issue of capturing user tacit knowledge. Section 4 presents the Group StoryTelling technique, while in Section 5 it is shown how this technique can be used to capture user experiences. Section 6 presents the tool called Feedback, designed according to this proposal. Section 7 presents some case studies using Feedback and, finally, Section 8 presents our conclusions.

2. Approaches for knowledge management in system maintenace

Basili, Caldiera e Rombach [1] defined the Experience Factory approach for organizing and reusing software development experiences, enforcing organizational learning. An Experience Factory is a knowledge repository continuously fed by information gathered during the development process. The Experience Factory in turn, processes all the information gathered from different projects and provides feeback while facing a specific activity during the development process.

Valett, Condon, Briand e Basili [11] proposed an Experience Factory focused on system maintenance for the FDD (Flight Dynamics Division) / GSFC (Goddard Space Flight Center). This Experience Factory is similar to the orginial proposal described above although having some particular characteristics, such as: feedback analysis must be shorter since the maintenance cycles are shorter than development cycles; experiences should include previous experiences of a same project, since the focus on process evolution is stronger during maintenance compared with the development process.

Souza, Anquetil e Oliveira [8] proposed the Postmortem Analysis (PMA) technique as an approach to manage maintenance knowledge. Three aspects were considered: when the technique should be used, which knowledge should be used and how to extract knowledge from project participants.

Dias, Anquetil e Oliveira [5] defined an ontology for system maintenace comprising five aspects: systems, computer science, the maintenance process, the organizational structure and the application domain.

Torres, Anquetil e Oliveira [10] proposed the use of Learning Stories technique to collect stories obtained at the end of a project aiming at capturing and disseminating knowledge from the different project participant viewpoints . This objective is achieved by following these steps: the project is performed; some project members are interviewed; the knowledge captured during interviews is converted into stories in a format which allows its dissemination. Later, when a new project is started, the created story can be retrieved and the team can learn about it and apply this knowledge to the new project.

3. Capturing information for system maintenance

There are many other approaches reported in literature about knowledge management for system maintenance [3] [7]. However, what can be observed is that these approaches deal with how to capture, organize, classify and reuse knowledge generated during the maintenance process and within the maintenance team. This work argues that special attention is necessary towards capturing information regarding the continuous knowledge and experiences built by system users. How is it possible to provide continuous and high interaction among users (specially when there are many) and the maintenance team, at low costs and without disturbing user work as well as maintenance team work? How to facilitate the communication process between users and the maintenance team where the latter prefers to receive an identified error or problem to correct and the former is not usually able to identify it, but has a rich story about daily work using the system?

There are ways applied by the system maintenance teams to capture user information such as questionnaires, interviews; system user committees, and even ethnographic observations. However, these approaches fail by involving a small number of possible system users, restricting its view and experiences, or by being costly, or by capturing overly strucutured information, in which users must report their experiences based on pre-defined concepts such as errors, problems or improvement suggestions.

We argue that maintenance teams should have a better understanding of the user work context and needs, where some of these needs may not even be noticed. Continuous communication among users, and between users and the maintenance team should be provided in order to allow for collective discovering opportunities for improving the system and daily work.

This work proposes an approach allowing for scalability, freedom for users to report their views about system use and the identification of implicit experiences, serving as a learning basis for maintenance teams.

4. Group StoryTelling

The Group Storytelling technique has been used in Knowledge Management to capture people's tacit knowledge in order to create collaborative stories for further analysis. People involved in a project, for example, can tell stories related to relevant events, attempting to reproduce the main facts and creating at the end, a complete story [4] [6].

The use of this technique in organizations can inprove communications and knowledge sharing. Communication in organizations tend to be very formal. Storytelling intends to engage people in a less formal interaction, using authentic language and an interesting narrative [9].

Group Storytelling is more appropriate than individual storytelling in situations where there is more than one person performing related tasks. The group can make a collective story about the activities performed by their members. All group members can present their impressions about the events of the story. At the end of the process, the knowledge generated by the stories is a combination of each participant's tacit knowledge in hypertext form [12].

The Group Storytelling technique can be used with or without a software tool. In both cases, however, we have the same objective: creating collective stories, through the collaboration of the participants in this process. Without a software tool, the group members must be together in the same place. The presence of a moderator is also necessary to moderate the interaction and to encourage participation. Individual interviews can be carried out during this process, in order to get more details about the participants of the group. However, the congregation of people in the same place and at the same time proves to be a hard and expensive task, because teams are usually geographically distributed. The use of a software tool enables Group Storytelling to be used even when people are far from one another.

Some additional advantages of a software tool are the creation of a group memory, the increase of perception and communication among participants, and the coordination of the collaborative storytelling process.

5. Telling stories about system use

Collaborative stories about software use exhibit important features to help software maintenance teams. The Group Storytelling technique can be used to capture tacit knowledge and to share this knowledge after it has been formalized. We applied this technique with these two objectives in mind.

In this work, we use the Group Storytelling approach to obtain system use stories, as experienced by system users, with relevant information for software maintenance teams. The story topics should be the work tasks supported by information systems in an organization. Information capture, storage and organization are supported by a software tool which implements the Group Storytelling technique.

The Group Storytelling dynamics proposed in this work is asynchronous and distributed. This dynamics is particularly useful in organizations where teams are geographically distributed.

5.1. Capture

Task selection. The capture stage starts with the selection of a task supported by an organization´s information system. This selection is made by the software maintenance team responsible for the coordination of the group storytelling process. The selected task will be the subject of the story.

Use definition. The group storytelling process can be used in two ways: limited use or continuous use. If the software maintenance team wishes to capture specific information about a certain system in a short period of time, and according to the kind of problem to be observed, they can choose the limited use, defining a period of use, the system users and the story objectives. Otherwise, the software maintenance team can choose continuous use, in which system users can participate at any time, and for an indefinite period.

Limited use intends to induce system users to create specific stories, in order to help solve specific software maintenance teams problems. On the other hand, the continuous use will make group storytelling as part of system user daily tasks.

Participant selection. The software maintenance team also defines the system users who will take part in the group storytelling process. If the limited use option was chosen, only the system users defined by the software maintenance team will be able to participate in story creation. If the continuous use was chosen, all system users may access the environment.

User participation. User participation happens as follows: the user selects the story in which he wishes to participate, and if his access is allowed, begins to provide information about the task subject of the selected story. The user should organize the given information in steps, so that each step represents a different stage in that task execution. For each step, the user can add contextual information and comment on other user reports. This is optional, but can help in the creation of the story context.

5.2. Retrieval

In the retrieval stage, the software maintenance team will be able to read the stories created by system users, including steps, comments, context information and user interactions during story creation. The software maintenance team can then update the stories, by adding comments and classifying the information. This classification will be done by assigning symptom descriptions, which are previously defined categories associated to words, sentences or paragraphs in the sories.

6. Feedback

According to the proposed group storytelling process, we have developed a software tool called Feedback. The main objective of the Feedback tool is to allow the collaborative creation of stories about the use of information systems in an organization, in order to help software maintenance teams. Feedback is a customization of the TellStory tool [6] which was developed using Zope – an open source plataform for building content management applications.

Software tool functionalities were developed to support the proposed process and can be divided according to users profiles. We have three user profiles: moderator, system user and maintenance team. The moderator is responsible for starting and finishing stories and for allowing or rejecting user participation. The system user is responsible for story creation, by writing and commenting steps and characters, and including documents. Finally, the maintenance team is responsible for reading the stories, identifying symptoms and commenting on the stories. To illustrate, we will use one of the case studies concerning a project management system in a government organization, named REQUEST.

Start story. In the capture stage, this functionality represents the task selection activity, the main subject of the story to be created. The moderator starts a new story, for instance, the story of the task *Tracking Project Activities – to track the development of a project*, and its purpose, such as: *Often professionals do not updated project status information. It seems that they did not created a habit to register information using the system*.

Allow/Reject user participation. The moderator allows or rejects user request to participate in story creation, according to the use definition (limited use or continuous use), selected by software maintenance team. The moderator can also invite users to participate in story creation. These functionalities represent the use definition and participant selection activities, in the capture stage.

Include step (Figure 1-a). The system user includes steps in the story, informing, for each step: title, description, environment, reasons, results and emotions. Users must be oriented to organize their steps in the same order they are performed, so as to achieve the main objective, the completion of the task and the subject of the story.

Read details. The user can read all the information related to the steps written by other users: title, description, environment, reasons, results and emotions.

Include/Read comments. The user can include and read all the comments related to the steps written by other users.

Include characters. The user can include characters in the story being created, informing: name, description, skills and task knowledge. The characters created can represent system users writing the story or other people mentioned during story creation.

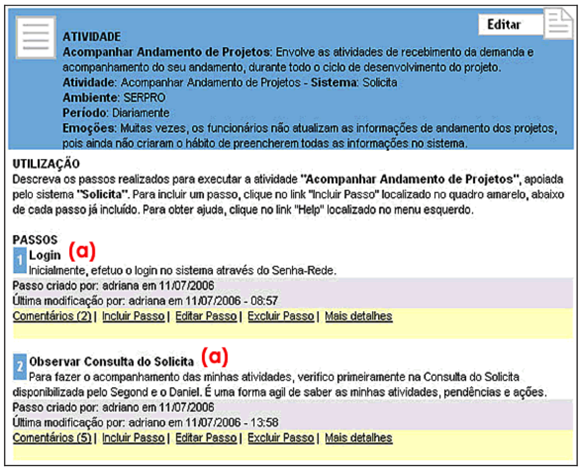
The text bellow presents an example of steps:

Step 1: Login – First of all, *I login using my account*.

Context information: Environment - *I use an Itautec desktop with Pentium IV*. Reasons: *I must login to have system acces.* Expected results: *I can have access to the system.*

Comments: User1: *the system login is suffering from a small number of user licences.* User2: *after 9 AM it is necessary to be lucky or to be patient. We have to try the login several times.*User3: *This delay demotivate us to report project status*. User 4: *I don't know why but I always succeed in loging in. Does it have something to do with my desktop or am I just lucky? When somebody is not able to login I offer my desktop and everyting works fine.*

Step 2 (provided by a different user from Step1): *My SMS – after login, I use the Personal Queries/My SMS function, where I can use a filter to select only the tasks under my responsibility…*



ATIVIDADE
Acompanhar Andamento de Projetos: Envolve as atividades de recebimento da demanda e acompanhamento do seu andamento, durante todo o ciclo de desenvolvimento do projeto.
Atividade: Acompanhar Andamento de Projetos - **Sistema**: Solicita
Ambiente: SERPRO
Período: Diariamente
Emoções: Muitas vezes, os funcionários não atualizam as informações de andamento dos projetos, pois ainda não criaram o hábito de preencherem todas as informações no sistema.

UTILIZAÇÃO
Descreva os passos realizados para executar a atividade **"Acompanhar Andamento de Projetos"**, apoiada pelo sistema **"Solicita"**. Para incluir um passo, clique no link "Incluir Passo" localizado no quadro amarelo, abaixo de cada passo já incluído. Para obter ajuda, clique no link "Help" localizado no menu esquerdo.

PASSOS
1 **Login** (a)
Inicialmente, efetuo o login no sistema através do Senha-Rede.
Passo criado por: adriana em 11/07/2006
Última modificação por: adriana em 11/07/2006 - 08:57
Comentários (2) | Incluir Passo | Editar Passo | Excluir Passo | Mais detalhes

2 **Observar Consulta do Solicita** (a)
Para fazer o acompanhamento das minhas atividades, verifico primeiramente na Consulta do Solicita disponibilizada pelo Segond e o Daniel. É uma forma agil de saber as minhas atividades, pendências e ações.
Passo criado por: adriano em 11/07/2006
Última modificação por: adriano em 11/07/2006 - 13:58
Comentários (5) | Incluir Passo | Editar Passo | Excluir Passo | Mais detalhes

Figure 1: Steps of a story.

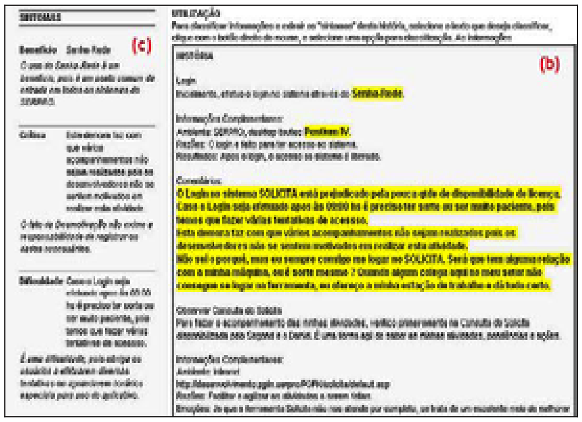
Include document. The user can include documents related to the story subject. The functionalities named C to G represent user participation in the group storytelling process, in the capture stage.

Finish story. When the moderator finishes a story, this leaves the construction phase and goes to the finishing phase. In the finishing phase, the moderator can still edit the final story text.

Read finished stories (Figure 2-b). The software maintenance team can read all the stories created by system users. This functionality represents the retrieval stage, in which software maintenance teams can extract relevant information written by system users.

Classify information. The software maintenance team can classify information from finished stories, indentifying symptoms that can be further explored as system evolutions. It is possible to select stretches of the story, classify the selection and comment on the classification. Texts marked in yellow were those selected by the maintenance team as possible symptoms to be classified (Figure 2-b).

For instance, the part of the story where one user says that he can not login after 9 AM can be classified as a *user difficulty* that the software team must find the cause. A list of symptoms is maintained by the team (Figure 2-c).



SINTOMAS

Benefício Senha-Rede

O uso da Senha-Rede é um benefício, pois é um modo comum de entrada em todos os sistemas do SERPRO.

Crítica [illegible]

O fato de Desmotivação não exime a responsabilidade de registrar os Acertos necessários.

Dificuldade Caso o Login seja efetuado após às 09:00 hs é preciso ter sorte ou ser muito paciente, pois temos que fazer várias tentativas de acesso.

É uma dificuldade, pois obriga os usuários a efetuarem diversas tentativas [illegible]

UTILIZAÇÃO

Para classificar informações e extrair os "sintomas" desta história, selecione o texto que deseja classificar, clique com o botão direito do mouse, e selecione uma opção para classificação. As informações

HISTÓRIA

Login

Inicialmente, efetuo o login no sistema através do Senha-Rede.

Informações Complementares:

Ambiente: SERPRO, desktop [illegible] Pentium IV.

Razões: O login é feito para ter acesso ao sistema.

Resultados: Após o login, o acesso ao sistema é liberado.

Comentários:

O Login no sistema SOLICITA está prejudicado pela pouca [illegible] de disponibilidade de licença. Caso o Login seja efetuado após às 09:00 hs é preciso ter sorte ou ser muito paciente, pois temos que fazer várias tentativas de acesso.

Esta demora faz com que vários acompanhamentos não sejam realizados pois os desenvolvedores não se sentem motivados em realizar esta atividade.

Não sei o porquê, mas eu sempre consigo me logar no SOLICITA. Será que tem alguma relação com a minha máquina, ou é sorte mesmo ? Quando algum colega aqui no meu setor não consegue se logar na ferramenta, eu [illegible]

Observar Consulta do Solicita

Para fazer o acompanhamento das minhas alocações, verifico primeiramente na Consulta do Solicita [illegible]

Informações Complementares:

Ambiente: Intranet

[illegible]

Razões: [illegible]

Emoções: [illegible]

Figure 2: Functionalities of Retrieval stage.

7. Case Studies

To evaluate the benefits of the proposed approach and the adequacy of the supporting tool, we have carried out three case studies in one company, the main business of it is to provide IT support for the Brazilian Government. There were seven participants – all of them have been using the system for at least one year - distributed into three groups in such a way that each person has been a member of two groups. Each group told one story.

Before the experiment, the participants received general instructions about the study and training on the Feedback tool. The initial step of each story had been created before the participants were allowed to tell their own stories. The objective of this first fragment was to provide some context and also to show the importance of the chronological order of the story fragments.

During the first stage of the case study, the participants built the collective stories. They added fragments to the story reproducing their experience with the system and their perspectives. They interacted with each other during this phase through the shared stories, being able to read other participants' fragments and comment on them. Having completed their stories, the participants answered a questionnaire about their understanding of the process, the usefulness of the tool and of the shared stories.

In the second phase, the users who had not participated in the first stage read the stories and indicated whether the stories carried relevant information about the system. They also answered a questionnaire similar to that applied to those who built the stories.

The third stage consisted of story evaluation by the analysts in charge of system maintenance. After reading the stories told by the users, the analysts added comments and pointed to what they thought were symptoms of possible problems. All three groups answered another set of questions related to their interpretation of the stories taking into account maintenance activities. We were interested in learning whether the information reported in the stories could contribute to the activities related to system maintenance. The case studies were designed to answer the following goals:

Goal 1: To evaluate whether the fragments created by system users generate a use story that can support the maintenance activity. Based on the concept of story, as defined in this work, the process suggested by our approach should result in a meaningful set of steps, and each step should contain some relevant information. In the case studies, the user narratives had these features, as confirmed by system analysts.

Goal 2: To evaluate whether collective stories can serve as technique to externalize tacit knowledge and that they were coherent and relevant to their purpose. Based on the analysis of the stories generated, we conclude that part of the knowledge was tacit, kept in system user minds, was externalized and became formal. The group storytelling approach and the Feedback tool were successful in supporting this transformation. Knowledge had been disseminated to other participants, who were able to comment and to add more relevant information to the stories.

Goal 3: To evaluate whether the knowledge obtained from the stories was useful to the system maintenance team. The symptoms identified by the maintenance team were useful to reach a common understanding of the problem. Problem classification was used as a source of information to new maintenance team members.

Goal 4: To evaluate whether users and analysts shared relevant knowledge. Users and analysts interact through the collaborative building of stories, which allowed the exchange of experiences among them. Information kept with few participants could be shared among all.

8. Conclusions

In this work we have presented an approach to assist software engineers, working in system maintenance, towards better understanding the organization and its requirements. We provide this support by capturing tacit knowledge about the actual system operation. Knowledge is represented by a set of collective stories told by those who have participated in or witnessed any event relevant to the maintenance request. The Feedback tool provides a shared space supporting the capture, storing and retrieval of this type of knowledge.

Although the knowledge management techniques had been applied before to the context of software maintenance, we believe that the group storytelling approach brought a new perspective to the tacit knowledge problem.

The issues outlined previously in this work, could be answered as follows:

How is it possible to provide continuous and high interaction among users (specially when there are many) and the maintenance team, at low costs and without disturbing user work as well as maintenance team work? The Feedback system, being distributed and assynchronous, allows scalability in capturing user stories , can be used whenever users feel able to, and serves as a knowledge repository that can be systematicaly used by the maintenance team.

How to facilitate the communication process between users and the maintenance team where the latter prefers to receive an identified error or problem to correct and the former is not usually able to identify it, but has a rich story about daily work using the system?

The Feedback tool supports the construction of collective stories as well as the association of the events represented therein with a set of symptoms of bad or wrong system behavior. This association provided a good starting point for maintenance work and tracking of previous problems. Users reported their stories in a natural way, without worrying about the need to classify their contributions and the Feedback tool provided the maintenance team with facilities to integrate different views, to analyse the information and to identfy and classify system use symptoms.

The technique and the tool were evaluated in three case studies. The initial observations have confirmed that the Feedback tool worked well and supported the group storytelling approach. The knowledge acquisition process was followed, and collaboration among participants occurred naturally, stimulating interaction. The resulting stories did capture part of the tacit knowledge and were considered useful by the maintenance team.

The work has some limitations to be addressed as future work. First, we need more experience with both the technique and with the Feedback tool, which includes performing empirical investigation. We also need a comparison between the traditional maintenance process and our proposal. A new set of functions will be added to deal with awareness and context.

10. References

[1] AURUM, A.; JEFFERY, R.; WOHLIN, C.; HANDZIC, M. Managing Software Engineering Knowledge, Springer, Germany, 2003.

[2] BASILI, V. R. ; CALDIERA, G. ; ROMBACH, H. D. Experience factory. In: MARCINIAK, John J. (Ed). Encyclopedia of Software Engineering. New York: John Wiley &Sons, 1994. v. 1, 469-476.

[3] BURGE, J.E.; BROWN, D.C. Rationale-Based Support for Software Maintenance. In: Dutoit, A.H., McCall, R., Mistrík, I., Paech, B. (eds) Rationale Management in Software Engineering, Springer, Germany, 2006, pp. 273-296.

[4] CARMINATTI, N.; BORGES, M. R. S.; GOMES, J. O. Analyzing Approaches to Collective Knowledge Recall, Computing and Informatics, 2006, Vol. 25 (6), 1001-1024.

[5] DIAS, M. G. B. ; ANQUETIL, N. ; OLIVEIRA, K., Organizing the knowledge used in software maintenance. Journal of Universal Computer Science, 2003, Vol. 9 (7), 641-658.

[6] LEAL, R. P.; BORGES, M. R. S.; SANTORO, F. M. Applying Group Storytelling in Knowledge Management. Proceedings of the International Workshop on Groupware, 2004, Costa Rica. Lecture Notes in Computer Science, 2004. Vol. 3198, 34-41.

[7] RODRÍGUEZ, O. M.; MARTINEZ, A.I.; VIZCAINO, A.; FAVELA, J.; PIATINI, M. Identifying knowledge management needs in software maintenance groups: a qualitative approach. In: Proceedings of the Mexican International Conference in Computer Science, 5., 2004, Colima. 72-79.

[8] SOUSA, K. D. ; ANQUETIL, N. ; OLIVEIRA, K. M. Learning software maintenance organizations. In: MELNIK, G. ; HOLZ, H. Advances in Learning Software Organizations, LNCS Vol. 3096, Berlin: Springer, 2004, 67-77.

[9] BROWN, J.S.; DENNING, S.; GROH, K.; PRUSAK, L. Storytelling in Organizations: Why Storytelling Is Transforming 21st Century Organizations and Management. Butterworth-Heinemann, 2004.

[10] TORRES, A. H. ANQUETIL, N. ; OLIVEIRA, K. M. Pro-active dissemination of knowledge with Learning histories. In: Proceedings of the International Workshop on Learning Software Organizations, Rio de Janeiro, Brazil, 2006, 19-27.

[11] VALETT, J.D.; CONDON, S.E.; BRIAND, L.; KIM, Y.M.; BASILI V.R. Building on Experience Factory for Maintenance, Proceedings of the Software Engineering Workshop, Software Engineering Laboratory, 1994.

[12] VALLE, C.; RAYBOURN, E.M., PRINZ, W., BORGES, M.R.S. Group storytelling to support tacit knowledge externalization. In: Proceedings of the International Conference on Human Computer Interaction, Crete, 2003. Vol. 4, 1218-1222.

[13] ZOPE.ORG – www.zope.org

Evolution and Runtime Monitoring of Software Systems

Hui Liang Jin Song Dong
School of Computing
National University of Singapore
{lianghui, dongjs}@comp.nus.edu.sg

Jing Sun
Department of Computer Science
The University of Auckland
j.sun@cs.auckland.ac.nz

Abstract

Software evolution is a critical and inevitable stage in the life cycle of all software systems. We propose an evolution technique based on aspect-oriented programming. In our technique, join points guide where the modification should be made, and inter-type declaration and advice describe the expected modification. To check whether the software system resulting from the evolution based on AOP technique behaves as expected by system requirements, we propose a runtime monitoring technique for verifying aspect-oriented programs dynamically. In our technique, the valuable information about desired dynamic behaviors of the system is extracted through animating the formal specification of the system. Meanwhile, the information about dynamic behaviors of concrete implementations of the target system is obtained through program debugging. Base on the attained information from both sides, the judgement on the consistency of the concrete implementation with the formal specification is timely made while the system is running.

1 Introduction

Software evolution is widely recognized as one of the most important issues in software development process. It is an inevitable and critical stage in the life cycle of software systems, particularly, those serving highly volatile business domains such as banking and telecommunications [15]. Generally, software evolution means to change the functionality of software according to the change of the specification/requirement or the operating environment of the software system.

Aspect-oriented programming (AOP) [11] has been proposed as a new methodology and a complement to traditional object-oriented programming (OOP) to improve the separation of concerns in software systems. The essential idea of aspect-oriented programming is that all concerns should be treated as modular units regardless of the limitations of the implementation languages. By placing these crosscutting concerns separately in an aspect, the core concerns are made more cohesive since their implementations are relieved of the burden of managing concepts unrelated to their purpose. With the effective modular decomposition, AOP facilitates the development of complex systems and make it easier to maintain the systems that have been developed.

Traditionally, AOP focuses on realizing the cross-cutting behaviors and is used to improve the separation of concerns in software systems . In this paper, we investigate how AOP can contribute to the evolution of core classes in an object-oriented programming language and propose an evolution technique based on aspect-oriented programming and design technique. Although our examples use Java and AspectJ, the idea presented in this paper can be applied in other programming contexts as well.

Furthermore, just as with the introduction of OOP, AOP not only brings a unique set of benefits; but also introduces a set of challenges, such as new problems with respect to verifiability and testability. By far, the development of verification techniques for aspect-oriented systems are still lag far behind what has been achieved for the static analysis of procedural and object-oriented programs. Because the expressive power that aspects unleash heightens the potential for insidious errors, finding cost-effective verification techniques that address aspect-oriented software is especially important in AOP.

Therefore, in this paper, we also propose a runtime monitoring technique, which is based on the animation of formal specifications, for aspect-oriented software systems. With the runtime monitoring technique, we will be capable of checking whether the software system resulting from the evolution based on AOP technique behaves as expected by system requirements which are described in a formal notation.

The rest of the paper is organized as follows. Section 2 introduces background information about AspectJ, the Z specification language, and specification animation. Section 3 presents the aspect-oriented programming based software evolution technique. Section 4 presents our runtime monitoring technique and a prototype system. Section 5 demonstrates the application of our evolution approach and runtime monitoring technique with a case study. Section 6 reviews some related work. Section 7 concludes the paper.

2 Background

2.1 AspectJ

AspectJ [1, 10, 13] is an implementation of aspect-oriented programming methodology. As an aspect-oriented extension to Java, it adds to Java some new concepts and associated constructs such as join points, pointcuts, advice, intertype declarations and aspects. *Join point* is a well-defined point in the execution of a program, such as a call to a method or an access to an attribute. *Pointcut* is a set of points that optionally expose some of the values in the execution of the join points. *Advice* is a method-like mechanism used to define certain code that executes *before*, *after*, or *around* a pointcut. *Inter-type declaration* is a static crosscutting instruction that introduces changes to classes, interfaces and aspects of the system. *Aspects* are modular units of crosscutting implementation, which encapsulate behavior and state of those crosscutting concerns whose implementations must span across the core concerns.

2.2 The Z specification language and specification animation

The Z specification language has been a widely accepted formal language for specifying the behaviours of software and hardware systems. Based on set theory and first order predicate logic, Z is a model oriented specification language. It models a system by describing its states and the ways in which the states can be changed. Actually, the Z specification language includes two parts: the mathematical language and the schema language [18]. The schema language can be used to structure and compose descriptions, making it possible to build big schemas from small ones. As an example, the Z specification in Figure 1 describes a simplified version of a bank account. It contains one state variable - *Balance*, and two operations - *Credit* and *Debit*.

Specification animation exhibits the dynamic behaviourial properties of a formal specification. It not only provides the specification designers with a way to test whether their specifications behave as expected, but also validates the behaviour of formal specifications with the end users. In the last decade, several animation tools have been developed for executing and interpreting formal specifications automatically [9, 8]. The animation tool used in our monitoring system is Jaza [17]. It is an animator for Z, which has a strong support for quantifiers and various less-often-used Z constructors(such as μ, λ, θ terms). It provides more efficient and convenient evaluation of schemas on ground data values; and it can handle not only unpredictable performance characteristics but also nondeterministic schemas.

Account
$balance : \mathbb{N}$

Init
$Account'$

$balance' = 0$

Credit
$\Delta Account$
$amount? : \mathbb{N}$

$balance' = balance + amount?$

Debit
$\Delta Account$
$withdrawamount? : \mathbb{N}$

$withdrawamount? < balance$
$balance' = balance - withdrawamount?$

Figure 1. The Z specification of a bank account

3 AOP-aided software evolution

AOP provides the basis for an effective software evolution technique because crosscutting concerns can be added without making invasive modifications on original programs. In this section, we describe an AOP-aided software evolution technique.

3.1 Determine the difference

First of all, the changes to the original system, which is corresponding to the transformation of the users' requirement, should be identified. In order to get a precise, concise and unambiguous specification of the system, the requirement is described in a formal specification language – Z notation. Given the new and old versions of the formal specification of a system, there is a function which figures out the difference between the old version and the new version. Taking the two versions of the formal specifications as inputs, the function will output the sets of removed members (fields and methods), added members, and modified members. The algorithm behind the function is shown as follows.

```
Algorithm   SpecDiff (OldSpec, NewSpec):
                    AddSV, RmvSV,MdfSV, AddOS, RmvOS, MdfOS
 input:
        OldSpec : old version of the formal specification of the system
        NewSpec : new version of the formal specification of the system
 output:
          AddSV: the set of added state variables
          RmvSV: the set of removed state variables
        MdfSV: the set of modified state variables
        AddOS: the set of added operation schemas
        RmvOS: the set of removed operation schemas
        MdfOS: the set of modified operation schemas
  global:
        ComnSV: the set of common state variables of the old and new
                version of the specification
        ComnOS: the set of common operation of the old and new
                version of the specification

  1.  ComnSV $\longleftarrow$ OldSpec.StateVariables $\cap$ NewSpec.StateVarialbes
  2.  AddSV $\longleftarrow$ NewSpec.StateVariables $\setminus$ ComnSV
  3.  RmvSV $\longleftarrow$ OldSpec.StateVariables $\setminus$ ComnSV
  7.  for each Sv $\in$ ComnSV do
  8.    opdiff $\longleftarrow$ Compare(NewSpec.Sv.Type, OldSpec.Sv.Type)
  9.    if opdiff then
  10.       MdfSV $\longleftarrow$ MdfSv $\cup$ {NewSpec.SV}
  11.   end if
  12. end for
  13. ComnOS $\longleftarrow$ OldSpec.OpSchemas $\cap$ NewSpec.OpSchemas
  14. AddOS $\longleftarrow$ NewSpec.OperationSchemas $\setminus$ ComnOS
  15. RmvOS $\longleftarrow$ OldSpec.OperationSchemas $\setminus$ ComnOS
  16. for each Op $\in$ ComnOp do
  17.   opdiff $\longleftarrow$ Compare(NewSpec.Op.Schema, OldSpec.Op.Schema)
  18.   if opdiff then
  19.       MdfOS $\longleftarrow$ MdfOS $\cup$ {NewSpec.Op.Schema}
  20.   end if
  21. end for
```

3.2 Construct aspects

After the difference between the old and new versions of the formal specification has been identified, we can construct the aspects that would modify the original system without invasive modifications on it. The approach to construct the aspect is described in the algorithm as follows:

```
Algorithm  CtrAspect (AddSV, MdfSV, AddOS, MdfOS): NewAspect
input:
      AddSV: the set of added state variables
      MdfSV: the set of modified state variables
      AddOS: the set of added operation schemas
      MdfOS: the set of modified operation schemas
output:
      NewAspect: an aspect

1.  declare NewAspect, an aspect crosscutting the original class
2.  for each Sv $\in$ AddSV $\cup$ MdfSV do
3.     inter-type declaration Dlr_sv is added to NewAspect   end for
4.  for each Os $\in$ AddOS do
5.     inter-type declaration Dlr_op is added to NewAspect   end for
6.  for each Os $\in$ MdfOS do
7.              pointcut PC_os: call(Os) is added to NewAspect
8.              around-advice AD_os is added to NewAspect
9.  end for
```

With the information about the difference between the original specification and the new specification as input, the above algorithm will construct an aspect - *NewAspect*. For each new state variable, an inter-type declaration *Dlr_sv*, which declares and initializes a new field in the original class, will be added to *NewAspect*. The modification that can be made to the state variable is the change of type. We can take it as an introduction of a new state variable. Therefore, for each modified state variable, an inter-type declaration *Dlr_sv*, which declare the corresponding class variable with the new type and initializes it, is added to *NewAspect*. For each new operation schema, an inter-type declaration *Dlr_os*, which declares a method corresponding to the new operation schema, is introduced to *NewAspect*. For each modified operation schema, *PC_os*, a pointcut which captures all the call of the corresponding methods is constructed; meanwhile, *AD_os*, an around-advice which implements the functions described by the modified operation schema is constructed and bound with the pointcut *PC_os*.

After construct the aspect, we remove class variables corresponding to state variables that are removed and modified in the new version of the specification from the original implementation. We also remove the methods corresponding to operation schemas that are removed in the new version of the specification. Then, the aspect we have constructed will be woven with the original implementation. As a result, a new version of the implementation can be generated and an AOP-aided evolution is achieved.

4 Runtime monitoring technique for AOP

After the evolution is accomplished, we need to check whether the new version of the software behaves as required by the new version of system requirements. In this section, we present a runtime monitoring technique for dynamically checking the consistency of the concrete implementation with the formal specification. The monitoring technique has been applied to object-oriented programs [14]. Now, we extend it so that it can work with aspect-oriented programs.

4.1 Overview of the runtime monitoring technique

Given concrete implementations and formal specifications, the workflow of the monitoring technique is shown in Figure 2. With the specification animator, our runtime monitoring technique gets the information about the dynamic behavioural properties of the formal specification through specification animating. And with the observer module, the information about the dynamic behaviour of the concrete implementation is gathered. Taking the execution sequences provided by the user as input, the controller module controls the specification animator and the observer module so that the concrete implementation is run in parallel with the animation of the formal specification. Mean-

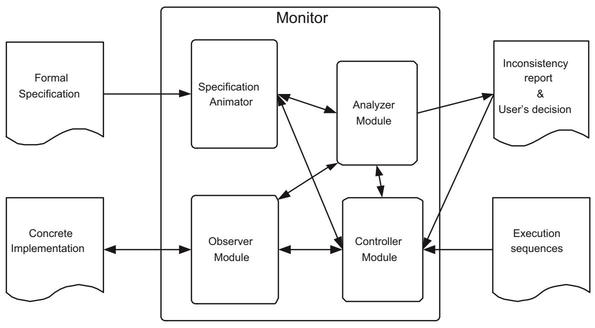


Figure 2. Runtime Monitoring.

```
public class Account {
    private int _balance;
    private int _accountNumber;
    public Account(int accountNumber)
      {_accountNumber = accountNumber;}
    public Account() { _balance = 0;}
    public void credit(int amount)
      { setBalance(getBalance() + amount);}
    public void debit(int withdrawamount)
     {
       int _balance = getBalance();
       if (_balance < withdrawamount) {}
         else {setBalance(_balance - withdrawamount);}
     }
    public int getBalance() { return _balance;}
    public void setBalance(int newbalance)
    {_balance = newbalance;}
}
```

Figure 3. Class *Account*

while, based on the information obtained from the specification animator and the observer module, the analyzer module provides judgement on the consistency of the concrete implementation with the formal specification. If any inconsistency is found, it will be reported to the user. Given the inconsistency report, the user needs to make a decision about how to deal with such an inconsistency. Then, the monitoring system will continue its work according to user's decision.

4.2 A prototype runtime monitoring system

To demonstrate our runtime monitoring technique, we have implemented a prototype runtime monitoring system. The monitoring system works with a formal specification written in Z formal language and its concrete implementation developed in AspectJ programming language. In our monitoring system, the animator used for animating the specification is Jaza [17], the observer module and controller module are implemented based on the Java Debugger API.

The monitoring system can work in two different modes, i.e., the debugging mode and the running mode. In the debugging mode, the user inputs all of the methods which are expected to be executed into the monitoring system as a whole sequence; and the system will automatically generate the sequence of corresponding running commands for the animator. Then, the system starts the dynamic checking of the behaviourial results of the implementation against the behaviourial results of the specification animation. In the debugging mode, our specification-based monitoring system can serve as an effective dynamic test execution and test result checking tool.

In the running mode, the user chooses the execution step by indicating the methods to be executed and inputting parameters (if necessary). The commands for executing corresponding operations in the formal specification will be automatically generated for the animator. The monitoring system will then check the running results of the implementation execution with the corresponding specification animation result to identify whether there is any inconsistency. In the running mode, the user indicates what will be executed next in a step-by-step manner. Thus, the system can achieve a runtime monitoring of concrete implementations against the formal specifications.

5 Case study

In this section, we demonstrate the application of our evolution and runtime monitoring techniques with the banking system presented in Section 2.2 as an example. As described by the Z specification in Figure 1, money can be withdrawn from the account only if the amount of the money to be withdrawn is less than the current balance. The Java code in Figure 3 implements such a class.

In Figure 4, the Z specification describes an account which has evolved from what is described in Figure 1. They are similar except that, in the new version, and a new state variable *minBalance* is introduced and that there is a minimum balance rule demanding that the amount of remained money in the *MBAccount* should not be less than *minBalance* at any time.

While the difference between the original and new versions of the specification has been identified, an aspect which cuts across the *Account* class and implement the minimum balance rule is constructed. As shown in Figure 5, the *MinimumBalanceRuleAspect* aspect introduces a data member *_minimumBalance* of type integer and a method for computing the available balance into the *Account* class. Meanwhile, through two pieces of advice, the *MinimumBalanceRuleAspect* aspect modifies the execution behavior of the *Account* module: 1) whenever a new object of *Account* class is created, the data member *_minimumBalance* will be initialized as 25; 2) before the execution of method *debit*, check whether the amount of remained money will be less

$$\begin{array}{|l}\hline \textit{MBAccount} \\ \textit{balance} : \mathbb{N} \\ \textit{minBalance} : \mathbb{N} \\ \hline \end{array} \qquad \begin{array}{|l}\hline \textit{Init} \\ \textit{MBAccount}' \\ \hline \textit{balance}' = 0 \\ \textit{minBalance}' = 25 \\ \hline \end{array}$$

$$\begin{array}{|l}\hline \textit{Credit} \\ \Delta \textit{MBAccount} \\ \textit{amount}? : \mathbb{N} \\ \hline \textit{balance}' = \\ \quad \textit{balance} + \textit{amount}? \\ \hline \end{array} \qquad \begin{array}{|l}\hline \textit{Debit} \\ \Delta \textit{MBAccount} \\ \textit{withdrawamount}? : \mathbb{N} \\ \hline \textit{withdrawamount}? < \\ \quad \textit{balance} - \textit{minBalance} \\ \textit{balance}' = \\ \quad \textit{balance} - \textit{withdrawamount}? \\ \hline \end{array}$$

Figure 4. The Z specification of a bank account with minimum balance rule

```
public aspect MinimumBalanceRuleAspect {
    private int Account._minimumBalance;
    public int Account.getAvailableBalance() {
       return getBalance() - _minimumBalance;
       }
    after(Account account):
    execution(Account.new(..)) && this(account) {
       account._minimumBalance = 25;
       }
    before(Account account, int amount)
     throws InsufficientBalanceException :
      execution(* Account.debit(*))
           && this(account) && args(amount) {
         if (account.getAvailableBalance() < amount) {
              throw new InsufficientBalanceException(
                      "Insufficient available balance");
             }
       }
 }
```

Figure 5. Aspect for minimum balance rule

than the *_minimumBalance* after the money is withdrawn as demanded; if yes, then an exception will be thrown. After weaving the *MinimumBalanceRuleAspect* aspect with the original class *Account*, the evolution required by the new version of specification is achieved.

Now, we try to verify the result of evolution using our runtime monitoring technique. After load the monitoring system with the new version of specification and implementation and finish configuration, we input the sequence of methods: credit(50); debit(40), and enable the monitoring system to work. When the method debit(40) is executed, the monitoring system reports an inconsistency as shown in Figure 6, informing the user that the operation *Debit* is not implemented correctly, and provides the user with two choices for what to do next. Checking the AspectJ code in Figure 5, we find that the last advice in the *MinimumBalanceRuleAspect* aspect only throws an exception

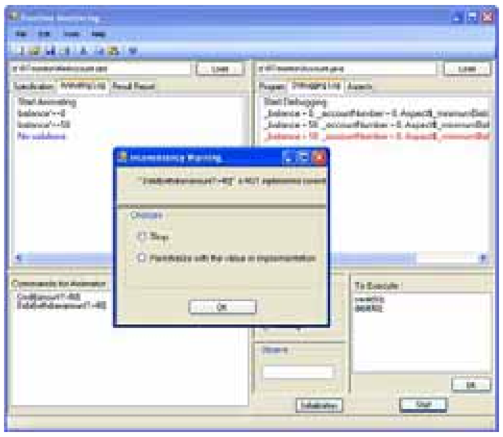


Figure 6. Runtime monitoring system in work

```
void around(Account account, int amount):
     execution(* Account.debit(*))
           && this(account) && args(amount) {
       if (account.getAvailableBalance() < amount) {}
           else{ proceed(account, amount);}
```

Figure 7. Correct advice

but does not prevent the operation debit from being invoked. It should be changed to the advice as shown in Figure 7. With the around advice, aspect modifies the execution behavior of the *Account* module in a way that the withdrawal is not allowed if the amount of remained money will be less than the *_minimumBalance* after the money is withdrawn as demanded. This is just exactly what is described by the Z specification in Figure 4. With the modified version of the aspect loaded to the monitoring system, no inconsistency is reported. It reassures the user that, after woven, the *MinimumBalanceRuleAspect* aspect and the *Account* class implements the specification as expected.

6 Related Work

There are some other researches on assisting software evolution with AOP. Greenwood *et.al* [6] proposed a framework which allows a system to be developed in a way that it is able to evolve using a combination of framed aspects and dynamic AOP. Previtali and Gross [16] presented an approach to support software evolution at run-time based on the ideas of AOP. Their approach aims to manage the update of applications that require continuous uptime such as file and authentication servers. Techniques and tools have been proposed for runtime monitoring with the purpose of detecting, diagnosing, and recovering from faults for traditional softwares. Java PathExplorer (JPaX) [7] is a runtime monitoring technique developed for sequential and concurrent Java programs. Monitoring and Checking (MaC) [12] provides a framework for runtime monitoring of real-time systems written in Java. Besides, Monitoring-oriented programming (MOP) [2, 4] is an approach that allows formal property specifications to be added on top of a target programming language and to generate monitoring code from the formal specification. A Java-MOP [3, 5] prototype has been implemented as a client-server application. In our approach, rather than generate monitoring code from the formal specification, the formal specifications of the system are animated to exhibit the expected behaviorial properties of the specified system. The information obtained from the specification animation will serve as the base for the judgement on the correctness of the concrete implementation.

7 Conclusions and future work

In this paper, we presented an AOP-aided software evolution approach. First, the old and new versions of the formal specification of a software system are compared to identify the differences. Second, aspects are constructed to achieve the expected modifications. After weaving the constructed aspects with original classes, the required evolution will be accomplished. Moreover, we present a runtime monitoring technique based on the animation of the formal specification of software systems for AspectJ programs. Our runtime monitoring technique analyzes the behavior of the target system and checks the correctness of each single execution based on the information from the animation of formal specification and the program debugging of the implementation. It can be used not only to dynamically verify the behaviors observed in the target system, but also to explicitly recognize undesirable behaviors in the target system with respect to given formal requirement specifications.

For now, we only consider the evolution of core classes in an object-oriented programming language. In the future, we will try to extend our technique so that it will be capable of handling the evolution of the cross-cutting aspects[1].

References

[1] Aspectj. http://www.eclipse.org/aspectj.

[2] F. Chen, M. D'Amorim, and G. Roşu. A formal monitoring-based framework for software development and analysis. In *Proceedings of the 6th International Conference on Formal Engineering Methods (ICFEM'04)*, pages 357–373. Springer-Verlag, 2004.

[3] F. Chen, M. D'Amorim, and G. Roşu. Checking and correcting behaviors of java programs at runtime with java-mop. In *Proceedings of Fifth Workshop on Runtime Verification, (RV'05)*, pages 3–20, 2005.

[4] F. Chen and G. Roşu. Towards monitoring-oriented programming: A paradigm combining specification and implementation. In *Proceedings of Third Workshop on Runtime Verification (RV'03)*, pages 108–127, 2003.

[5] F. Chen and G. Roşu. Java-mop: A monitoring oriented programming environment for java. In *Proceedings of the Eleventh International Conference on Tools and Algorithms for the construction and analysis of systems (TACAS'05)*, pages 546–550. Springer-Verlag, 2005.

[6] P. Greenwood, L. Blair, N. Loughran, and A. Rashid. Dynamic framed aspects for dynamic software evolution.

[7] K. Havelund and G. Roşu. An overview of the runtime verification tool Java PathExplorer. *Formal Methods in System Design*, 24(2):189–215, 2004.

[8] D. Hazel, P. Strooper, and O. Traynor. Requirements engineering and verification using specification animation. In *ASE '98: Proceedings of the Thirteenth IEEE Conference on Automated Software Engineering*, page 302. IEEE Computer Society, 1998.

[9] M. A. Hewitt, C. O'Halloran, and C. T. Sennett. Experiences with PiZA, an Animator for Z. In *ZUM '97: Proceedings of the 10th International Conference of Z Users on The Z Formal Specification Notation*, pages 37–51. Springer-Verlag, 1997.

[10] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold. An overview of AspectJ. In *ECOOP '01: Proceedings of the 15th European Conference on Object-Oriented Programming*, pages 327–353. Springer-Verlag, 2001.

[11] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242, 1997.

[12] M. Kim, M. Viswanathan, S. Kannan, I. Lee, and O. Sokolsky. Java-MaC: A run-time assurance approach for Java programs. *Formal Methods in System Design*, (2):129–155, 2004.

[13] R. Laddad. *AspectJ in Action: Practical Aspect-Oriented Programming*. Manning Publications Co., Greenwich, CT, USA, 2003.

[14] H. Liang, J. S. Dong, J. Sun, R. Duke, and R. E. Seviora. Formal specification-based online monitoring. In *Proceedings of the 11th IEEE International Conference on Engineering of Complex Computer Systems*, pages 152–160, 2006.

[15] T. Mens, J. Buckley, M. Zenger, and A. Rashid. Towards a taxonomy of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice (Special Issue on USE)*, 17(5), 2005.

[16] S. C. Previtali and T. R. Gross. Dynamic updating of software systems based on aspects. In *ICSM '06: Proceedings of the 22nd IEEE International Conference on Software Maintenance*, pages 83–92, 2006.

[17] M. Utting. Data structures for Z testing tools. In *Proceedings of FM-TOOLS*, 2000.

[18] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof.* Prentice-Hall International, 1996.

[1]More detailed information about the techniques presented in this paper can be found at http://www.comp.nus.edu.sg/∼lianghui/thesis.pdf.

On Modern Debugging For Rule-Based Systems

Valentin Zacharias
FZI Karlsruhe
Haid-und-Neu Strasse 10-14
76131 Karlsruhe, Germany
zacharias@fzi.de

Andreas Abecker
FZI Karlsruhe
Haid-und-Neu Strasse 10-14
76131 Karlsruhe, Germany
abecker@fzi.de

Abstract

With the growing interest in rule languages in the Semantic Web and the Business Rule community it is time to look again at the issue of debugging rule bases. New challenges have arisen since the time most concepts for today's debuggers where created: end user programming has grown in importance, graphical editors have become more common and programs are increasingly interconnected. Today there is no debugger for rule-based systems that takes into account the declarative nature of rules and that addresses these challenges.

This paper proposes Explorative Debugging as a paradigm for building debuggers that truly take into account the declarative nature of rules. The Inference Explorer is presented as an implementation of the explorative debugging ideas.

1. Keywords

Rule-based systems, debugging, F-logic, explorative debugging

2. Introduction

Locating bugs is an inevitable part of any software development activity. Technical support for the finding of bugs can speed up the debugging process tremendously - good debugging support is an important ingredient for the efficient creation of software. These statements are equally true for all programming paradigms, debugging may look different and may be needed less often for rule-based systems, but it is just as indispensable as it is for imperative programming.

Recently, with the large scale practical use of business rule systems [20] and the interest of the Semantic Web community in rule languages [12] there is an increasing need for tools supporting the development of rule-based systems. This alone would warrant a fresh look at the debugging of rule-based systems. Moreover, the computer science field has changed considerably since the days when most of the concepts for debuggers for rule-based systems where created; in particular the development of rules for the Semantic Web is posing new challenges.

This paper starts with a look at these new challenges for software development and the requirements for debuggers that follow from that. It then gives a short overview of past and current debuggers for rule-based systems and discusses whether they can satisfy these requirements. It goes on to describe Explorative Debugging as a novel concept for the debugging of rule bases and shortly sketches the Inference Explorer - an explorative debugger for F-logic.

3. Challenges And Requirements

Tools for the development of rule-based programs have seen relatively little development in the past years; the most sophisticated debugging systems are still often those that are more than 10 years old. At the same time, however, the computer science landscape has changed, changing the environment a debugger has to work in. This section describes the challenges for a debugger in a modern environment and the requirements that follow from that.

3.1. End User Programmers

The amount of software in society increases continually, and more and more people are involved in its creation. The creation of software artifacts used to be the very specialized profession of a few thousand experts worldwide, but has now become a set of skills possessed to some degree by tens of millions of people. For the US it is estimated that there are at least four times as many end user programmers as professional programmers [19]; with estimates for the number of end user programmers ranging from 11 million [19] up to 55 million [4]. At the same time, however, this development means that the average programmer is not trained as well as the knowledge engineers that created the first expert systems. An end user programmer is usually trained for a non-programming area and just needs a program, script or spreadsheet as a tool for some task. End user programmers usually can't justify making an investment in programming training comparable to that of professional programmers.

End user programmers are particularly important for the Semantic Web[3]. Because for the Semantic Web to succeed a large number of current web developers need to start building Semantic Web applications and it is known that web developers have a particularly high percentage of end user programmers [16, 10]

End user programming is posing many challenges to software engineering, software development tools (see [2] for an overview) and debugging tools (e.g. [17, 18])). Of particular importance are simplicity and interactivity. For a debugger to be simple it has to be usable without (much) training. *Simplicity* also means that the debugger for a declarative system needs to work on the declarative semantics and not on the procedural nature of the inference engine. A debugger that works on the procedural nature of the inference engine forces the user to understand its inner workings as well and thereby becomes difficult to use.

Interactivity means that the debugger needs to be integrated with the rest of the development system to support the trial and error way of knowledge formulation of end user programmers. End users often incrementally build their software system by experimenting with it; during this process, they expect debugging support to be always available [17].

3.2. High Level Editors

High level editors are tools that allow creating programs at an abstraction level above the source code. Usually these editors create the source code as output. The most common forms of high level editors are visual editors, but they can also be based on forms or (constraint) natural language.

Today computer programs are still created mainly by writing source code, however, the role of high level, graphical interfaces is increasing. For instance UML support has found its way into many IDEs and is used particularly in the early stages of a software development process. In Semantic Web tools and in business rule systems, graphical editors are even more common than in today's IDEs for object oriented development. A 2004 survey found that 73 of the investigated 94 ontology tools included or planned graphical tool support [7]. All of the currently availabe commercially Semantic Web Editors (TopBraid Composer [22], Semantic Works [1] and OntoStudio [15]) include full fledged graphical editors. Protégé 2000 [13], probably the most well known open source editor for Semantic Web data, utilizes form like, high level editors instead of source editors. The state of the art business rule system Common Knowledge 3.0 [14] offers seven different editors to formulate business knowledge, including tables, trees and workflow diagrams.

Developers and researchers in the Semantic Web and Business Rule communities hope that this non-source code, higher level editors will make it easier to create programs in their respective formalism. It is assumed that higher level editors free the user from worrying about syntactic goals, allow her to pursue the necessary steps in a more natural order and reduce the need for awkward combinations of primitives [9].

Graphical and non-source code editors are a challenge for the debugging support. Such editors make it more difficult to relate the program execution to the entities the user works with and to display it in a way that is understandable to the user.

In a source code oriented programming environment the source files and line numbers form the relatively simple basis for the instrumentation of the program by the debugger and for communication between debugger and the user. On the one side the debugger can add information about the files and line numbers to the compiled program and use this information to communicate with the runtime environment. At the same time the debugger can then directly utilize this information to communicate the current state of program execution to the user.

In an environment with graphical and high level editors the running program is usually created in a two step process: first a source code representation is created from the visual representation in the editor and then this source code is compiled. Both the source code and the nature of the translation from visual representation to source code are usually hidden from the user. In such an environment the debugger must either support a more complex program instrumentation based on the visual elements of potentially many different editors or translate line based data on the execution of the program into a user understandable representation. At the same time the debugger must ensure that these translations preserve enough information to enable the user to debug all or at least the most common problems.

3.3. Interfaces to other systems

Almost no large computer program today is built from ground up without interfaces to other systems. Systems use databases, connect to web services over the internet or at least access fast subroutines written in other languages. Most of the other systems that a rule base will use will not themselves be rule bases. Any large rule-based application will involve non rule-based elements.

Consider for example the OntoBroker inference engine

[6]. It supports so called *builtins* as mechanisms to access databases, string processing or fast math processing from inside a rule language. These builtins can be used like normal predicates in rules, but are evaluated by calling a Java program. The variable bindings created by these builtins are determined by an object oriented program outside the scope of the inference engine. Another example is again the business rule system Common Knowledge [14] that allows a relatively free mixing of procedural and rule-based program parts.

Modern debuggers for rule-based systems must take into consideration these interfaces to other systems, in particular since it can be expected that the transitions between the paradigms will be particularly problematic. The user must be able to investigate these transitions, learn about the non-rule-based elements of her program and ideally be able to do at least black box tests on that parts.

4. Current Debugging Support For rule-based Systems

All deployed debugging tools for rule-based programs known to the authors are based on the procedural or imperative paradigm. This debugging paradigm is well known from the world of procedural and object oriented programming and characterized by the concepts of breakpoints, step, step-into and step-over[1]. A procedural debugger offers a way to indicate a position in the program where the program execution is to stop (a breakpoint or spypoint) and has to wait for commands from the user. When the program execution stops at such a point, the user can look at the current state of the program and give commands to execute a number of the successive instructions. In rule-based systems the breakpoint is usually attached to a rule, a predicate or a part of a rule. The state of the program execution is characterized by current variable bindings, which parts of the proof succeeded already and which failed. Depending on the exact debugger not all of that state may be visible. More sophisticated debuggers for declarative programs use an explicit model of procedural workings of the inference engine (usually the Byrd box model[5]) as basis for their explanation of the programs working.

Procedural Debuggers for rule-based systems are available as purely textual tracers[24], with a simple graphical user interface[24] or integrated into graphical knowledge acquisition tools[14]. The most sophisticated of these systems was probably the Transparent Prolog Machine[8] that offered carefully crafted graphical representations of the inference process at different abstraction levels. All but the debugger for the business rule system [14] fail to satisfy all of the requirements stated in the previous section. They are not simple to use; force the user to learn about the execution model of the inference engine. These systems also assume a monolithic, unconnected knowledge base and are not integrated with graphical or high level editors. The debugger of the business rule system common knowledge studio [14] is an exception, because it is integrated with graphical editors. It is, however, very limited it its support for debugging of more complex problems where the problem lies in the interaction between rules or parts of the program.

Many other debugging paradigms have been proposed in research, the most notable ones being algorithmic debugging[21][2], methods to automatically find and possibly correct problems in the knowledge base and debugging through a textual dialog between user and system[23]. So far, however, none of these systems has proven to be robost and effective enough for large scale pratical use. No systems implementing these paradigms is easily available to today's developer of rule-based systems.

5. Explorative Debugger

We propose explorative debugging as a better debugging paradigm for rule-based systems. Explorative debugging works on the declarative semantics of the program and lets the user navigate and explore the inference process. It enables the user to use all her knowledge to quickly find the problem and to learn about the program at the same time. An Explorative debugger as we understand it is not only a tool to identify the cause of an identified problem but also a tool to try out a program and learn about its working.

Explorative Debugging puts the focus on rules (or other high level entities the user manipulates[3]). It enables the user to check which inferences a rule enables, how it interacts with other parts of the rule base and what role the different rule parts play. Unlike in procedural debuggers the debugging process is not determined by the procedural nature of the inference engine but by the user who can use the logical/semantic relations between the rules to navigate. A cornerstone of explorative debugging is the explicit showing of logical relations between rules and how the interaction between rules creates the results. These connections are of particular importance because they are usually not visible anywhere else in tools for the development of rule bases.

An explorative debugger is a tool that:

- Shows the inferences a rule enables.

[1]In rule-based systems the analog concepts are also called spypoint, creep, skip, abort and retry

[2]Similar works has also been published under the terms declarative debugging, declarative diagnosis, guided debugging, rational debugging and deductive debugging

[3]For the sake of clarity we will speak only of rules, rule parts and rule bases in the following sections; however, the concepts of explorative debugging could easily be applied to other kinds of declarative knowledge

- Visualizes the logical/semantic connections between rules and how rules work together to arrive at a result. It supports the navigation along these connections.
- It allows to further explore the inference a rule enables by digging down into the rule parts.
- Is integrated into the development environment and can be started quickly to try out a rule as it is formulated.

6. Inference Explorer

We created the prototype Inference Explorer - an F-logic[4] [11] debugger integrated into the ontology development suite OntoStudio [15] - as a first attempt to realize the explorative debugging paradigm.

Rules form the center element for the Inference Explorer. The debugger is always focused on one rule. It displays this rule, whether this rule's body is satisfiable given the current knowledge base and what it can conclude. The debugger is also able to display justifications for the conclusions a rule draws. The debugger supports the user in understanding the current rule, why the current rules allows to infer something or why not.

A screenshot of the debugger is shown in Figure 1. The interface consist of three main parts: the rule details in the top, the result view bottom left and the details view in the bottom right. Both the result view and the details view change depending on what the user has selected.

The rule details view shows a textual description of the currently selected rule (in stylized English). The rule displayed is a 'user level rule' that may in fact be represented by more than one rule in the knowledge base. The user can select parts of the rules to get more information about these and the rule parts are sometimes shown in different colors to highlight unsatisfiable parts or builtins (predicates whose results are computed by a java program outside the inference engine).

The result view to the bottom left of the debugger shows the variable bindings for which the current rule infers something, the values that satisfy the rule body. When the user selects a rule part in the rule details view the result view changes to display the variable bindings that satisfy this rule part.

The details view to the bottom left changes greatly in response to the currently selected element in the debugger.

- When nothing or a rule part is selected, the details view shows the other rules in the knowledge base the current rule (part) may depend on; i.e. rules for which a head atom unifies with one of the (selected) body atoms of the current rule. The user can click on any rule listed there to open it in the debugger.
- When a builtin rule part is selected, the rule details view shows a documentation for it.
- When a result line is selected in the result view, the details views shows the prooftree/derivation tree for this result. The derivation tree is a translated/user level version of the derivation tree returned by the inference engine. This configuration is shown in the screenshot in Figure 1. The user can click on any rule in the prooftree to open the debugger for it.

The Inference Explorer can be used to debug queries as well as rules - after all a query is just a rule without a head.

The Inference Explorer is integrated with a graphical rule editor. The inference explorer can be opened directly from the graphical editor and the user can easily jump from the debugger to the editor. Furthermore when a user clicks on a rule part in the debugger this part is also highlighted in the graphical editor.

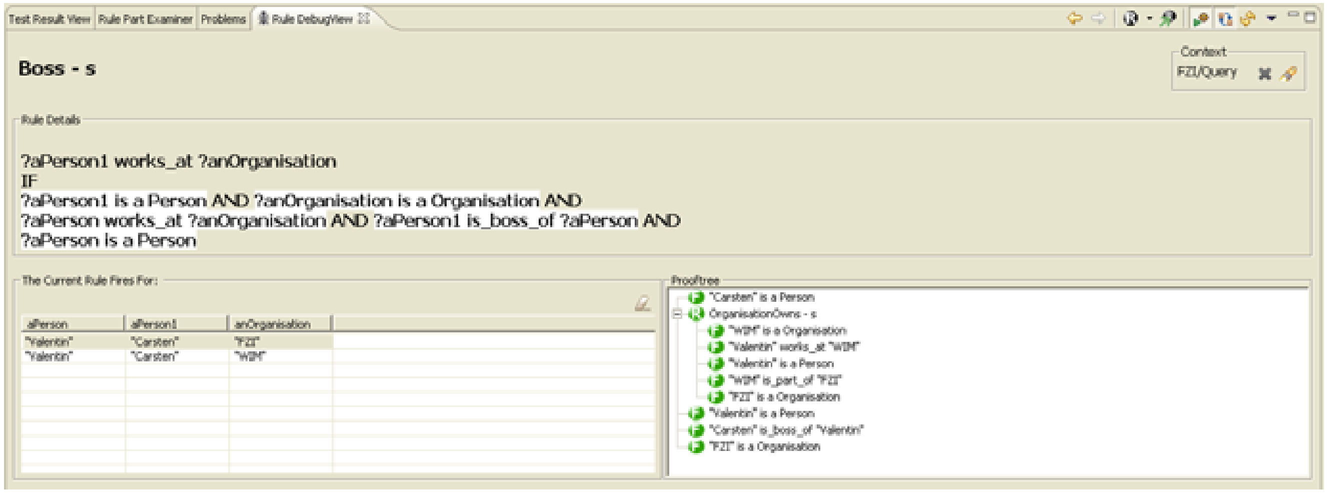
The Debugger also has some support for rule bases that call programs outside of the inference engine. These extra-logical parts are represented in F-logic as builtins - special predicates whose results are not inferred by the inference engine but computed by other programs. Builtins are used to realize for example database access or a query to a web service. The inference explorer detects these builtins in the rules and displays the documentation for each. For the future it is planned to also include a simple way to try out these builtins.

The Inference Explorer shows the connection and interactions between rules in two different ways: as prooftree and as depends on links. The prooftrees for results show how the entities in the knowledge base interacted to compute an actual result. Depends-on links show the static connections between rules that make it likely that two rules could interact - if the knowledge base contained the right facts. The static links are particular important in a case when a query does not return a result because a rule it depends on isn't satisfiable. In such case the user can use the static depends-on links to quickly navigate to the faulty rule. The showing of the connections between rules in these two ways is arguably the most important part of the debugger because the interactions between the rules and the actual structure of the whole program is shown nowhere else.

7. Conclusion

The growing importance of end user programming, the increasing use of graphical/high level editors and intercon-

[4]F-logic or Frame Logic was developed as an object oriented approach to first order logic. It defines syntax and semantic of an object oriented deductive database. Main features of F-logic include object identity, complex objects, inheritance, polymorphic types, query methods and encapsulation.



Boss - s

Context
FZI/Query

Rule Details

```
?aPerson1 works_at ?anOrganisation
IF
?aPerson1 is a Person AND ?anOrganisation is a Organisation AND
?aPerson works_at ?anOrganisation AND ?aPerson1 is_boss_of ?aPerson AND
?aPerson is a Person
```

The Current Rule Fires For:

aPerson	aPerson1	anOrganisation
"Valentin"	"Carsten"	"FZI"
"Valentin"	"Carsten"	"WIM"

Prooftree

```
"Carsten" is a Person
OrganisationOwns - s
    "WIM" is a Organisation
    "Valentin" works_at "WIM"
    "Valentin" is a Person
    "WIM" is_part_of "FZI"
    "FZI" is a Organisation
"Valentin" is a Person
"Carsten" is_boss_of "Valentin"
"FZI" is a Organisation
```

Figure 1. *Screenshot of the Inference Explorer*

nected nature of todays programs are challenges most current debuggers for rule-based systems fail to address. One of the main reasons for this failure is the use of the procedural debugging paradigm that makes these debuggers hard to use.

Explorative debugging is a new paradigm for building debugging tools for rule-based programs. It stipulates that debuggers should work on the declarative semantics of a rule base instead of the procedural nature of the inference engine. Debuggers should show which inferences the current rule enables and how it relates to other rules in the knowledge base. An explorative debugger allows the user to further explore the inferences enabled by a rule by focussing on rule parts. It needs to be integrated with a development environment to allow the user to quickly try out a rule as it is created.

Explorative debugging is applicable not only to F-logic but to most declarative languages. Examples are horn logic programs, normal logic programs, queries and view definitions in relational databases and even description logics such as OWL. The exclusive focus on the declarative properties of rules, however, makes Explorative Debugging less useful for debugging Prolog programs - the Prolog cut statement influences the imperative evaluation of the program and hence makes it hard to debug a program only on the declarative level.

The Inference Explorer presented in this paper implements the explorative debugging paradigm, is also integrated with a graphical editor and offers some support for the debugging of non-rule-based program parts. It adresses the challenges faced by modern debuggers and is by far the most sophisticated debugger available for F-logic. In our day-to-day work it has already proven to speed up the creation of rule bases considerably.

For the future we plan to extend the Inference Explorer to support other rule languages. We also plan to investigate the applicability of the Explorative Debugging paradigm for other declarative languages, in particular OWL, the web ontology language.

References

[1] Altova. Semanticworks. http://www.altova.com/, 2007. (accessed 2007-02-27).

[2] Myers B., Ko A., and Burnett M. Invited research overview: End-user programming. In *ACM Conference on Human-Computer Interaction (CHI'06)*, 2006.

[3] T Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 20:34–43, 2001.

[4] Barry Boehm. *Software Cost Estimation with COCOMO II*. Prentice Hall PTR, 2000.

[5] L. Byrd. Understanding the control flow of prolog programs. In *Proceedings of the Workshop on Logic Programming*, 1980.

[6] S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology-based access to distributed and semi-structured unformation. In *Database Semantics: Semantic Issues in Multimedia Systems*, pages 351–369, 1999.

[7] Michael Denny. Ontology tools survey, revisited. *XML.com*, 2004.

[8] M. Eisenstadt, M. Brayshaw, and J. Paine. *The Transparent Prolog Machine: visualizing logic programs*. Intellect Books, 1991.

[9] R.R.G. Green and M. Petre. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages and Computing*, 1996.

[10] Warren Harrison. The dangers of end-user programming. *IEEE Software*, July/August:5–7, 2004.

[11] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of the ACM*, 42(4):741–843, 1995.

[12] Michael Kifer, Jos de Bruijn, Harold Boley, and Dieter Fensel. A realistic architecture for the semantic web. In *RuleML*, pages 17–29, 2005.

[13] N.F. Noy, M. Sintek, S. Decker, M. Crubezy, R.W. Fergerson, and M.A. Musen. Creating semantic web contents with Protégé-2000. *IEEE Intelligent Systems*, 2(16):60–71, 2001.

[14] ObjectConnections. Common knowledge. http://www.objectconnections.com/, 2007. (accessed 2007-02-13).

[15] Ontoprise. Ontostudio. http://www.ontoprise.de/, 2007. (accessed 2007-02-27).

[16] M. B. Rosson, J. Balling, and H. Nash. Everyday programming: Challenges and opportunities for informal web development. In *Proceedings of the 2004 IEEE Symposium on Visual Languages and Human Centric Computing*, 2004.

[17] J. Ruthruff and M. Burnett. Six challenges in supporting end-user debugging. In *1st Workshop on End-User Software Engineering (WEUSE 2005) at ICSE 05*, 2005.

[18] J. Ruthruff, A. Phalgune, L. Beckwith, and M. Burnett. Rewarding good behavior: End-user debugging and rewards. In *VL/HCC'04: IEEE Symposium on Visual Languages and Human-Centric Computing*, 2004.

[19] C. Scaffidi, M. Shaw, and B. Myers. Estimating the numbers of end users and end user programmers. In *L/HCC'05: Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 207–214, 2005.

[20] Kristen Seer. The 2005 business rules awareness survey. *Business Rule Journal*, 2005.

[21] E. Y. Shapiro. *Algorithmic program debugging*. PhD thesis, Yale University, 1982.

[22] TopQuadrant. Topbraid composer. http://www.topbraidcomposer.com/, 2007. (accessed 2007-02-27).

[23] William van Melle, Edward H. Shortliffe, and Bruce G. Buchanan. Emycin: A knowledge engineer's tool for constructing rule-based expert systems. In *Rule-based expert systems*. Addison-Wesley, 1984.

[24] J. Wielemaker. An overview of the swi-prolog programming environment. Technical report, University of Amsterdam, The Netherlands, 2003.

Truth Eliciting Mechanisms for Trouble Ticket Allocation in Software Maintenance Services

Karthik Subbian
IBM® India Software Laboratory
Embassy Golf Links Business Park
Bangalore 560 071
ksubbian@in.ibm.com

Y. Narahari
Indian Institute of Science
Computer Science and Automation
Bangalore 560012
hari@csa.iisc.ernet.in

Abstract

Ticket allocation or problem allocation is a key problem in the software maintenance process. Tickets are usually allocated by the manager or the technical lead. In allocating a ticket, the manager or technical lead is normally guided by the complexity assessment of the ticket as provided by the maintenance engineers, who are entrusted with the responsibility of fixing the problem. The rationality of the maintenance engineers could induce them to report the complexity in an untruthful way so as to increase their payoffs. This leads to non-optimal ticket allocation. We address the problem of eliciting ticket complexities in a truthful way from the maintenance engineers, using a mechanism design approach. In particular, the problem is modeled as that of designing an incentive compatible mechanism and we offer two possible solutions. The first one, TA-DSIC, is a dominant strategy incentive compatible solution which ensures that truth revelation is optimal for each maintenance engineer irrespective of what the other engineers report. The second solution, TA-BIC, is a Bayesian incentive compatible mechanism which only guarantees that truth revelation is optimal for a maintenance engineer whenever all other engineers are also truthful. We show that the proposed mechanisms outperform conventional allocation protocols in the context of a representative software maintenance organization.

1. Introduction

A trouble ticket (or synonymously a ticket) is a software problem as reported by a customer to be analyzed and fixed by a team of maintenance engineers. The process of allocating the ticket to one of the maintenance engineers is called as a *Ticket Allocation Problem*. The challenge in this problem is to identify the right person to solve the problem

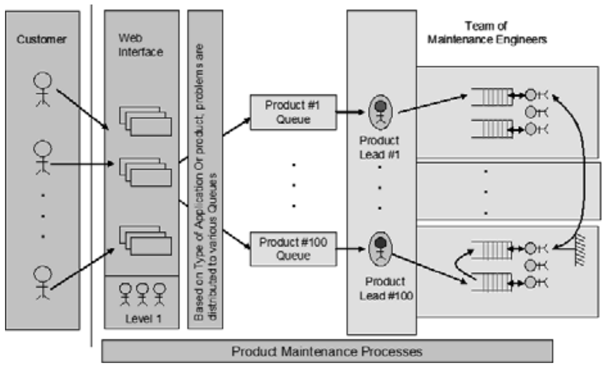

Customer
Web Interface
Level 1
Based on Type of Application Or product; problems are distributed to various Queues
Product #1 Queue
Product #100 Queue
Product Lead #1
Product Lead #100
Team of Maintenance Engineers
Product Maintenance Processes

Figure 1. Typical Software Maintenance Work flow

guaranteeing a specified service level. A typical software maintenance process to assign and resolve trouble tickets is shown in Figure 1. A few analytical approaches have been explored to improve the efficiency of this process. For example, the work by Kulkarni et al. [6] models the maintenance process using queueing theory and determines the optimal number of engineers to be allocated for the task of maintenance during a specific time period. The work by Antoniol et al. [5] models the maintenance organization as a queueing network to assess staffing, evaluate service level, and finds the probability of meeting the maintenance deadlines. Many authors also use statistical techniques [1] and empirical techniques [4] to analyze and improve the software maintenance process.

The thrust of the above relevant papers has been primarily on analyzing the maintenance data for improving the maintenance process. However, all the above papers implicitly make a crucial assumption, namely, that the data is truthfully reported by all the agents and the agents are loyal to the organization. However, the rationality of the maintenance engineers may induce them to report the complexity of a ticket in an untruthful way so as to increase their payoffs. This leads to non-optimal ticket allocation. This paper addresses the problem of truthful elicitation of ticket complexities using a game theoretic and mechanism design approach.

The paper offers the following contributions:

- We model the ticket allocation process in software maintenance organizations as a strategic form game involving rational and intelligent players (namely the lead and maintenance engineers). The problem of ticket allocation is modeled as that of designing an *incentive compatible mechanism*, that is a mechanism which makes truth revelation an optimal strategy for the players. This is the subject of Section 2.

- In Section 3, we propose two possible solutions to the ticket allocation (TA) problem with rational players. The first one is a *dominant strategy incentive compatible* (DSIC) solution based on the classical Vickrey-Clarke-Groves (VCG) mechanisms [7][2]. We call this ticket allocation mechanism as TA-DSIC. This ensures that truth revelation is optimal for each maintenance engineer irrespective of what the other engineers report. The second solution is a *Bayesian incentive compatible* mechanism which only guarantees that truth revelation is optimal for a maintenance engineer whenever all other engineers are also truthful. We call this mechanism TA-BIC. This second solution is based on the dAGVA mechanism [7][2].

- We discuss a stylized case study of a software maintenance process and carry out a variety of experimentation using the proposed mechanisms and also using conventional protocols for ticket allocation. We demonstrate the efficacy of the proposed mechanisms through the experiments. We do this in Section 4.

2. The Model

In this section, we describe the ticket allocation problem and its characteristics, and model it as a mechanism design problem.

2.1. The Problem

Every software maintenance organization as described in Figure 1 has a set of maintenance engineers with a set of queues which each of these engineers own. Whenever a problem arrives at the product queue, the product lead assigns this ticket to one of the maintenance engineers. This ticket assignment is completely based on the experience of the product lead. The problem of allocating the ticket that has arrived into the product queue to one of the engineers is called *Ticket Allocation Problem*.

2.2. Some Characteristics of the Problem

A ticket or a maintenance request can be characterized by three important parameters [3]. They are service time, waiting time, and probability of fix.

1. Service time (st_i): This is the amount of time that engineer i would need to fix the ticket.

2. Waiting Time (wt_i): This is the amount of time the ticket waits in the queue of engineer i.

3. Probability of Fix (f_i): This is the probability with which engineer i can fix the ticket. A typical fix ratio would be (number of tickets fixed in a time period) divided by the (number of tickets assigned to the engineer in that time period).

Each engineer announces a service time, a waiting time, and a probability of fix, which is (st_i, wt_i, f_i), to the lead. Then using the set of such announcements from all engineers the lead assigns the ticket to one of the queues and allocates a payment for fixing the ticket. This is very much like an auction where the bids are $((st_i, wt_i, f_i))_{i\in N}$ and the allocation, payments represent the outcome of the auction. This is illustrated using Figure 2.

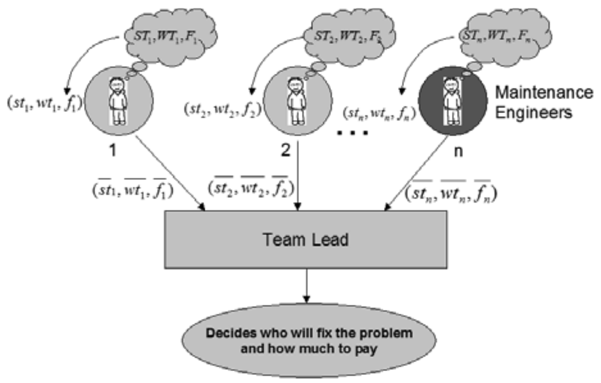

ST_1, WT_1, F_1
ST_2, WT_2, F_2
ST_n, WT_n, F_n
(st_1, wt_1, f_1)
(st_2, wt_2, f_2)
(st_n, wt_n, f_n)
Maintenance Engineers
1
2
n
$(\overline{st_1}, \overline{wt_1}, \overline{f_1})$
$(\overline{st_2}, \overline{wt_2}, \overline{f_2})$
$(\overline{st_n}, \overline{wt_n}, \overline{f_n})$
Team Lead
Decides who will fix the problem and how much to pay

Figure 2. Ticket Allocation Problem

2.3. Ticket Allocation as a Mechanism Design Problem

Mechanism design is the problem faced by a social planner who seeks to make a socially desirable decision based on the collective inputs from all the players participating in the mechanism. In our problem, the social planner is the lead who makes the decision of allocation and payment. The players participating in the mechanism are maintenance engineers. From now, on the word *players* will be used interchangeably with *maintenance engineers*. A more formal description of the problem follows.

1. Players (N): The maintenance engineers or the players are indexed by $i = 1, 2, ..., n$, who will announce their types to the social planner or the product lead.

2. Multidimensional Type Set (L_i): The type of a player is a triple that consists of a service time, waiting time, and a probability of fix. The type set of a player describes the set of all possible values that can be taken by the types of a player.

   $$L_i = \{(st_i, wt_i, f_i) : st_i \in [\underline{st}_i, \overline{st_i}] \subset \Re, \\ wt_i \in [\underline{wt}_i, \overline{wt_i}] \subset \Re, f_i \in [0,1] \subset \Re\}$$

3. Single Dimensional Type Set (Θ_i): We define a function $q_i(.)$ which maps each type (which is a triple) into a single dimensional type (for the sake of convenience). In the sequel of this paper, we will work with these single dimensional type sets.

   $$\Theta_i = \{(\theta_i) : \theta_i = q\,(l_i)\,\forall l_i \in L_i\}, \forall i \in N \\ q : L_i \to \Theta_i \\ q\,(l_i) = q((st_i, wt_i, f_i)) = -\tfrac{st_i * wt_i}{f_i}$$

   The negative sign in this function $q(.)$ denotes the amount of effort spent by the engineer to fix the problem. We can also use weights in order to give higher or lower importance to either of these factors. The function $q'(.)$ below uses weights α, β, γ for this purpose.

   $$q'\,(l_i) = q'((st_i, wt_i, f_i)) \\ = -\left(\frac{(st_i)^{\alpha}(wt_i)^{\beta}}{(f_i)^{\gamma}}\right)^{\frac{1}{(\alpha+\beta-\gamma)}}, \alpha > 0, \beta > 0, \gamma > 0$$

   Throughout this paper, we have used function $q(.)$ for conversion to single dimensional type.

4. Outcome set (X): This denotes the set of all possible outcomes (allocations and payments) that a social planner can choose from,

   $$X = \{(k, t_1, t_2, ..., t_n) : k = (y_1, y_2, ..., y_n), \\ y_i \in \{0,1\}, \sum_{i=1}^{n} y_i = 1, \sum_{i=1}^{n} t_i \geq 0, t_i \geq 0, \forall i \in N\}$$

   where k is the project choice. The vector $k = (y_1, y_2, ..., y_n)$ denotes the allocation rule; if $y_i = 1$ then the ticket is allocated to player i, otherwise it is not allocated to player i. The vector $(t_1, t_2, ..., t_n)$ denotes the amount of payment that player i would receive for fixing the problem. If $t_i > 0$, then player i receives a positive sum of money (to fix the problem), otherwise pays a positive sum of money.

5. Valuation function $v_i(.)$: Valuation of the ticket for a player i is the amount of effort spent by the engineer, when the ticket is allocated to the engineer, otherwise the valuation is zero. Formally we have considered the valuation function to be linear and it is defined as,

   $$v_i(k, \theta_i) = v_i(y_1, ..., y_n, \theta_i) = \theta_i y_i, \forall i \in N, \forall \theta_i \in \Theta_i$$

6. Utility Function $u_i(.)$: Utility function for each player i denotes the amount of utility that each player receives by participating in the game. Typically, utility is payment minus valuation. If the utility is positive for a player, then the payment made for fixing a problem is greater than the valuation of effort made by the player for the ticket. Thus, we define the utility function as,

   $$u_i(x, \theta_i) = v_i(k, \theta_i) + t_i, \forall i \in N, \forall \theta_i \in \Theta_i$$

7. Belief Function $\phi_i(.)$: Every player i knows his own type and has a belief about the type preferences of the other players. This type preference is expressed in the form of a belief function $\phi_i(.)$, which is a probability distribution function over the type set Θ_i of player i. $\Phi_i(.)$ denotes the cumulative distribution function for the probability distribution function $\phi_i(.)$. The belief of each player is independent of the other players, hence we assume the set of distribution functions $(\phi_1, \phi_2, ..., \phi_n)$ are statistically independent. Formally,

   $$\phi : \Theta \to [0,1] \\ \phi = \phi_1(.) \times \phi_2(.) \times ... \times \phi_n(.)$$

   We assume that the distribution $\phi_i(.)$ for every player i follows uniform distribution over all types of that player, and we will use this assumption in our case study.

8. Social Choice Function $f(.)$: The decision made by the social planner is depicted in terms of a social choice function. A social choice function chooses an outcome from the outcome set depending on the set of all individual types of players (engineers in our case). The social choice function is,

   $$f : \Theta \to X$$

   where

   $$\Theta = \underset{i \in N}{\times} \Theta_i$$

3. Incentive Compatible Mechanisms for Ticket Allocation

In this section, we describe two incentive compatible solutions, TA-DSIC and TA-BIC, that elicit truthful information from the players.

3.1. The TA-DSIC Mechanism

The payment rule we use here is the well known *Clarke's payment* [7], which is a special case of the Groves payment scheme [8]. If the social choice function is *allocatively efficient (i.e., allocating the good to the person who values it the most)* and uses Clarke's payment rule then the SCF is DSIC [7].

The following optimization problem determines the allocatively efficient project choice for the ticket allocation problem.

$$\begin{array}{l}\max\limits_{k\in K}\sum\limits_{i=1}^{n}\theta_i y_i\\ s.t,\\ \sum\limits_{i=1}^{n} y_i = 1, y_i \in \{0,1\}\forall i\in N\\ \theta_i = -\frac{st_i * wt_i}{f_i}\forall i \in N\end{array}$$

Following is the Clarke's payment rule that we will use in our social choice function,

$$\begin{array}{l}t_i(\theta) = \sum\limits_{\substack{j\in N\\ j\neq i}} v_j(k^*(\theta),\theta_j) -\\ \sum\limits_{\substack{j\in N\\ j\neq i}} v_j(k^*_{-i}(\theta_{-i}),\theta_j), \forall i \in N, \forall \theta_i \in \Theta_i\end{array}$$

where $k^*(\theta)$ is an allocative efficient project choice when player i plays the game and $k^*_{-i}(\theta_{-i})$ is the allocative efficient project choice when player i does not participate in the game. The formulated social choice function has the following properties other than the DSIC property.

1. Allocative Efficiency: Allocates the ticket to the person who values it the most (with minimum service time, minimum waiting time and maximum probability of fix)

2. Ex post Individual Rationality: Every agent will voluntarily participate as he/she may get at least as equal to what he/she may get by not participating in the game. So, every player finds it in his best interest to voluntarily participate in the game. Formally,

$$u_i(f(\theta),\theta_i) \geq 0, \forall \theta \in \Theta, \forall i \in N$$

Example:
Consider three engineers who are asked to submit bids for a ticket. Table 1 specifies the bids submitted by each player 1,2 and 3. In this example, player 2 is the winner and will be allocated the ticket. The payments for the players are computed as shown below,

Players	st_i	wt_i	f_i	θ_i
1	10	8	0.5	-160
2	5	8	0.5	-80
3	9	7	0.6	-105

Table 1. Example for TA-DSIC

$$\begin{array}{l}t_1 = -80-(-80) = 0; v_1 = 0; u_1 = 0;\\ t_2 = 0-(-105) = 105; v_2 = -100; u_2 = -80+105 = 25;\\ t_3 = -80-(-80) = 0; v_3 = 0; u_3 = 0;\end{array}$$

In this case, player 2 gets a payment of 25 for fixing the ticket, and this is paid as an incentive to induce truth revelation. All other agents get zero utility by participating in the game which is not worse than all of them not participating in the game. This property corresponds to *ex post individual rationality*.

The disadvantage with this mechanism is that the cost spent by the lead (105 in the example) to induce truth revelation is quite high. This is because the DSIC property is very strong (truth revelation is a best response for each player irrespective of what the other players report) and the incentives to be given to satisfy this property are consequently bound to be quite high. This motivates us to explore the next mechanism TA-BIC, where the SCF is constrained by a relaxed form of incentive compatibility, which is Bayesian Incentive Compatibility. The TA-BIC mechanism is based on the expected externality mechanism or the dAGVA mechanism [7].

3.2. The TA-BIC Mechanism

The dAGVA theorem [7]. confirms that in quasi-linear environments, there exist social choice functions which are both *Ex post Efficient* (that Allocatively Efficient as well as Budget Balanced) and truthfully implementable in Bayesian Nash equilibrium (*Bayesian incentive compatible*). The payment rule in this case is,

$$t_i(\theta) = E_{\theta_{-i}}\left[\sum_{j\neq i} v_j(k^*(\theta_i,\theta_{-i}),\theta_j)\right] + h_i(\theta_{-i}) \ \forall\, i$$

We can always choose $h_i(\theta_{-i})$ such that the SCF is strictly budget balanced [7]. The strict budget balanced property ensures that there is no need for any injection of money from external sources into the system. So, based on the truthful revelations of individuals participating in the

Players	st_i	wt_i	f_i	θ_i
1	10	8	0.5	-160
2	5	8,12	0.5	-80,-120
3	9	7	0.6	-105

Table 2. Example for TA-BIC

game, some of them pay and some of them receive, the sum of payments and receipts is perfectly balanced, that is,

$$\sum_{\mathrm{i}\in\mathrm{N}} \mathrm{t_i} = 0$$

The $h_i(\theta_{-i})$ in our case is,

$$h_i(\theta_{-i}) \quad = \quad -\left(\frac{1}{n-1}\right)\sum_{j\neq i}\xi_j(\theta_j) \;\; \forall\, i = 1,\ldots,n$$

where,

$$\xi_i(\theta_i) \quad = \quad E_{\theta_{-i}}\left[\sum_{j\neq i} v_j(k^*(\theta_i,\theta_{-i}),\theta_j)\right] \;\; \forall\, i = 1,\ldots,n$$

where,

$$\Theta_{-i} = \{\theta_{-i} : \theta_{-i} \in \underset{\substack{j\in N\\ j\neq i}}{\times} \Theta_j\}$$

Example: We will extend the same example provided for this purpose in Section 3.1, by adding one more type value for some player. Say, we add a type value of 12 for waiting time for player 2. We will assume uniform distribution for $\phi_i(.), \forall i \in N$. The data for the example is provided in Table 2,

As we have not changed the allocation rule, the winner is still player 2 for the auction. But the amount of payment to each player varies from the previous mechanism as the payment rule is changed in this case. Here we list the payment computations for all players.

$\xi_1(\theta_1) = -92.5; \xi_2(\theta_2) = 0; \xi_3(\theta_3) = -40;$
$h_1(\theta_{-1}) = +20; h_2(\theta_{-2}) = +66.25; h_3(\theta_{-3}) = +46.25;$
$t_1 = -92.5 + 20 = -72.5; t_2 = 0 + 66.25 = 66.25$
$t_3 = -40 + 46.25 = 6.25$

The total payment made in this case is 72.5 and player 2 receives 66.25. Interestingly, player 3 receives money 6.25 for just participating in the game and promoting the competition among the players. Due to the BIC property of the SCF the total payment has come down from 105 from the previous mechanism (TA-DSIC) to 72.5.

The disadvantage with this mechanism is that, the mechanism is not individually rational. In this example, we find player 2 gets utility of (-80+66.25=) -13.75 and player 1 gets utility of (0-72.5=) -72.5 for revealing types truthfully. It is not necessary for the maintenance organization to have individual rationality property in their SCF, as they can force all players to participate in the game. Yet, if the nature of inflow of tickets is such that it favors only a subset of engineers in the team (consistently), then it is highly likely that the motivation levels of the other set of engineers would be lost over time.

4. A Case Study

In our case study, we consider a maintenance organization with three maintenance engineers, a lead and a customer. Once the ticket is created by a customer, the lead requests bids (in the form of service time, waiting time and fix ratio) from these three engineers for the reported problem. The lead then decides the engineer and payment for the engineer for fixing the problem using either TA-DSIC or TA-BIC mechanism. For the TA-BIC mechanism the engineers report the type set and their belief functions. We assume the reported belief functions to follow uniform distribution for this case study purpose.

We model the real world allocation using a simple scheme called *regular allocation*. These allocations are primarily based on the experience of the lead where the new ticket is allocated to one of the engineers based on some (random) rational decisions as chosen by the lead. Also, we assume no players report truth consistently in regular allocation.

We try to address the following two questions in our analysis. (1) Does the customer prefer truth revealing mechanisms? (2) Do all engineers prefer truth revelation in such mechanisms?. We address these two questions in Scenario 1 and 2 respectively.

Scenario 1 - Total Payment Analysis: In this scenario we analyze the total sum of money paid to all engineers for solving 100 tickets using $regular$, TA-DSIC and TA-BIC mechanisms. In Figure 3, the (y-axis) shows the cumulative sum of money paid to all engineers for the total number of tickets resolved at any instant of time in the organization. It is evident from Figure 3 that the amount of money paid by customer for TA-DSIC and TA-BIC mechanisms is less than the *regular allocation* scheme. Also, due to the *Bayesian Incentive Compatibility* nature of the dAGVA mechanism, the payments in TA-BIC are far less than that in TA-DSIC. As the two mechanisms TA-DSIC and TA-BIC outperforms *regular allocation* in terms of the total payment, customer prefers these incentive compatible mechanisms.

Scenario 2: - Truth Revelation Analysis: Every player

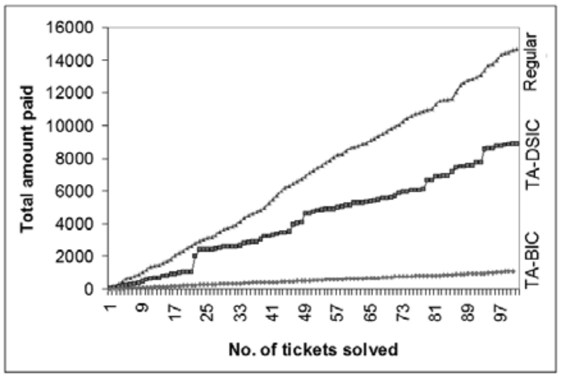


Figure 3. Total Payment Analysis

prefers truth revelation, if there is no effect of lie of other player on their own individual utility. Hence, in this scenario we analyze the impact of utility for all players in the TA-DSIC mechanism when one player (say player 1) does not reveal true valuations. Figure 4 shows the cumulative utility for each player for the total number of tickets resolved at any instant of time in the organization. There are two utility curves for each player i. The first curve *ui* denotes the utility of player i when player 1 reveals truth and the other curve *liar_ui* denotes the utility of player i when player 1 does not reveal truth. We find that the utility of player 1 (*liar_u1*) when he acts as a liar is less than when he reveals truth (*u1*). Also, it is in the best interest of other players (2 and 3) to continue revealing truth, when player 1 acts as a liar, as it does not decrease their utility. So, apparently every engineer prefers to reveal truth in this mechanism. We can similarly show that in the TA-BIC

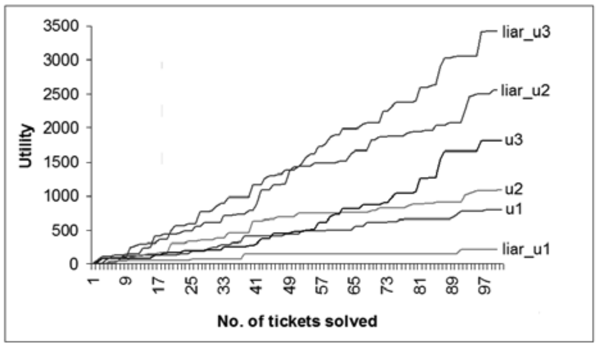


Figure 4. Truth Revelation Analysis

mechanism when all other players reveal truth, there is no (incentive) increase in utility for any individual player i to lie.

5. Conclusion and Future Work

In this paper we have shown that truth eliciting (or incentive compatible) mechanisms outperform *regular allocation* schemes and are more suited for software maintenance organization with rational and intelligent maintenance engineers. We see this work can be extended further in the following directions:

1. Construct a mechanism which is optimal in a sense that it does a fair trade-off between the TA-DSIC and TA-BIC mechanisms. This mechanism will be individually Rational and Bayesian incentive compatible. So, it brings in all the good properties of the TA-DSIC and TA-BIC mechanisms.

2. Expand the horizon to cooperative scenarios, where we can allow the players to collaborate and solve a problem. The payments and allocations in such setting would be very different from the non-cooperative setting defined here.

3. Build a sophisticated tool for auctioning tickets in software maintenance using defined allocation and payment rules.

References

[1] L. Arfa, A. Mili, M. Alaya, H. Amor, and K. Ketata. Software maintenance management in Tunisia - a statistical study. *Proceedings of the IEEE International Conference on Software Maintenance*, pages 192–195, 1990.

[2] D.Garg and Y.Narahari. Foundations of mechanism design. *Technical Report, Department of Computer Science and Automation, IISc, Bangalore, India*, 2006.

[3] E.B.Swanson. The dimensions of maintenance. *Proceedings of the 2nd International Conference on Software engineering*, pages 492–497, 1960.

[4] E.Burch and H.J.Kungs. Modeling software maintenance requests: A case study. *Proceedings of the IEEE International Conference on Software Maintenance*, pages 40–47, 1997.

[5] G.Antoniol, G.Casazza, G. Lucca, M. Penta, and F.Rago. A queue theory-based approach to staff software maintenance centers. *Proceedings of the IEEE International Conference on Software Maintenance*, pages 510–519, 2001.

[6] V. Kulkarni and S. Sethi. Optimal allocation of effort to software maintenance: A queuing theory approach. *Working Paper, The School of Management, University of Texas, TX, USA.*, 2005.

[7] A. Mas-Colell, M. Whinston, and J. Green. *Microeconomic Theory*. Oxford University Press, New York, 1995.

[8] R. B. Myerson. *Game Theory: Analysis of Conflict*. Harvard University Press, Cambridge, Massachusetts, 1997.

GRAPHICAL NOTATION FOR NATURAL LANGUAGE AND KNOWLEDGE REPRESENTATION

Magda G. Ilieva
Dept. of Computer Science and Software Engineering
Concordia University, Montreal, Canada
magda@cse.concordia.ca

ABSTRACT
This article describes graphical language for the representation of textual user requirements. We need such a language in order to approach the automatic construction of graphical software engineering (SE) models (in particular UML diagrams) as a translation from one graphical language into another. In order to fulfill this goal, we have included universal notation in graphical language, which is equally valid for the presentation of linguistic knowledge and the problem domain knowledge. Using unified graphical presentation of natural language and knowledge, we are literally able to draw text as a picture. This process is based on a deep analysis and a clear representation of the text, using a limited set of meaningful graphical symbols. As a result, we achive a methodology for mining a huge volume of knowledge about relations and structures between entities: words in the text and objects/concepts in the problem domain. The extracted relations generalize knowledge and can be formulated in a presentation appropriate for different purposes. In our case - many target SE models.

KEY WORDS
Modelling, Knowledge systems, NLP, UML, Semantic Network

1. INTRODUCTION
The graphical language (GL) notation came up in our attempt to automatically translate textual user requirements into Unified Modeling Language (UML) diagram [14]. In order to resolve this issue, we face two tasks: the first is to understand the problem in the description and the second is to solve it using established or novel methods. The first task – "understanding the problem" – can be divided in two subtasks: understanding the language and understanding the knowledge expressed (presented) through it. If we juxtapose presentation and understanding we will obtain two types of presentation – presentation of the language, and presentation of the knowledge it carries. Although the linguistic knowledge serves as a basis for obtaining knowledge related to the problem, very often these two types of knowledge are presented differently and distinctly: linguistic knowledge with the corresponding presentation, and problem knowledge with the corresponding presentation. The gap between these two types of presentations is overcome by restricting Natural Language (NL) in such a manner that a limited volme of specific knowledge is arranged in a predefined structure. That is time consuming, leads to loss of information and limits the applicability of the SE model. The question arises: would it not be easier to consider the language, and the knowledge it carries, through a single graphic presentation? Then, the automatic construction of UML diagrams would change from translation of one graphic language into another. It is precisely this idea that is developed in the present article. To begin with, section 2 reviews related works. In section 3 a particular graphic notion is described, which presents the language and the knowledge it carries simultaneously. Text rewritten through GL notion is Semantic Network (SN), and in section 4 there is an example of that. In section 5 we compare our semantic network with the semantic networks summarized in [12], and we indicate future tasks as well.

2. RELATED WORK
Fig.1 presents the scheme which summarizes "understanding and representation" of textual knowledge. It consists of three different parts based on knowledge and processing procedures. First, we are concerned with Domain Knowledge (DK) in the text, but its automatic extraction cannot be effected without considering Linguistic Knowledge (LK). At this stage some systems offer processing for simplifying text and reducing ambiguity. Second, the central part gathers processings which make a model of NL and the knowledge carried in it. These are: syntax (Sy), logical form (LF) and Intermediate Model (IM). Third part of the scheme includes processings that transform the language model and the knowledge (general and domain) it contains into an application specific model. According to this scheme we can categorize the published systems into two types:
The first type of systems follows the upper road of the schema in Fig.1. They prefer to analyse controlled NL because domain knowledge inside a form can be extracted easily and reliably. At each step of the processing they consider domain knowledge and obtain specific final models for specific domain knowledge.
In [5] for example, the controled text consists of knowledge about the types of data, operations over them and relations between them. For the presentation of this knowledge an intermediary graphic form (IM) is chosen which looks like tree data structure, with three types of nodes: data, functionality and context. This graphical model is translated into an end OO model, presented within VDM (Vienna Development Method) notation.

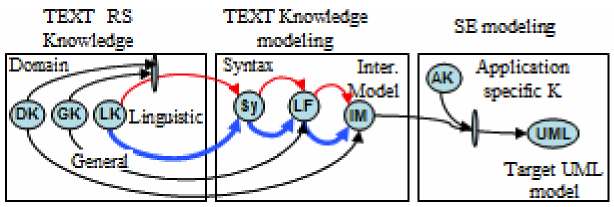


Fig1. Phases of automatic NL Requirements analyzing

In [8] the UCDA (Use Case driven Development Assistant) is presented, which uses as knowledge intermediate presentation (IM) – graph with four types of behaviour: request, validation, change, and response. This type of intermediary presentation fits well into the end graphic UML OO diagram.

In ProCasor project [6], the IM consists of three types of activity (emitted, absorbed and internal) and three types of connections (sequencing, alternative, repetition). The final presentation is a type of activity diagram.

In [7] linguistic patterns are extracted, which are subsequently presented into IM - conceptual patterns for an object model. The IM consists of nine graphic conceptual patterns from which is built a diagram of classes and the relations between them.

Among ***the second type of systems*** are those that attempt to make models more universal and independent from DK and the text. These systems follow the lower road of the scheme. They process unrestricted NL from which are obtained two consequative models: the first model obtained is LF, which adheres more closely to the text and the general knowledge (GK); the second model is IM which adheres more closely to SE model and DK.

The KCPM (Klagensfurt Conceptual Predesign Model) described in [3,4], in the part for the LF presentation contains 12 distinct verb classes, from which are extracted 5 phrase categories. In the part for the IM presentation, these 5 phrase categories assign 3 types of primitives: condition, operation, and cooperation. From the graphic elements which presents these primitives a cooperation type scheme is built, which later is mapped into the UML activity diagram. KCPM is a basic model of the system NIBA [2]. However, in order to make an OO class diagram, NIBA transfers LF into another IM, namely Semantic Network. This graphic consists of things types and connection types. The translation from SN into OO class diagram is not entirely automatic – the designer chooses potential candidates for the conceptual scheme.

Colour-X [1] builds LF as conceptual prototype language which replaces "three central representations, i.e. (structured) natural language sentences, lexicon knowledge and conceptual models". This system doesn't aim to obtain UML or a similar final model. The philosophy of the Colour-X project is to provide the user with a static view and a dynamic view of general knowledge.

Our approach differs from the two approaches described above. We resemble the systems from the second type, obtaining more general and independent information from text and DK presentation. We differ, however, by creating a common model of linquistic and domain knowledge in order to achieve one general graphical IM which aims to reach different target UML diagrams. Colour-X remains on stage LF; it has a unique presentation of different knowledge extracted from the text, which needs to be additionally processed to obtain target SE model. KCPM and NIBA use different IM for different target models. Our approach aims to be more universal. It analyses uncontrolled NL by building a common presentation of LF and IM, which we use for building various final models. The following section describes the notation of our graphic representation.

3. GRAPHICAL LANGUAGE NOTATION

Our graphical language is based on tabular presentation of sentences, built after syntax analysis. We divide the sentence into three basic groups (columns in the table) according to the function each one performs: **Su**(bject), **Pr**(edicate) and **Ob**(ject). The subject and the object are noun groups; they can be simple (nouns and attributes) or complex (consisting of simple nouns connected with operators – prepositional, conjunctional). The predicate is a verb group consisting of a main verb, and its corresponding adverb, modality, infinitive, and auxiliary verbs. These three basic types form a sentence. Several sentences can be connected with conjunctions or relative pronouns in order to produce a complex/compound sentence. It is vital that each main and subordinate sentence contains this triplet (Su, Pr, Ob). Some of the positions in it can be left unfilled for various reasons: i) syntactic peculiarity (inverse phrase, relative phrase following the predicate), ii) understanding from the context or iii) use of the passive verb form. To fill the empty positions we use heuristics (see section 4/Heuristics) or the help of an analyst in interactive mode.

Concepts: The basic building blocks of our graphic language are concepts (entities) and the relations between them. The concepts are nouns/names in NL taking on the role of subject or object. They are characterised by name, gender, singular/plural, definite/indefinite article. In the graph we present only the name of the concept, while the other information is kept in the tabular presentation of the text, which serves us as a knowledge base (KB) for all kinds of syntax and semantics knowledge extracted during the analysis. The concepts are presented in the manner of an oval form containing the name (see Tab.1 line 1), while the relations between the concepts are presented as a pointed and labelled arc. The following paragraphs describe different relations in which concepts are involved.

Predicative: This relation connects two concepts with a verb. It is presented by an arc labelled with the verb, which points from the subject towards the object (Tab.1 line 6). When the action is transferred from the object (Ob) towards the indirect object (iOb) the label of the connecting arrow also contains the preposition which transfers the action. For example: *RTPS charge driver* <u>*at*</u> *a gate* (Tab.1 line 8). We use the active voice of verbs (already changed and saved into KB). Attributes of verbs are adverbs, modality, and time. They can be added in brackets to the label of the arc. This information is contained in the tabular KB, but is not used for all applications.

Prepositional: This relation is between two concepts connected with a preposition. Unlike the predicate arc, the prepositional arc is a dashed line (Tab.1 line 7).

Attributive: This type of relation presents two kinds of grammatical constructions: "adjective(s) followed by noun" or "noun <u>*is*</u> adjective/s". For example: *nice blue car* or *the car* <u>*is*</u> *blue*. More adjectives can be connected through commas or conjunctions. This information we define during the phase of syntax analysis. The graphic presentation of the attributive relation is presented in the form of a solid line oval for the concept (noun) and an oval with a dashed line for the attributes (adjectives) attached to the noun (Tab.1 line 9). One concept can enter into predicative relations alone or with some of its attributes. In the first case the arc of the relation connects to the concept and in the second case to the attribute.

Compositional: This relation type presents the following syntax constructions:

Compound noun (noun-noun modifiers), for example: "*account statement*" or "*toll gate sensor*". The other reading of a compound noun is: *the account has a statement*; *the statement belongs to an account*; *statement of account.* We present the compound noun as an oval for each concept participating in the combination, and we place the ovals adjacent to one another (Tab.1 Line 9). With experience, however, we have developed heuristics to keep the graphic simple and clear. Thus, if a part of the compound noun does not participate on its own in relations, we prefer to express all parts within a single oval, for example, *special line* (Tab1 line 9). If different parts of the compound noun participate in different relations, the arc of the relation is attached to the last concept of the combination, for example: "*toll gate*" – attached to gate or *"toll gate sensor"* – to sensor (see SN in fig2). In that way we keep the order of inheritance.

Noun - verb_be - noun, for example: *the man is the owner* or *Smith is a professor*. We express the noun in the role of an object as a concept (oval form), attached to the main concept (subject) (Tab1, line9a).

Key-word structure: There is one characteristic category key word in the texts, which present structures from data or objects. Such as: *type, kind, consists, include, part, have*…etc. The concept related to the key word forms the head of the structure, while the concept/s (after verb *be* or enumeration) - its body. In the example: *There are three types of toll gates: single toll, entry and exit toll,* the head is *toll gate,* and the *body* - single toll, entry and exit toll. Graphically the head of this structure is depicted as a concept, and the body in two ways: i) rectangular callouts, with label the key word and content the concepts from the body of the structure. This representation is used when the concepts from the body don't participate in other relations in the SN (Tab.1 line 11); ii) fork in the tips of which are placed the concepts from the body of the structure. This representation is used when the concepts from the body participate in other relations in the SN (Tab.1 line 10).

Compression info: In order to achieve precision and eliminate ambiguity, often in written language are used clarifications and redefinitions. This is reached through the use of punctuation, explicatory relative sentences, apposition, etc. We extract this information during the syntax analysis and through heuristics we decide whether or not to include it in the semantic network, and if so in what form. The compressed form is depicted through a numbered pin attached to the concept, which it is related to, and under this number in a legend we write the "compressed" text (Tab. 1 line 17).

Synonyms: For example: *device (gizmo)*. We recognize by the punctuation that a synonym is introduced. We write the synonym after the name of the main concept in the oval form, and divide the two names with a perpendicular line (Tab.1 line 3).

Relations between relations: Graphically these relations are defined with an arc which is between the concept and arc or between two arcs.

IF – THEN – ELSE. In technical texts are often seen conditional sentences which have the following syntax structure: *if* Re1 *then* Re2 *else* Re3. In place of IF one can use: *when, whenever,* etc; In place of THEN: comma, etc. Re1, Re2 and Re3 can be any of the relations described above, concept or attribute, as the presence of Re1 is a condition for the presence of Re2, while the failure of Re1 leads to Re3. This dependence is expressed through two arcs: one connects Re1 with Re2 as in IF-THEN, the other – Re1 with Re3 as in IF-ELSE (Tab.1 line 15). In order to distinguish the two conditional connections, the beginning of one of them (truth) is shaded diamond, and the other (false) is empty. One of the relations can be a result of various conditions, coming from various sentences in various parts of the texts.

Result of the predicative relation. Let's consider the following two sentences, which are consequential in a given text: *The sensor reads the gizmo. The information read is stored by the system*. We notice that the verb from the first sentence becomes an attribute of concept in the next sentence, which most probably means that the concept in the second sentence is a result of the action of the first. This relation is graphically expressed with a pointed arc from the predicate of the first sentence to the resulting concept, as shown in Tab.1 line 12.

Predicative relations connected with conjunctions or relative pronouns. Often one and the same subject fulfils two actions in a single sentence. We can present this as we unite the two predicative arcs with a graphic sign for conjunction/disjunction attached to the subject (Tab.1 line 13).

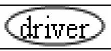
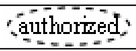
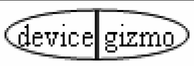
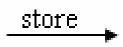
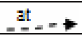
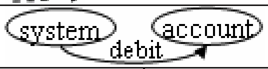
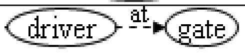
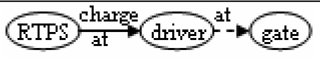
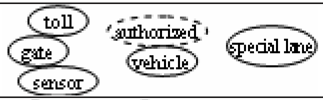
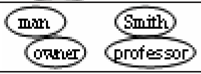
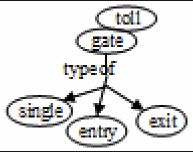
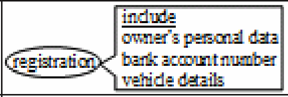
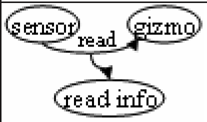
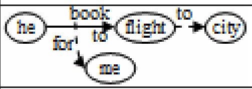
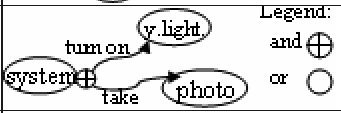
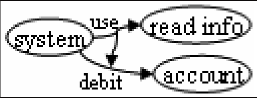
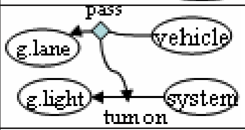
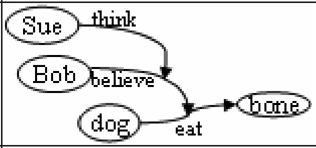
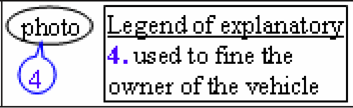
Let's also consider the example: *Sue thinks that Bob believes that a dog is eating a bone*. The graphic of this sentence is as in Tab.1 line 16.

Other examples demonstrating predicative relations: *The client activates a gizmo using an ATM; the system used read info to debit account* (Tab.1 line 14).

4. CASE STUDY

In order to demonstrate the application of our graphic language we propose a solution for an example taken from [18]. In the source, that example is chosen to show extraction of statistical information helping the work of the analyst, and not an automatic analysis of NL. The example is interesting in that the text is uncontrolled and contains various language constructions. The text of the case study follows; the semantic network with described notation of Table 1 is presented in Fig.2.

Table 1. Basic graphical notations with examples

№	Notation	Meaning	Examples
1		Concept with name *driver*	Nouns, proper nouns
2		Attribute with name *authorized*	Adjectives
3		*Gizmo* is synonym of *device*	Device(gizmo); device named gizmo; device called gizmo
4		Predicate with name *store*	All verbs
5		Prepositional connection *at*	All prepositions
6		Predicative relation *debit* between *system* and *account*	**Su**bject **V**erb **Ob**ject – sentence "*system debit account*"
7		Prepositional relation *at* between *driver* and *gate*	Noun preposition Noun "*driver at gate*"
8		The action *charge* is transited from *driver* to *gate*	Su Ve Ob iOb – sentence *RTPS charges driver at gate*
9	 toll gate sensor authorized vehicle special lane	Compositional relations: i)compound noun containing 3 nouns ii)attribute + noun (grouped and ungrouped)	i) *Tool gate sensor* ii) *Authorized vehicle* *Spacial lane*
9a		Compositional relations: Noun- verb *be* –noun	*The man is an owner* *Smith is a professor*
10		Key word structure - fork representation *kind of, type of, has a, consist, include, sort of...*	*There are three <u>types of</u> tool gates: single, entry and exit.*
11	 registration include owner's personal data bank account number vehicle details	Key word structure - callout representation	*Registration includes: owner's personal data, bank account number, vehicle details*
12		One relation gives a result	*i) The sensor reads a gizmo. Read info...* *ii) Client activates a gizmo. Gizmo activation...*
12a		Prepositional phrase attached to the predicate	(*He booked a flight to the city) for me*
13	 Legend: and or y.light turn on system take photo	Conjunction/disjunction of 2 relations with the same Su	*System turns on yellow light <u>and</u> takes a photo.*
14		One relation causes another	*System used read info to debit account.*
15		Conditional relation: **If** rel1 **then** rel2	*If vehicle passes green lane, system turn on green light.*
16		Relative sentence connections	*Sue thinks that Bob believes that a dog is eating a bone.*
17	 photo 4 Legend of explanatory 4. used to fine the owner of the vehicle	Compression info	*photo, <u>used to find the owner of the vehicle</u>*

In a road traffic pricing system, drivers of authorized vehicles are charged at toll gates automatically. The gates are placed at special lanes called green lanes. A driver has to install a device (a gizmo) in his/her vehicle. The registration of authorised vehicles includes the owner's personal data, bank account number and vehicle details.The gizmo is sent to the client to be activated using an ATM that informs the system upon gizmo activation. A gizmo is read by the toll gate sensors.
The information read is stored by the system and used to debit the respective account. When an authorised vehicle passes through a green lane, a green light is turned on, and the amount being debited is displayed. If an unauthorised vehicle passes through it, a yellow light is turned on and a camera takes a photo of the plate (used to fine the owner of the vehicle). There are three types of toll gates: single toll, where the same type of vehicles pay a fixed amount, entry toll to enter a motorway and exit toll to leave it. The amount paid on motorways depends on the type of the vehicle and the distance traveled.
Syntax analysis of the text precedes the building of a semantic network. We use the following processing stages of the text: Part-of-Speech (POS) tagger [15] to obtain syntax category of words; morphological analysis in order to identify inflextion of the words; parser [16] and syntactic chunking [17] in order to identify the three main groups in the sentence – Su, Pr, Ob; general knowledge glossaries for key words; heuristics.
Algorithm for defining the triplet Su, Pr, Ob: The sentences with the verb in a passive form are processed by transfering the verb into active form and changing the positions of the object and the subject. For example: *The gates are placed at special lanes; A gizmo is sent to the client.* After turning the passive form into active this sentences becomes: *Someone places the gates at special lane; Someone sends the gizmo to the client.* Since the examples lack a subject we can keep the position of the subject empty and the analyst will fill it in interactive mode. However, we use heuristics and propose a solution. The heuristics are based on: i) knowledge for agent (*system* is an agent by default); ii) "head" of the phrase (prepositional, passive or active); iii) patterns; iv) the principle of the closest neighbour.
There are three main algorithm steps:
Step 1: Filling in the empty Su positions in passive sentences. We explored many examples of complex sentences with sub sentences in passive voice of the verb. We summarized the cases of using verbs in passive voice into the following three categories: i) conjunction '*and'* between two 'passive' sub sentences introduces repetitiveness which leads us to assume that the subject from the first sentence is repeated in the second; ii) *be+passive* in the second sub sentence takes the subject from the previous sentence; iii) pure passive verb is attached to the nearest neighbour (either subject or object from the previous sentence) that it supports. In this case the neighbour is accepted as subject.
Step 2: We transfer the passive sentences into active. The concept from the subject position becomes direct object, Su position remains empty, indirect object preserves its position, the verb transforms from passive into active.
Step 3: We fill in all Su positions that are left empty. The possible candidates to fill in the empty Su position are: i) an agent, if any is introduced in the previous sentence; ii) *system* by default. We support the glossary with possible agents. Application example of the algorithm follows.

1	The gizmo	is sent	to the client	
2	*The gizmo*	to be activated		
3		using	an ATM	that
4		informs	the system upon gizmo activation	.

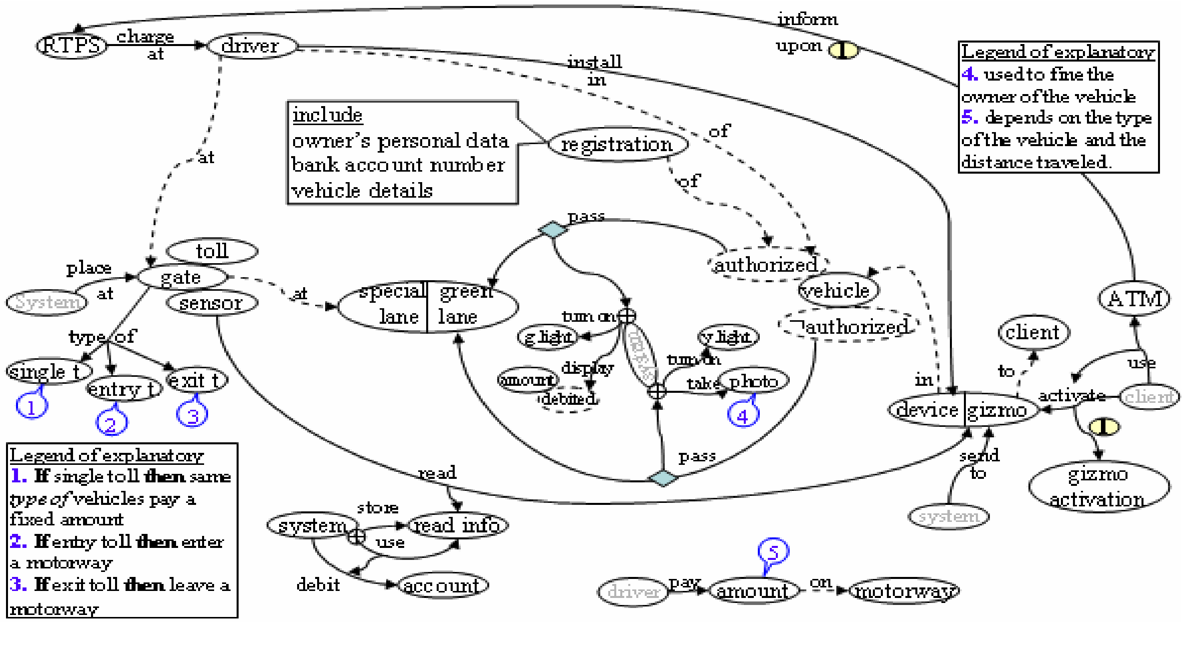


Fig. 2. Semantic Network of textual requirements specification

Step1 ii) defines the subject position of the second sentence, *the gizmo*. Step 2 transforms the passive voice into the active voice and we obtain:

1		sent	the gizmo to the client	
2		activate	gizmo	
3		using	an ATM	that
4		informs	the system upon gizmo activation	.

Step3: An agent is introduced in the first sub sentence – *client,* which becomes the subject of the second and third sentences. *'ATM'* becomes Su of the fourth sentence since it is introduced by a relative pronoun. The only position left empty is the Su position in the first sentence which by default becomes '*system'*. The processed sentences become: ***1****System sends gizmo to the client;* ***2****Client activates gizmo;* ***3****Client uses ATM;* ***4****ATM informs the system upon gizmo activation.*

In tabular view similar to the example illustrated above we inscribe the entire text, from which we later build a SN. Constructing table and its advantages are described in [9].

Heuristics for knowledge compression: In order to preserve a clear picture of SN we prefer to compress some parts of the sentences. Good candidates for compression are explanations to concepts: 1) Business rules. In order to find them we seek i) key words, e.g. *possible, request, except, depends, only*…ii) modality of the verb: *must identify, can be performed…*iii) quantifiers: *all, every, nobody…* 2) Appositions and inversion 3) explanation attached to the enumeration by brackets or relative pronoun (see the second to last sentence in the case study).

We continue to develop and improve the graphic language for presentation of textual user requirements. We seek the most appropriate and readable graph, as well as heuristics for extraction and presentation of all relations.

POS and referential ambiguity: Table presentation of text gives good visualization of text for easy verification of these two problems. The position of the concept/word in the table must correspond to its syntax category. For example, a common mistake is considering verb as a noun and vice-versa. With a quick proofread of the predicate column this mistake can be identified. Similar is the resolution for substitution of reference with real concept – it is found in close proximity in the Su or Ob column.

5. CONCLUSION

Summary: The goal of our project "graphical language for presentation of textual user requirements" is to create a universal graphic model of NL text and the knowledge it carries. We consider this model to be a basis for obtaining graphical models for requirements specification, for example UML models. Instead of developing a unique graphic presentation of the knowledge appropriate for each different UML model we follow this approach: NL is universal as a method for presentation. If we have a graphic model of the language, we can extract necessary knowledge from it and proceed to other graphic models through automatic translations. We present this idea through the following scheme: NL➔ GL ➔ UML models.

The transitions between the different phases of the transformation of NL into UML diagrams can contain different treatments according to the chosen technologies for implementation, but in our system the treatments follow the order: NL ➔ POS tagging/parsing/chunking ➔ Table presentation (visualisation, verification) ➔ XML presentation ➔ Visualisation ➔ Semantic Network ➔ Translation into other graphical models.

Until now we have proposed translation of GL into the following UML models: OO model in [9], and domain model, activity diagram, use case path in [10]. The translation is based on finding the correlation between the graphic templates from source GL and target models.

Comparison with the other Semantic Networks: Our graphic language is a type of semantic network [11, 12].

It has graphical power to present all examples given in [12] which demonstrates the possibilities of a different SN. The philosophy of our graphical model is that there is a relation between the basic building blocks of language and knowledge. This relation consists of 3 elements; it has direction and can influence or is influenced by other relations. For example, Definitional Network [12] is a hierarchy of concept that is easily presented in GL notation through compositional relation.

Implications are the basic elements in the Existential Graph of Peirce [13] and Implicational Networks [12]. Implications are presented easily in GL notation as shown in line 15, Tab.1. Fig. 3 illustrates three types of SN for the example: *If a farmer owns a donkey, then he beats it.* Our GL graph is in the central position.

However, our graphic language is an Assertional Network [12]. Our representation of asserton differs from the relational graph (RG) of Peirce in the sense that we consider entities independently, while in RG they are united around lines of identity. Assertion for us is not an indivisible unit, but a relation built by the entities/concepts, which can participate in more than one assertion.

This independence of concepts allows us the possibility to connect them in different relations and in this way to present a text as a whole, rather than different sentences. This independence of the concepts differentiates our SN from Dependency Graph [12], and other graphs similar to it (SNePS, Schank's conceptual dependencies, Conceptual graphs of Sowa) [12], which present one sentence/relation as a frame; while one entity enters into different relations it is repeated in each of them. Fig. 4 presents the example *Sue thinks that Bob believes that a dog is eating a bone* realized in 3 different SN. Our GL graph is in the central position.

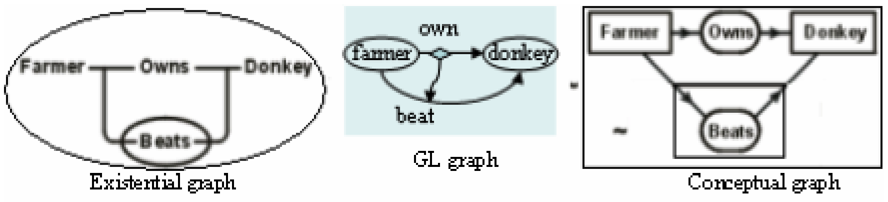


Fig. 3. Three types of SN for: *If a farmer owns a donkey, then he beats it.*

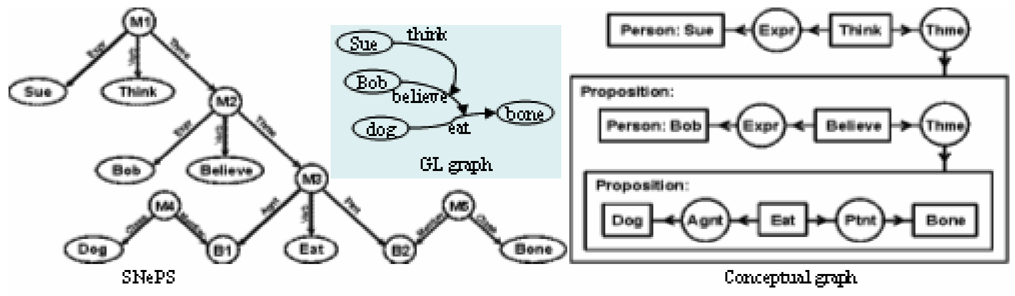


Fig. 4. Three types of SN for: *Sue thinks that Bob believes that a dog is eating a bone*

The advantage of our SN is in attaching to each entity all relations which it participates in, and this is of major importance for the construction of various models. For example, in OO model it is very important to have for some object/concept all actions which it executes, the attributes it possesses. Through GL presentation of text we have the possibility to compare superficial graphic characteristics, which leads to the discovery of different heuristics for possible inclusion of different concepts in different roles in the SE models. For example: the active nodes (the ones which send more predicate arcs than they receive) are serious candidates for objects in OO models. The passive nodes can have different "swimming lanes" in the sequence message chart, if they receive messages (predicate arcs) from more participants in the network, or else they can be presented as an addition to the activity of a basic/active "swimming lane". Our SN is unique also because it uses different graphic notations and positions for different language elements and roles, and in this way implicitly includes semantics without having to use apparent semantic labels. For example, agent is this concept, which gives the beginning of an arc. However, one and the same entity can play a different role in different relations – that of an agent and that of a patient as well. Knowledge supporting and aiding the semantics is saved in tabular presentation, which can include different numbers of columns for the different knowledge appropriate for different applications. The tabular presentation also maintains the order of different assertions (sentences) in the way they appear in the text. In other words the tabular presentation is nothing other than the text kept in full, but structured in three basic columns (Su,Pr,Ob) and with sentences transformed from the passive into the active voice. Other examples of tabular presentation can be seen in [9, 10].

GL can also be considered as Executable Networks [12], because it easily maps into an activity or Use Case Path diagram, which is nothing else but a series of actions and conditions, executed by actors. Finally, our GL can be called Hybrid Networks [12], because it presents different knowledge with different syntax constructs. Its main purpose is to automatically translate textual user requirements written in uncontrolled NL into UML, or other types of SE diagrams.

Future work: We continue to develop and improve our graphic language, seeking the most precise and readable graphic presentation of linguistic, general and domain knowledge all together. We are working on automatic transformation of GL into different types of SE models. We are working on the visualisation and improvement of the interface with the analyst. We are developing a prototype of Integrated Framework for Automatic Analysis of Textual User Requirements, to which GL serves as a basic module.

REFERENCES

[1] Burg, J.F.M. and van de Riet, R.P.: Analyzing Informal Requirements Specifications: A First Step towards Conceptual Modeling, Proc.2 nd International Workshop on Applications of Natural Language to Information Systems, 1996.

[2] Fliedl, G.; Kop, Ch.; Mayerthaler, W.; Mayr, H.C.; Winkler Ch.: The NIBA workflow: From textual requirements specifications to UML-schemata In: ICSSEA '2002.

[3] Fliedl, G.; Kop, Ch.; Mayr, H.C. From Scenarios to KCPM Dynamic Schemas: Aspects of Automatic Mapping. In: NLDB'2003 pp. 91 - 105.

[4] Kop, Ch.; Mayr, H.C.: Mapping Functional Requirements: From Natural Language to Conceptual Schemata, In Proc. of SEA 2002.

[5] Lee, B.-S., Bryant, B.R.: Automated conversion from requirements documentation to an object-oriented formal specification language. In Proc. of SAC 2002.

[6] Mencl, V.: Deriving Behavior Specifications from Textual Use Cases. In Proceedings of WITSE04.

[7] Moreno A.: Object-Oriented Analysis from Textual Specifications, In Proc. of SEKE 97.

[8] Subramaniam K., Liu D., Far H. B. and Eberlein A.: UCDA: Use case driven Development Assistant Tool for Class Model Generation, In SEKE 2004, 324-329

[9] Ilieva M., Ormandjieva O.: Automatic Transition of Natural Language Software Requirements Specification into Formal Presentation, NLDB 2005, pp. 392-397

[10] Ilieva M., Ormandjieva O.: Models Derived from Automatically Analyzed Textual User Requirements. In Proc. of SERA 2006, pp. 13-21.

[11] John F. Sowa, Knowledge Representation: Logical, Philosophical, and Computational Foundations, Brooks Cole Publishing Co., Pacific Grove, CA, ©2000

[12] John F. Sowa, Semantic Networks http://www.jfsowa.com/pubs/semnet.htm

[13] Peirce, Ch. S.: "Manuscript 514", with commentary by J. F. Sowa, www.jfsowa.com/peirce/ms514.htm

[14] Unified Modeling Language (UML 2) http://www.uml.org/

[15] MBT tagger: http://ilk.uvt.nl/~zavrel/tagtest.html

[16] W.Daelemans, S.Buchholz, J.Veenstra: Memory-based shallow parsing. In Proceedings of Computational Natural Language Learning, 1999.

[17] Veenstra J.: Memory-Based Text Chunking. In Proceedings of ACAI, Chania, Greece, 1999.

[18] A.Sampaio, N.Loughran, A.Rashid, P.Rayson: Mining Aspects in Requirements. Workshop on Early Aspects 2005.

A Hybrid Approach for Natural Language Query Translation

Pornpimon Teekayuphun[1] and Ohm Sornil[2]
[1,2] School of Applied Statistics, National Institute of Development Administration
118 Seri Thai Road, Bangkapi, Bangkok, Thailand 10240
[1]ponpimon.t@grads.nida.ac.th, [2] osornil@as.nida.ac.th

Abstract

Natural language interface to database allows users to access information stored in a database by supplying requests expressed in some natural language to the database system. In this paper, an approach for translating queries into Structured Query Language (SQL) is presented in the context of an injury surveillance database, where queries and data are expressed in Thai language. The process consists of three major phases: keyword formation, keyword sense disambiguation, and SQL query formulation. Each phase as well as the overall process are evaluated in details. The results show that the overall query translation achieves an accuracy of 87%.

1 Introduction

Natural language interface to database (NLIDB) [1] is a system that allows a user to access information stored in a database by typing requests expressed in some natural language. The user query is then translated into a query language suitable for the database system, e.g., SQL. Previous works are classified according to query translation techniques: syntax-based, semantic grammar, logical form, and pattern matching approaches.

In syntax-based systems [7, 13], the user input is parsed, and the resulting parse tree is directly mapped to an expression in a query language through mapping rules. The main difficulty for the syntax-based approach is to devise mapping rules that will transform a parse tree directly into the query language.

In semantic grammar systems [11, 12], the output of this system is a semantic tree that can be easily transformed to the query language. Each node in the parse tree may correspond to a semantic object in the database, so this approach heavily relies on the knowledge domain. For this reason, systems based on this approach are very difficult to apply to other knowledge domains.

In logical form systems [5, 14], a natural language question is transformed into an intermediate logical query, expressed in some internal representation language. The logical query is then translated to an expression in the query language. In systems based on the logical form approach, a question undergoes tokenization, tagging, parsing and semantic analysis. The entire process is vulnerable to ill-formed sentences.

Pattern matching approach [6] is proposed as an alternative to grammar based system. In these systems, question patterns are associated with database query patterns. The patterns are created to correspond to the kind of information needed, and the system attempts to recognize these patterns in the input question. Although this approach does not require elaborated parsing and interpretation modules, its shallowness often leads to failures.

In this paper, we present a Thai language interface to a database system which does not rely on syntactic grammar. The system operates on the Injury Surveillance database. The proposed approach extracts the needed information for query translation by attempting to determine all keywords in the user's query. Keywords represent particular database objects (i.e., tables, attributes and attribute values) and components of SQL statement. Prediction by Partial Matching (PPM) is employed to create keywords from the input. However, it is possible for a keyword to have multiple meanings within a domain. This system uses the random walk with restart (RWR) algorithm to help disambiguate keyword meanings. Finally, an SQL query is formulated based on a semantic graph.

The rest of the paper is organized as follows. Section 2 introduces the Injury Surveillance database. Section 3 describes the proposed natural language query translation approach, consisting of the keyword formation, the keyword sense disambiguation, and the query formulation processes. Section 4 presents the performance evaluation. Finally, Section 5 provides concluding remarks for the research.

2 Injury Surveillance Database

The Injury Surveillance (IS) database collects data of patients injured from accidents. The data contained in the IS database are patients' data, accident locations, causes of injuries, first aids, injured organs and treatment outcomes. All database objects are labeled with natural language terms. Besides database objects, components of SQL statements such as operators and functions are also labeled with natural language terms. These terms are used as keywords which represent particular database objects and SQL components.

For SQL formulation purpose, the EER model is transformed to a semantic graph to be used as the knowledge source. The semantic graph is a weighted, undirected graph. A node in the graph represents a database table, and an edge corresponds to a relationship between relational tables as shown in Figure 1. Edge weights are computed from relative frequencies of the relationships used as join condition in the queries as:

$$P(r_{ab}) = \frac{c_{ab}}{c}, \qquad (1)$$

where r_{ab} is a relationship between node a and node b, c_{ab} is the number of times that r_{ab} is used as join condition in the queries, and c is the total number of times that all relationships are used as join condition in a set of training queries. Then the weight of the edge between node a and node b is calculated as:

$$\mathrm{w}(r_{ab}) = \frac{1}{\log \mathrm{P}(\mathrm{r}_{\mathrm{ab}})}. \qquad (2)$$

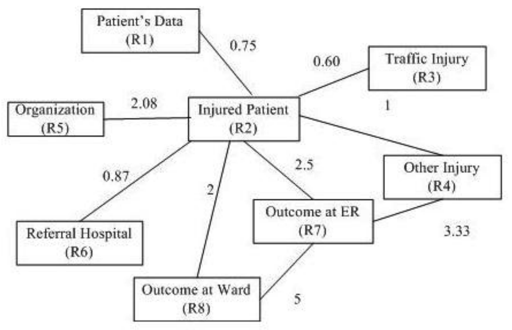


Figure 1. A semantic graph

3 Query Translation and Disambiguation

The proposed natural language query translation process is divided into four phases, as shown in Figure 2: Thai word segmentation, keyword formation, keyword sense disambiguation and SQL formulation.

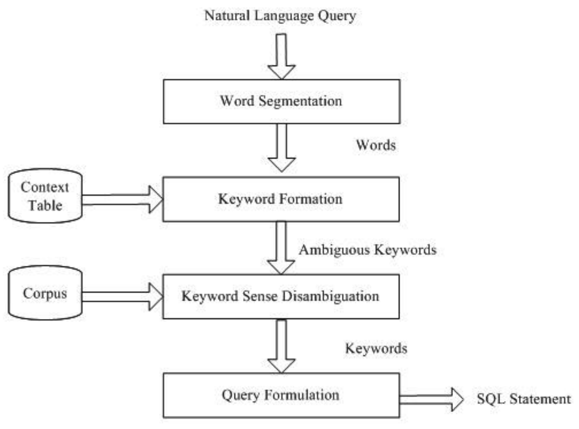


Figure 2. Architecture of the query translation process

3.1 Thai Word Segmentation

A query for the IS database is expressed in Thai language, whose characters are written continuously without space. It is first segmented using a technique described in [10] which employs Prediction by Partial Matching (PPM) to segment a Thai natural language query into syllables. Then, the syllables are combined into words by an application of logistic regression, according to contextual information.

3.2 Keyword Formation

A keyword is defined as a word or phrase that holds a particular meaning within a database domain or a component of SQL statement. This phase is to group words from a user's input to form keywords. Besides extracting keywords from the user's query, this phase attempts to solve the ambiguities of words or phases in the query by grouping them to form keywords which correspond to only one database object.

In order to achieve the above results, first the system maps synonymous words to the same code in order to reduce the number of words processed in further steps. Then, words are grouped into keywords by using Prediction by Partial Matching (PPM) technique. PPM [3], a symbolwise compression scheme, generates a prediction for each input symbol based on its previous contexts. The prediction is encoded in terms of a conditional probability, conditioned on the preceding context. PPM contains predictions, computed from the training data, for the largest context (k) as well as all shorter contexts in a set of context tables.

In the light of PPM, we turn the keyword forming process into the problem of inserting spaces between pairs of words in the query. A training corpus is used to calculate the probabilities of each word based on its previous contexts. Queries in the corpus are manually grouped into phrases or word groups, each represents a specific meaning, by inserting "*" between pairs of words. The probability of each word in the corpus are estimated based on the previous context and maintained in context tables, as shown in Figure 3.

***k*=0**			***k*=2**		
	Count	Prob		Count	Prob
วัน	47.5	0.0002	*ใน → *	32.5	0.203
วันที่	825.5	0.0040	*ใน→ เขต	123.5	0.772
ว่า	106.5	0.0005	*ใน→ prov	2.5	0.016
วิธี	92.5	0.0004	Esc	1.5	0.009
Wname	364.5	0.0018	ในเดือน→ month	107.5	0.995
Esc	182.5	0.0009	Esc	0.5	0.005

***k* =1**		
	Count	Prob
เดือน → month	107.5	0.82
เดือน → ที่	22.5	0.17
Esc	1	0.01

Figure 3: PPM Context Tables

Given an order of k, the algorithm computes the probability of each possible next word (i.e., the next word in the user's query or a space) by considering a context of size k at a time and then proceeds to the next word in the query. The process is repeated until it reaches the end of the query.

We illustrate the insertion of spaces between words using a query "ต้องการ ทราบ จำนวน อุบัติเหตุจราจร" ("How many traffic accidents are there?"). PPM is employed to predict the next possible word by trying to find the context of length *k* (*k*=2 in this example) for words in the context table (i.e. ต้องการ ทราบ -> จำนวน). If the context is not found, it passes the probability of the escape character at this level and goes down one level to the (*k*-1) context table to find the current context of length *k*-1 (i.e., ทราบ -> จำนวน). The process is repeated until a context is found. If it continues to fail to find a context, it may go down to order (-1) corresponding to equiprobable level for which the probability of any next word is 1/|*A*|, where *A* is the number of distinct words.

If, on the other hand, a context of length *q*, 0<=*q*<=*k*, is found, then the probability of this next word is estimated to be the product of probabilities of escape characters at levels *k*, *k*-1, …, *q* + 1, multiplied by the probability for the context found at the *q*-th level.

The model for space insertion becomes a tree-like structure, as shown in Figure 4. To make Figure 4 easy to understand, we transform Thai words in the example query ("ต้องการ ทราบ จำนวน อุบัติเหตุจราจร") into the symbols, "A B C D". After a tree-like structure is created, the algorithm selects as the final result the path with the highest probability at the lowest node. The highest probability path for this example is "AB * C * D" or "ต้องการทราบ * จำนวน * อุบัติเหตุจราจร"

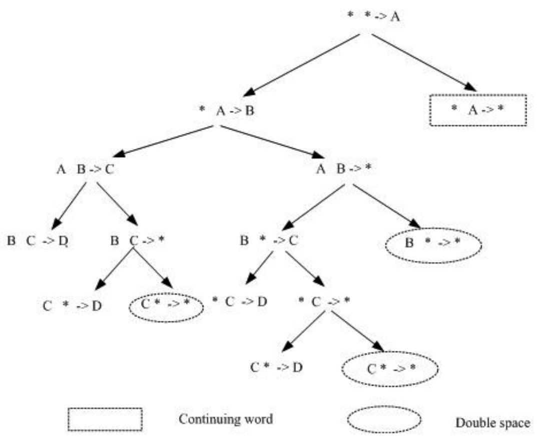


Figure 4. A keyword forming model

To improve the efficiency of the algorithm, the structure can be pruned by the following set of rules 1) Adjacent spaces are not allowed and 2) There must be no space between continuing words. A continuing word is a pair of words appearing in the context table of order 1, but not appearing in that of order 2, and has * between the words. Figure 4 is pruned according to the above rules, thus it does not generate further sub-trees. The nodes surrounded by dotted ellipses and dotted rectangles correspond to rules 1 and 2, respectively.

3.3 Keyword Sense Disambiguation

After the keyword forming process, some keywords are ambiguous, such as 'วันที่' (date) which may correspond to three attributes: "ADATE", "HDATE", "RDATE". In the keyword sense disambiguation, we attempt to map ambiguous keywords to relevant database objects. Before starting this process, stopwords which do not hold any significant meaning in the database are first excluded from this process. We turn the keyword sense disambiguation into a graph ranking problem. The graph (context graph) is constructed from a training set of data. The training query "*s*", database object "*o*" and user's query "*q*" are represented as nodes in the graph, as shown in Figure 5. To solve keyword ambiguities, we have to find relationships between node *q* and every node *o* which corresponds to the ambiguous keyword in the user's query. Then, the random walk with restart (RWR) algorithm [8] is exploited to estimate the affinity of node *o* with respect to node *q*.

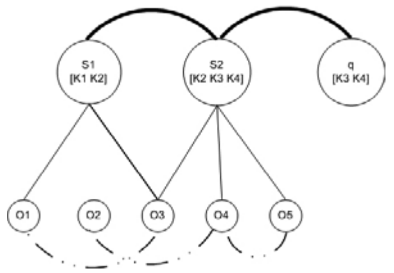


Figure 5. A context graph

3.3.1 Context Graph Construction

The information used for the construction of the context graph is extracted from the training queries. A training query is composed of a set of keywords. Each query must be mapped to a collection of objects in the database. The training queries and database objects are represented as nodes in the graph. All nodes are connected together by 3 types of weighted links:

- Object-Object Link: Connects nodes corresponding to two objects together. Its weight based on the co-occurrence relation of the objects in the training queries. Given two objects, *x* and *y*, with probabilities P(*x*) and P(*y*) respectively, their mutual information, I(*x*,*y*), is calculated as:

$$I(x,y) = \log_2 \frac{P(x,y)}{P(x)P(y)} . \tag{3}$$

In our application, object probabilities P(*x*) and P(*y*), are estimated by the number of observations of object node *x* and *y* in the training corpus and normalizing it by *N*, the size of the corpus. Joint probability, P(*x*,*y*), are estimated by the number of times that *x* and *y* appear in the same queries. If *I(x,y)* is greater than a threshold, a link is placed between *x* and *y*.

- Object-Query Link: Object-Query link is the link between an object and a query whose keyword refers to that object. The weight for the link between an object node o and a query node s is estimated from the number of times that the individual object is referred by keywords in the query s, normalized by the total number of observation of that object in the corpus. Any pair of object and query nodes whose weight is greater than the threshold is connected with an edge.

- Query-Query Link: Query-Query link is based on a similarity relation between the queries. A pair of query nodes will be connected if they are close enough. The distance measure between queries can be computed by the cosine similarity between the vectors of queries [2]. The training query q_i and q_j are represented as keyword vectors. The vector model can estimate the degree of similarity of these queries as the correlation between the vectors $\bar{q}_i$ and $\bar{q}_j$. This correlation can be quantified by the cosine of the angle between these two vectors. That is,

$$sim\left(q_i,q_j\right)=\frac{\vec{q_i}\bullet\vec{q_j}}{\left|\vec{q_i}\right|\times\left|\vec{q_j}\right|}. \qquad (4)$$

If the score of any pair of query nodes is greater than a threshold, a link is placed between them with weight equal to sim(q_i, q_j).

3.3.2 Random Walk with Restart

To solve keyword ambiguities, we have to determine how relevant the object node o is, with respect to the user's query node q. We use the random walk with restart (RWR) algorithm to estimate the affinity of node o to node q. RWR is able to bias toward the restart node. The percentage of time an RWR walker spends on an object node is proportional to the closeness of the object node to the user's query node.

For keyword disambiguation, we want to rank the object node with respect to the user's query node, so we set the user's query node as the restart node. At each node, RWR walker chooses randomly and moves to the next node among the available links with an exception to return to the restart node with probability c. To estimate the affinity of node o with respect to node q, we need to find $u_q(o)$, the steady-state probability that the random walker will reach node o from node q. The object node which corresponds to an ambiguous keyword and owns the highest probability is chosen as the sense of the keyword.

When the user enters a query which contains ambiguous keywords, a query node q is added to the context graph and then is linked to the training query nodes s which are close to it. The distance between node q and node s is measured by the cosine similarity [2] between the query vectors. We set q as the restart node, conduct RWR from node q, and compute the steady-state probability vector $\bar{u}_q = (u_q(1), u_q(2), \ldots, u_q(N))$, where N is the number of nodes in the graph, by using the following equation.

$$\overline{u_q} = \left(1-c\right)A\overline{u_q} + c\overline{v_q}. \qquad (5)$$

Where A is the column-normalized adjacency matrix of the graph; $\bar{v}_q$ is a column vector with its entire N elements zero, except for the entry that corresponds to node q, set this entry to 1; and c is the probability of restarting the random walk from node q. At initial state $\bar{u}_q = \bar{v}_q$, and the equation is evaluated iteratively until $\bar{u}_q$ has not converged. For a given keyword k, the candidate object with the highest value is chosen as the sense of k.

3.4 Query Formulation

To form an SQL query, the system extracts semantic components of keywords. The semantic components of each keyword are composed of type, description, and body [9]. Type defines the kind of the database objects or SQL components to which the keywords are mapped. For this research, there are 10 types: table, attribute, attribute value, half interval operation (e.g. >10), interval operation (e.g. >5 and <7), aggregation operation (count, sum), group by, predicate (a combination of components of type attribute, half interval and interval), group by function (a combination of components of type group by and type table which follow the group by components), and function (a combination of type aggregate and type table or attribute). Description defines name of a specific database objects (e.g. table name) to which this object belongs. Body represents a fragment of SQL statement.

To illustrate this process, we use an example of natural language query "How many referred patients are there, group by discharge date?". The resulting keywords after the keyword forming process with their corresponding semantic components are 'count' (Aggregate, -, -), 'aTable' (Table, R2, 'R2'), 'groupby'(Group by, -,-) , 'rdateC' (Field, R8.rdate, 'R8.rdate'), and 'otherHosp' (Value, R6.htohosp, 'R6.htohosp <> "" '). First, 'count' is combined with 'aTable' to form a function component. Next, 'groupby' is combined with 'rdateC' to form a group by function component. The results of all combinations are: 'count aTable'(Function, R2, 'count(R2.*)'), 'aTable' (Table, R2, 'R2'), 'groupby rdateC' (Group by, R2.rdate, 'group by (R2.rdate)), 'rdateC' (Field, R8.rdate, 'R8.rdate'), 'otherHosp' (Value, R6.htohosp, 'R6.htohosp <> " "').

The components of types table, attribute and function represent data requested by the user and correspond to the SELECT clause. The description component, which indicates the table name, defines the data source and corresponds to the FROM clause. The components of type predicate and attribute value define selection criterions and correspond to the WHERE clause. The resulting SQL statement of the above example query is shown below.

```
SELECT R8.Rdate, Count(R2.HN)
FROM R2, R8, R6
WHERE R6.Hprov <> " " and R2.hn = R6.hn and
        R2.hn = R8.hn
GROUP BY R8.Rdate
```

4 Performance Evaluation

In this section, the proposed method is evaluated against the IS database.

4.1 Performance of the Keyword Formation

Three measures (recall, precision, and error rate) are used to evaluate the effectiveness of the keyword forming technique. Keywords in the test data have been identified manually and are used to measure the performance.

Let N be the number of keywords manually identified,
e be the number of keywords incorrectly identified,
c be the number of keywords correctly identified,
thus $n = c+e$ be the number of keywords identified.

The three measures are defined as follows:

Recall = c/N (6)

Precision = c/n (7)

Error rate = e/n (8)

In order to calculate the above measures, we manually record the starting and ending word positions of each phrase in each test query. Then, the test queries are put into the keyword forming process and again the word positions of the resulting phrases are recorded. For example, for a given sentence, "$W_1W_2W_3W_4W_5W_6W_7$", the starting and ending positions of the hand segmented phrases in the test queries, "W_1 W_2W_3 W_4 W_5W_6 W_7", is recorded as (1,1) (2,3) (4,4) (5,6) (7,7) and that of the phrases derived from the keyword forming process, "W_1 W_2W_3 $W_4W_5W_6$ W_7", is recorded as (1,1) (2,3) (4,6) (7,7). The number of correctly and incorrectly segmented phrases are counted by comparing these two sets of positions, indicated by matched and mismatched pairs, respectively – three correct and two incorrect, in this example.

First, we study the effects of PPM orders on the correctness of the keyword forming phase. The size of the corpus is 19,296 words. The results, in Table 1, show that the higher the order, the better the results, however, with more memory and time required in computation, especially with a large corpus.

Table 1. The effects of PPM orders on the performance of the keyword forming phase

	Order 1	Order 2	Order 3	Order 4	Order 5
Recall	0.88	0.93	0.98	0.99	0.99
Precision	0.85	0.90	0.95	0.97	0.98
Error Rate	0.11	0.10	0.05	0.03	0.02

The results in Figure 6 reveal that the average time used by order 3 is 2.5 seconds, order 4 and order 5 is 6.6 and 14.9 seconds respectively. Since order 3 yields reasonably high values for all three measures which are not much lower than those of order 4 and 5 and takes the least time, PPM order 3 is used in further experiments.

Table 2 shows the performance on three data sets, each with 500 natural language queries. We can see that the results are at least 97.4% in recall, 93.7% in precision, and at most 6.5% in error rate.

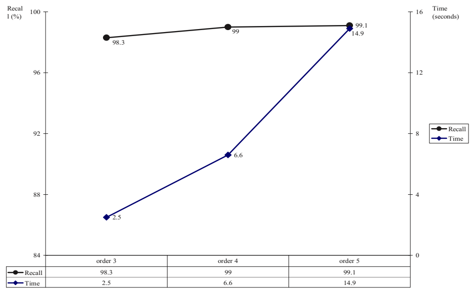


Figure 6: Recall and time taken by order 3, 4 and 5

Table 2. Results of the keyword forming phase

	First Dataset	Second Dataset	Third Dataset
Recall	0.974	0.983	0.977
Precision	0.937	0.954	0.941
error rate	0.065	0.047	0.061

4.2 Performance of Keyword Sense Disambiguation

We evaluate the keyword sense disambiguation by measuring the accuracy that the ambiguous keywords in test queries are correctly mapped to relevant database objects, by measuring whether output objects from this process are really relevant to the intent of the user as specified in the query. We evaluate this phase by using three datasets, each with 100 queries containing at least one ambiguous word. The score of accuracy is computed as following equation,

$$\text{Score}(q_i) = \begin{cases} 1 & \text{if the keyword mapped to the relevant object,} \\ 0 & \text{otherwise.} \end{cases}$$

Then,

$$\text{Overall Score} = \sum_{i=1}^{n} Score(q_i) \qquad (9)$$

Let q_i be a test query i.
Let Score(q_i) be the accuracy score of each test query.
Let n be the number of test query.

We compare the performance of the RWR algorithm with the performance of PageRank [4], a popularly used graph-based ranking algorithm, and a baseline method assigning each ambiguous keyword the most frequently mapped sense in the training data. The results in Table 3 show that the accuracy rate of the proposed method is higher than those of baseline and PageRank. The results demonstrate the suitability of RWR for this task.

Table 3. The keyword sense disambiguation performance of RWR, baseline (BL), and the PageRank algorithm (PR)

	First Dataset			Second Dataset			Third Dataset		
	RWR	BL	PR	RWR	BL	PR	RWR	BL	PR
Accuracy rate	76.0	57.4	54.2	75.5	59.1	55.3	75.5	55.1	52.2

4.3 Performance of Query Formulation

To evaluate the query formation process, 259 queries are randomly selected and translated, assuming perfect keyword forming and keyword sense mapping. 253 queries are formed correctly which account for 97.6%.

4.4 Performance of Overall Process

Finally, we study the overall process and analyze at which steps errors begin. We analyze in details 100 random queries. The performance at each step is the following.

4.4.1 Keyword Formation

For these test queries, Number of keywords manually identified (N), correctly identified (c), incorrectly identified (e), and number of all keywords identified (n) are 1,223, 1,211, 27 and 1,238 respectively. Therefore, the recall, precision and error are 0.99, 0.94, 0.02 respectively.

4.4.2 Keyword Sense Disambiguation

Due to errors from the previous phase and those of this phase, the performance after completing this phase is 89%.

4.4.3 Query Formulation

From errors cascaded from all previous phases, there are only 89 queries which are valid to form the SQL statement. There are 87 out of 89 queries which are formed correctly according to the details in Table 4. Therefore, the overall performance of query translation is 87%.

Table 4. Results of the query formulation

SQL	Correct	Incorrect
Select From	4	
Select From Where	9	
Select From Join	6	
Select From Where Join	38	1
Select From Where Group by	4	
Select From Group by	9	
Select From Join Group by	1	
Select From Where Join Group by	2	
Select Count From Where	4	
Select Count From Where Join	8	1
Select Count From Where Join Group by	1	
Select Count From Where Group by	1	

5 Conclusions

Natural language interface to database allows users to access information stored in a database by supplying requests expressed in natural language to the database system. In this paper, an injury surveillance database is used with queries expressed in Thai language. Three major phases are performed to construct an SQL statement from a natural language query: keyword forming, which attempts to group words in a user query to form keywords whose meanings are relevant in the database context; keyword sense disambiguation, which attempts to map the ambiguous keywords to relevant database objects; and query formulation, which forms SQL queries from the database objects identified in the user query and the database schema. The proposed approach is evaluated with the overall accuracy of 87%.

References

[1] Androutsopoulos. I., Ritchie. G.D., and Thanisch. P., 1995. Natural language interfaces to database – an introduction. Natural Language Engineering, 1(1), 29-81.

[2] Baeza-Yates. R., and Ribeiro-Neto. B., 1999. Modern Information Retrieval. ACM Press. New York.

[3] Bell. T.C, Cleary. J.G., and Witten. I.H., 1990. Text Compression. Prentice Hall. NJ.

[4] Brin. S., and Page. L., 1998. The Anatomy of A Large-Scale Hypertextual Web Search Engine. Computer Networks and ISDN Systems, 30, 1-7.

[5] Grosz. B.J., et al., 1987. TEAM: An experiment in the design of transportable natural-language interfaces. Artificial Intelligence, 32(2), 173-243.

[6] Johnson. T., 1985. Natural Language Computing : The Commercial Applications. Ovum Ltd. London.

[7] Lee. H.D., and Park. J.C., 2002. Interpretation of Natural Language Queries for Relational Database Access with Combinatory Categorial Grammar. International Journal of Computer Processing of Oriental Language, 15(3), 281-304.

[8] Pan. J.Y., Yang. H.J., Faloutsos. C., and Duygulu. P., 2004. GCap: Graph-based Automatic Image Captioning. In Proceedings of the 4th International Workshop on Multimedia Data and Document Engineering (MDDE'04), Washington DC, USA.

[9] Samsonova. M., Pisarev. A., and Blagov. M., 2003. Processing of Natural Language Queries to a Relational Database. Bioinformatics, 19 (1), 241-249.

[10] Sornil. O., and Chiwanarom. P., 2004. Combining prediction by partial matching and logistics regression for Thai word segmentation. In Proceedings of COLING 2004, Geneva, Switzerland.

[11] Waltz. D.L., 1978. An English language question answering system for a large relational database. Communications of the ACM, 21(7), 526 – 539.

[12] Wang. S., Meng. X.F., and Liu. S., 1999. Nchiql: A Chinese natural language query system to databases. In Proceedings of 1999 International Symposium on Database Applications in Non-Traditional Environments (DATE'99). Kyoto, Japan, 453 – 460.

[13] Woods. W.A., Kaplan. R.M., and Webber. B.N., 1972. The lunar sciences natural language information system. Final Report. BBN Report 2378. Bolt Beranek and Newman Inc., Cambridge. Massachusetts.

[14] Wu. X., and Ichikawa. T., 1992. KDA: A knowledge-based database assistant with a query guiding facility. IEEE Transactions on Knowledge and Data Engineering, 4 (5), October, 443-45.

Effective Fault Localization using BP Neural Networks

W. Eric Wong[1], Lei Zhao[1, 2], Yu Qi[1], Kai-Yuan Cai[2], and Jing Dong[1]

[1] Department of Computer Science, University of Texas at Dallas, USA
Email: {ewong, yxq014100, lxz064000, jdong}@utdallas.edu
[2] Department of Automatic Control, Beijing University of Aeronautics and Astronautics, China

Abstract

Fault localization is the most expensive activity of program debugging. It identifies the exact locations of program faults. Finding these faults using an ad-hoc approach or based only on programmers' intuitive guesswork can be very time consuming. A better way is to use a well-justified technique, supported by case studies for its effectiveness, to automatically identify and prioritize suspicious code for an examination of possible fault locations. To do so, we propose the use of a back-propagation (BP) neural network, a machine learning model which has been successful applied to software risk analysis, cost prediction, and reliability estimation, to help programmers effectively locate program faults. A BP neural network is suitable for learning the input-output relationship from a set of data, such as the inputs and the corresponding outputs of a program. We first train a BP neural network with the coverage data (e.g., statement coverage) collected from executing a program, and then we use it to compute the *risk* of each statement, in terms of *its likelihood of containing faults*. Suspicious code is ranked in descending order based on its risk. Programmers will examine such code from the top of the rank to identify faults. A case study using the seven programs in the Siemens suite is conducted. Our results suggest that a BP neural network-based fault localization method is effective in locating program faults.

Keywords: fault localization, program debugging, BP (Back-Propagation) neural network, risk of code, execution slice, successful test, failed test

1. Introduction

During program debugging, fault localization is the activity of identifying the exact locations of program faults. It is a very expensive and time consuming process. Its effectiveness depends on developers' understanding of the program being debugged, their ability of logical judgment, past experience in program debugging, and how suspicious code, in terms of its likelihood of containing faults, is identified and prioritized for an examination of possible fault locations. It is very often that programmers have a huge amount of data collected from program testing available in hand while they are performing program debugging. The challenge is how to use such data to help them effectively locate program faults.

In this paper we propose a fault localization method based on a Back-Propagation (BP) neural network which is one of the most popular neural network models in practice [3]. A BP neural network has a simple structure, which makes it easy to implement by computer programs or circuits. At the same time, BP neural networks have the ability to approximate complicated nonlinear functions [5]. They have been successfully applied in software engineering. For example, Neumann [10] proposes a technique for combining principal component analysis and BP neural networks for software risk analysis. Tadayon [14] presents a BP neural network approach for software cost estimation. Su and Huang [13] report a BP neural network-based study for software reliability estimation. Anderson, Mayrhauser and Mraz [1] apply BP neural networks to predicate the severity levels of program faults that are likely to be uncovered, if any, by each test case. However, to our best knowledge, no studies have used BP neural networks for fault localization.

Using code-based coverage testing tools such as χSuds [16], we can report the test coverage for each test execution. In our proposed method, the coverage data of each test case is focused on the statement coverage in terms of which statements are executed by which test case.[1] The execution result of each test case is also collected. Together, the coverage data and the execution result are used to train a BP neural network so that the network can learn the relationship between them. We also use a set of *virtual test cases* that each covers only one statement in the program. When these coverage data are input into the network, the outputs can be regarded as the likelihood (i.e., risk) of each statement of containing the fault. Suspicious code is ranked in descending order based on its risk. Programmers can examine the statements from the top of the rank one by one in order of their risk. A case study using the programs in the Siemens Suite [12] is conducted to demonstrate the effectiveness of our method. The results when compared with those from other studies are very promising.

The reminder of this paper is organized as follows. Section 2 gives an overview of the back-propagation neural networks. Section 3 explains the proposed fault localization method. In Section 4 we report a case study and the comparison of effectiveness between our method and others. Discussions about the proposed method appear in Section 5. Section 6 lists some related studies. The conclusion and future work are presented in Section 7.

2. An overview of the BP neural networks

A neural network is made up of many simple elements called neurons or nodes. Neurons are connected together with weights on the connections so that they can process information collaboratively and store the information on these weights. Neural networks have many advantages over other models, such as:

1) They have the ability to learn unknown models.
2) They are fault tolerant. Because the information is distributed among the weights on the connections, a few faults in the training data have little influence on the model.
3) They have the ability to adapt themselves to time-variant models.

A BP neural network is a kind of feed forward neural network. Neurons in a BP neural network are connected only with the neurons in adjacent layers. A BP network can learn a complicated nonlinear input-output relationship from a set of sample data (including inputs and the corresponding expected outputs). Figure 1 shows the structure of a three-layer BP neural network. The data flow in a BP neural network is delivered from the input layer, through hidden layer(s), to the output layer, without any feedback.

An *error back-propagation* algorithm is used in the training of a BP neural network. After the neural network is set up, the BP algorithm uses a set of sample data to train the network by the following steps:

1) Input the sample inputs to the neural network to generate the actual outputs. This is a forward processing.
2) Calculate the errors between the actual outputs and the expected outputs given in the sample data. Then, propagate the errors backward to the input layer. In the backward process, the weights on connections are changed such that the errors are reduced.

[1] Coverage with respect to other criteria such as decision, c-uses, and p-uses are also collected and available for program debugging.

The training will be repeated several times in order to ensure the errors are small enough.

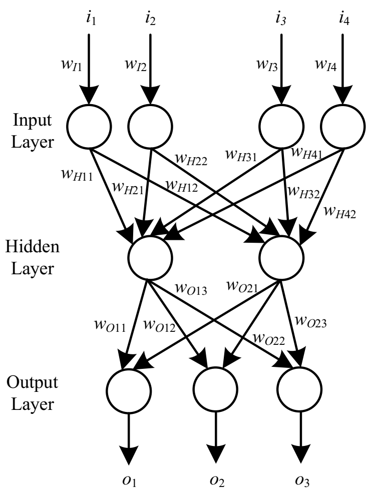


Figure 1. Structure of a BP neural network

In this paper, we use a BP neural network for fault localization for the following reasons:

1) BP neural networks have been proved to be broadly applicable models and have been successfully used in the solutions of several problems such as software risk analysis [10], reliability estimation [13], software cost prediction [14], and severity levels of program faults that could be detected [1].
2) BP neural networks have the ability to approximate complex nonlinear functions. For example, they can be used to simulate the relationship between test coverage (e.g., the statement coverage with respect to each test case in our case) and execution results (e.g., succeed or fail).
3) BP neural networks are trained by a supervised training algorithm (e.g., an *error back-propagation* algorithm). When expected outputs are known, a supervised training algorithm can train the network more accurately and efficiently than an unsupervised training algorithm can. Since in program debugging, we know the expected output of each sample input (i.e., whether the program execution succeeds or fails with respect to each sample input), a supervised neural network is more suitable for the fault localization problem.

3. Proposed Method

3.1. Fault localization with a BP neural network

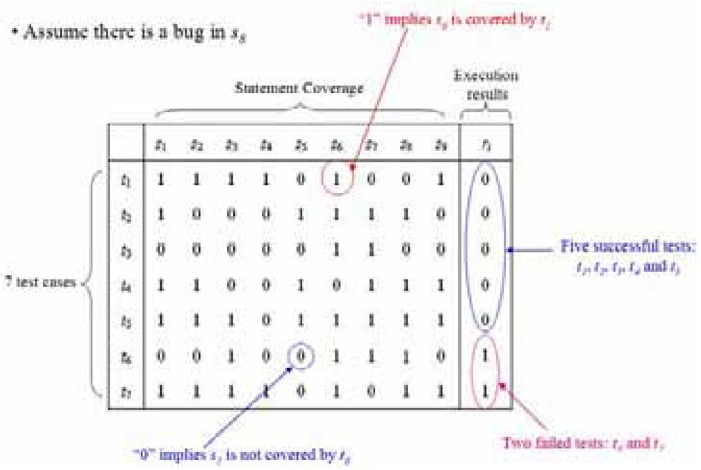
Suppose we have a program P with m executable statements[2] and exactly one fault. Suppose also P is executed on n test cases of which k tests are successful and n-k are failed. Table 1 lists notations that are used for the rest of the paper.

Figure 2 gives an example of coverage data (statement coverage in this case) and execution results that we need for the proposed fault localization method. Each row contains the statement coverage (1 means the statement is covered and 0 means not covered) and the execution result (0 means the execution is successful and 1 means failed) of a test case. For example, statement s_6 is covered by a successful test t_1, and statement s_5 is not covered by a failed test t_6.

[2] All the comments, blank lines, non-executable statements (e.g., function and variable declarations) are excluded for analysis.

Table 1 Symbols used in this section

m	number of executable statements
n	number of test cases
t	a test case executed on P
c_t	the coverage vector of t
$S(t)$	the set of the executable statements covered by the execution of t
r_t	the execution result of t ("successful" or "failed")
s_i	the ith executable statement of the program


• Assume there is a bug in s_5

"1" implies s_6 is covered by t_1

	Statement Coverage									Execution results
	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	r_i
t_1	1	1	1	1	0	1	0	0	1	0
t_2	1	0	0	0	1	1	1	1	0	0
t_3	0	0	0	0	0	1	1	0	0	0
t_4	1	1	0	0	1	0	1	1	1	0
t_5	1	1	1	0	1	1	1	1	1	0
t_6	0	0	1	0	0	1	1	1	0	1
t_7	1	1	1	1	0	1	0	1	1	1

7 test cases

Five successful tests: t_1, t_2, t_3, t_4 and t_5

Two failed tests: t_6 and t_7

"0" implies s_5 is not covered by t_6

Figure 2. Sample coverage data and execution results

Vector c_{t_i} denotes the coverage data obtained from the execution of test case t_i. For example, c_{t_1} = (1, 1, 1, 1, 0, 1, 0, 0, 1) extracted from the first row in Figure 2 gives the statement coverage of the execution of t_1. We refer to c_{t_i} as the *coverage vector* of t_i.

Assume there is a set of *virtual test cases* v_1, v_2, …, v_m whose coverage vectors are $c_{v_1}, \cdots, c_{v_m}$, where

$$\begin{bmatrix} c_{v_1} \\ c_{v_2} \\ \vdots \\ c_{v_m} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \qquad (1)$$

The execution of virtual test case v_i (i=1, 2, …, m) covers only one statement s_i. If the execution of v_i fails, the probability that the fault is contained in s_i is high. This implies that during the fault localization, we should first examine the statements whose corresponding virtual test case fails. However, we cannot find v_1, v_2, …, v_m in the real world. In order to estimate the execution results of these virtual test cases, we build a three-layer BP neural network with m input-layer neurons and one output-layer neuron, and train it using c_{t_i} and r_{t_i} (i=1, 2, …, m) as the input data and corresponding expected output, respectively. The structure of a BP neural network is shown in Figure 3. The *transfer* functions of the neurons are set to the sigmoid function y=1/(1+e^{-x}).

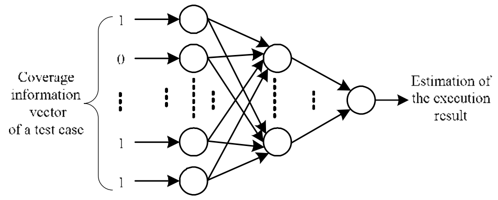
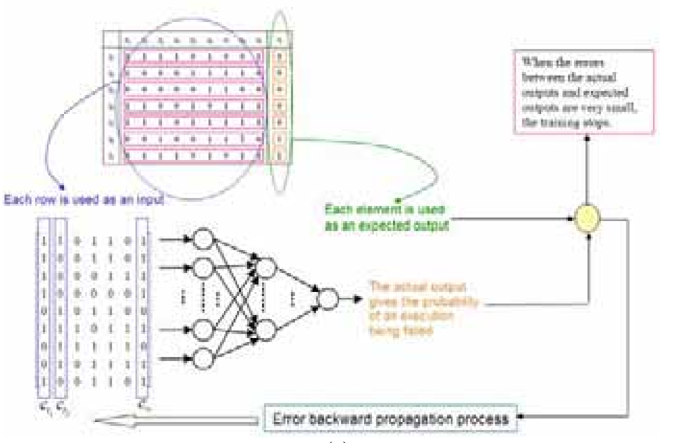
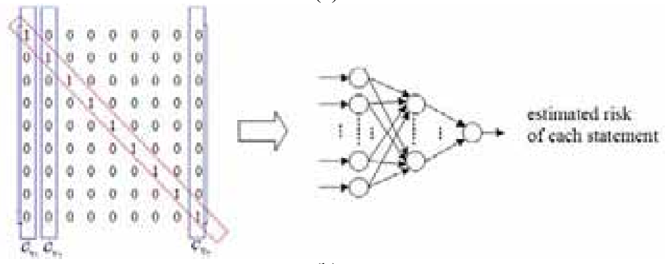

1
0
1
1
Coverage information vector of a test case
Estimation of the execution result

Figure 3. The BP neural network used in our method

When the coverage vectors c_{v_i} (i=1, 2, …, m) of a virtual test case v_i are input to the trained neural network, the output of the neural network (denoted by r'_{v_i}) is an estimation of the execution result of v_i. The value of r'_{v_i} is between 0 and 1. The larger the value of r'_{v_i}, the more likely it is that s_i contains the fault. We can treat r'_{v_i} as the risk of s_i in terms of its likelihood of containing the fault.


When the errors between the actual outputs and expected outputs are very small, the training stops.
Each row is used as an input
Each element is used as an expected output
The actual output gives the probability of an execution being failed
Error backward propagation process

(a)



(b)

Figure 4. (a) train a BP neural network and (b) estimate the risk of each statement using a trained BP neural network

Figure 4 shows the process of computing suspicious statements using a BP neural network. We summarize this part as follows:

Procedure I

1) Build up a BP neural network with m input-layer neurons, three hidden-layer neurons, and one output-layer neuron. The *transfer* function of each neuron is set to the sigmoid function y=1/(1+e^{-x}).
2) Train the neural network using c_{t_i} and r_{t_i} (i=1, 2, …, m) as the input data and the corresponding expected output data, respectively.
3) Input $c_{v_1},\cdots,c_{v_m}$ in Equation (1) into the trained neural network and get the outputs $r'_{v_1},r'_{v_2},\cdots,r'_{v_m}$.
4) Rank s_1, s_2, …, s_m based on $r'_{v_1},r'_{v_2},\cdots,r'_{v_m}$ in descending order and examine the statements one by one from the top until the fault is located.

Let's demonstrate our method through an example. Suppose we have a program with nine statements (m=9) and one fault in statement s_8. We executed seven test cases (n=7) on it, in which two of them failed. Figure 2 shows the coverage data and execution result of each test case. We do the following:

- Build up a BP neural network with nine inputs neurons, one output neuron and three middle-layer neurons. The *transfer* functions of neurons are set to the sigmoid function.
- Train the BP neural network with the coverage data. The first input vector is (1, 1, 1, 1, 0, 1, 0, 0, 1) and the expected output is 0. The second input vector is (1, 0, 0, 0, 1, 1, 1, 1, 0) and the expected output is 0, and so on. Repeat training the network with these data until the errors between expected outputs and actual outputs are small enough (e.g., all are less than 10^{-3}).
- Input the coverage vectors of the virtual test cases into the trained neural network. The output with respect to each statement is shown in the Table 2.

Table 2. Actual output with respect to each statement

statement	output	statement	output	statement	output
s_1	0.0011	s_2	0.9322	s_3	0.9976
s_4	0.9821	s_5	0.0151	s_6	0.0119
s_7	0.3984	s_8	0.8257	s_9	0.0058

- After ranking the statements based on their risk, we get s_3, s_4, s_2, s_8, s_7, s_5, s_6, s_9, s_1. That is, s_3 is most likely to contain the fault.
- Examine the statements one by one in order of s_3, s_4, s_2, s_8, s_7, s_5, s_6, s_9, s_1. The fault will be found when s_8 is examined. In this example, we examined 4 statements before we found the location of the fault.

3.2. Reduce the number of suspicious statements

Our proposed neural network-based fault localization method can be further improved by considering execution slices of failed tests. An execution slice with respect to a given test case in our study contains the set of statements executed by this test. Similarly, it can be the set of all blocks, decisions, c-uses, or p-uses executed by this test, if necessary [15]. Since many of the statements are unrelated to the faults, we can develop additional heuristics to reduce the number of suspicious statements.

In general, the fault should be covered by failed tests, or at least related to the statements covered by failed tests. This implies the most suspicious statements are the statements covered by all the failed executions. Let S_I denote the set of the statements covered by all failed executions. We have

$$S_I = S(t_{f_1}) \cap S(t_{f_2}) \cap \cdots \cap S(t_{f_k}),$$

where $t_{f_1},t_{f_2},\cdots,t_{f_k}$ are the failed test cases and $S(t_{fi})$ is the execution slice of t_{fi}, i.e., the set of statements covered by t_{fi}. In special cases, S_I does not contain the faulty statement(s). A good solution to this problem is to find a failed test t_{f_M} which covers the fewest statements and examine the un-checked statements covered by t_{f_M} (i.e., those in the execution slice of t_{f_M} but not in S_I). For simplicity, let us use S_M as the set of the statements covered by t_{f_M} (i.e. S_M= $S(t_{fM})$). Based on the above discussion, when we are looking for a fault, the statements in S_I should first be examined; if the faulty statement is not there, the statements in S_M - S_I should be examined next. For example, with respect the sample in Figure 2, S_I = $S(t_6)\cap S(t_7)$={s_3, s_6, s_8}, S_M = $S(t_6)$={s_3, s_6, s_7, s_8 }, and S_M - S_I={s_7}. We should first search for the fault in s_3, s_6, s_8. If the fault is not in these three statements, s_7 is the next statement that should be examined.

Integrating this execution-based heuristic with the neural network-based method discussed in Section 3.1, we summary our fault localization method as follows:

Step 1. Get the intersection of all failed execution slices (S_I) and the minimum failed execution slice (S_M).

Step 2. Apply Procedure I (in Section 3.1) to the statements in S_I to examine whether the fault is in these statements. If the fault location is found, go to *Step 4*.

Step 3. Apply Procedure I to the statements in S_M -S_I to examine these statements one by one until the fault is found.

Step 4. Stop.

For discussion purposes, we refer to this method as "BPNN method" in the following sections.

4. Case Study

In this section, we present our case study to show the effectiveness of the BPNN method.

4.1. Programs, test cases and defects

Seven programs in the Siemens Suite were used in our study. These programs are widely used as a benchmark for comparing different fault localization methods [6]. All these seven programs are implemented in the C language. Table 3 gives the information of the seven programs, including number of faulty versions, number of executable statements, number of test cases, and number of functions. There are 132 faulty versions of these seven programs. One faulty version contains only one defect. Some of the defects involve more than one statement. We instrumented and compiled these programs using χSuds. All test cases were re-executed on a SunOS (v5.9) server to collect coverage data and execution results. All the data shown in this section were collected in the executions.

Table 3. Programs in the Siemens suite

Program	No. of faulty versions	No. of executable statements	No. of test cases	No. of functions	Description
print_tokens	7	344	4130	20	Lexical analyzer
print_tokens2	10	355	4115	21	Lexical analyzer
schedule	9	292	2650	18	Priority scheduler
schedule2	10	262	2710	16	Priority scheduler
replace	32	512	5542	21	Pattern replacement
tcas	41	135	1608	8	Altitude separation
tot_info	23	273	1052	16	Information measure

Eleven of the 132 versions were not used in our case study. No test case can reveal the fault in version 10 of "printtokens2," version 32 of "replace," and version 9 of "schedule2". The faults in versions 4 and 6 of "print_tokens" are in the header file instead of in the C file. In versions 19 and 27 of "replace," and versions 1, 4, 6 and 9 of "schedule," all the failures are due to a "segmentation fault." Since our instrumentation tool (χSuds) may lose some coverage data when a segmentation failure occurs, these six versions are also excluded from our study. After removing these versions, we have 121 faulty versions.

4.2. Experiment results

We applied the proposed method, BPNN method, to localizing the faults in these faulty programs. The effectiveness of BPNN method was compared with another method - "Tarantula" which is proposed by Jones and Harrold [6]. Its performance is shown to be better than other fault localization methods (such as set-union, set intersection, nearest-neighbor, and cause-transitions) with respect to the Siemens suite [11,2]. Hereafter, we only focus on the comparison between the effectiveness of the Tarantula technique and that of our method.

In reference [6], the percentage of un-examined code is defined as a score to evaluate the performance of a fault localization method. The percentage of faulty versions in which the method's scores are higher than 99%, 90%, 80%, 70%, 60%, 50%, 40%, 30%, 20%, 10%, and 0 are collected. We notice that when we use Tarantula in a program, some statements have the same rank with the faulty statements. Two curves (Tarantula Best and Tarantula Worst in Figure 5) are used to show the performance of Tarantula. Tarantula Best is obtained by assuming the faulty statement is the first one to be examined among those statements with the same rank. Correspondingly, Tarantula Worst is obtained by assuming the faulty statement is the last one to be examined among those statements with the same rank. The actual performance of Tarantula is between Tarantula Best and Tarantula Worst. The BPNN method can avoid this problem. In most cases, the BP neural network will assign a different risk to each statement. Figure 5 shows that the performance of the BPNN method is much better than Tarantula Worst. With respect to Tarantula Best, BPNN is better in some situations and worse in other situations. However, we must point out in general, it is impractical to assume Tarantula Best will occur. In fact, our intuition suggests that when the size of the program being debugged increases, it is more likely for Tarantula to group more statements with the same suspiciousness. This also makes it even less likely to have the faulty statement to be the first one to be examined among all the statements with the same suspiciousness.

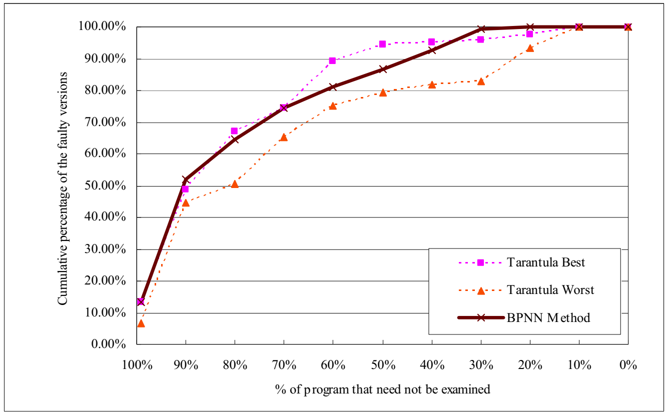


Figure 5. Comparison between BPNN and Tarantula

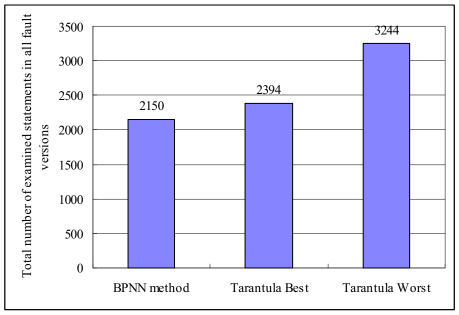


Figure 6. Number of examined statements in all 121 faulty versions

An alternative measure of the performance of a fault localization method is the number of examined statements in all faulty versions. Since the number of examined statements is closely related with the cost of fault localization, more examined statements imply a higher cost for fault localization. So it is reasonable to use this simple measure to compare the effectiveness of BPNN method and Tarantula. The total number of examined statements in the 121 faulty versions using the BPNN method and Tarantula are shown in Figure 6. The bars labeled as Tarantula Best and Tarantula Worst give the minimum and maximum number of statements to be examined using Tarantula. We need to examine 2150 statements to locate the faults in all 121 versions using BPNN. The actual number of examined statements using Tarantula is between 2394 and 3244. This means we can examine 10.19% ~ 33.72% fewer statements if we use BPNN instead of Tarantula.

5. Discussion

In this section, we discuss several issues about the BPNN method based on the results of our study.

5.1. The number of hidden-layer neurons

To the best of our knowledge, there is no theoretical study which provides an algorithm to help us decide how many hidden-layer neurons should be used. Three neurons in hidden layer are used in the BPNN method. This neural network configuration works well in our study. We have also tried different number of neurons in the hidden layer. The results are similar to those obtained when using three hidden-layer neurons, but the training of the BP neural network takes much more time.

5.2. The time complexity

The time complexity of training a neural network is an important aspect when we use neural network models. Leung et al. [7] present an estimation of the time complexity of the BP training algorithm which is in the order of $O(mn^h)$, where $h = 2 \sim 4$ for different programs, m is the number of statements and n is the number of test cases. The average time spent in our study to find the fault location with respect to each of the seven programs is listed in Table 4. All the times were measured on a machine with a 1.86 GHz Core 2 Duo CPU. The time listed in the table does not include the time required for examining the suspicious statements.

Table 4. Average time for locating a fault

Program	Number of Statements$^\$$	Number of test cases	Average Time (in seconds)
print_tokens	177	4130	165.48
print_tokens2	179	4115	158.19
replace	218	2650	238.63
schedule	125	2710	79.03
schedule2	111	5542	294.99
tcas	55	1608	36.50
tot_info	124	1052	41.36

$^\$$The number of statements of each faulty version may vary slightly. Here we use the number of statements in the correct version of each program.

The time required by the BPNN method is acceptable in the fault localization of the programs in Siemens Suite. When the size of the program increases, we may need another approach to reduce the size of the neural network to make the time complexity acceptable. One possible solution is to use function coverage data or module coverage data instead of statement coverage data to find which suspicious functions/modules the fault may be in. After that, statement coverage data can be used to locate faults within those suspicious functions/modules.

5.3. Expandability of the proposed method

An important advantage of the proposed BPNN method is its expandability. The proposed method is based on a neural network, which does not depend on a particular mathematic model. Given the coverage data and the execution results of testing, the BPNN method can identify the relationship between the input and output and thus the suspiciousness of each statement.

The coverage data used in our method can be changed to block coverage data, function coverage data, or module coverage data. Execution count data (the number of times each statement is executed by a test) can also be used in the BPNN method to localize the fault by replacing the 0/1's in the coverage data with the execution counts.

Other program spectra [4] are also feasible to be used in our proposed method, such as Branch Hit Spectra, Branch Count Spectra, Path Hit Spectra, Path Count Spectra, Data-dependence Hit Spectra, and Data-dependence Count Spectra.

6. Related Studies

In this section, we discuss four different fault localization methods: Tarantula, set-intersection, set-union, and nearest neighbor. All these studies use coverage data and execution results of test cases to find the locations of program faults.

Jones and Harrold present a fault localization method named Tarantula in [6]. Tarantula ranks all the statements by their suspiciousness of containing the fault, which is estimated by an experimental formula based on statement coverage data. Developers can examine the statements one by one in descending order of their suspiciousness.

Renieris and Reiss [11] propose a nearest neighbor debugging method. They propose a method to find the most similar successful execution to a given failed execution by the distance between two executions. The code in the difference set between the failed execution and its most similar successful execution should first be examined. If the fault is not there, the code in the adjacent node of the examined nodes in the program dependence graph need to be examined until the fault is localized.

Set union method and set intersection method are also presented in [11]. Set union method tries to find and examine the code that is executed by failed tests but not executed by successful tests. Set intersection method reduces the size of the code that needs be examined by excluding the code that is executed by all successful tests but not by failed tests.

We focus on these four methods because they use the same information as the BPNN method, such that we can compare the performance of BPNN with them. We notice that there are several other studies using other information for fault localization, such as the cause transition method proposed by Cleve and Zeller [2,17,18], the statistical debugging method proposed by Liblit et al. [8], and the SOBER method proposed by Liu et al. [9]. In future work, we need to design an experiment to compare the performances of all the fault localization methods.

7. Conclusion and Future Work

We proposed a neural network-based method for fault localization in this paper. Execution slices are used to reduce the number of suspicious statements, which can improve the performance of the proposed method. Empirical study shows that the proposed method is effective.

In future work, we will further investigate the following problems: 1) how to reduce the size of the BP neural network used in fault localization, 2) how to combine the information of different program spectra to improve the performance of fault localization, and 3) how to cluster the failed test executions to help the proposed method deal with multiple faults in a program.

References:

1. C. Anderson, A. Mayrhauser, and R. Mraz, "On the use of neural networks to guide software testing activities," in

Proceedings of the IEEE International Test Conference on Driving Down the Cost of Test, pp. 720-729, October 1995.

2. H. Cleve and A. Zeller, "Locating causes of program failures," in *Proceedings of the 27th International Conference on Software Engineering*, pp. 342-351, St. Louis, Missouri, May 2005.
3. L. Fausett, *Fundamentals of neural networks: architectures, algorithms, and applications*, Prentice-Hall, 1994.
4. M. J. Harrold, G. Rothermel, K. Sayre, R. Wu, L. Yi, "An empirical investigation of the relationship between spectra differences and regression faults," *Journal of Software Testing, Verification and Reliability*, 10(3):171-194, September 2000.
5. R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Proceedings of 1989 International Joint Conference on Neural Networks*, Washington DC., pp. 593-605, June 1989.
6. J. A. Jones and M. J. Harrold, "Empirical evaluation of the Tarantula automatic fault-localization technique," in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005)*, pp. 273-282, Long Beach, California, November 2005.
7. W. K. Leung, R. Simpson, "Neural metrics-software metrics in artificial neural networks," in *Proceedings on the Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies*, pp. 209-212, Brighton, UK, August 2000.
8. B. Liblit, M. Naik, A. X. Zheng, A. Aiken, and M. I. Jordan, "Scalable statistical bug isolation," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 15-26, Chicago, Illinois, June 2005.
9. C. Liu, L. Fei, X. Yan, J. Han, and S. P. Midkiff, "Statistical debugging: a hypothesis testing-based approach," *IEEE Transactions on Software Engineering*, 32(10):831-848, October 2006.
10. D. E. Neumann, "An enhanced neural network technique for software risk analysis," *IEEE Transactions on Software Engineering*, 28(9):904-912, September 2002.
11. M. Renieres, S. P. Reiss, "Fault localization with nearest neighbor queries," in *Proceedings of 18th IEEE International Conference on Automated Software Engineering*, pp. 30-39, Montreal, Canada, October 2003.
12. The Siemens Suite, http://www-static.cc.gatech.edu/aristotle/Tools/subjects/, January 2007.
13. Y. S. Su and C. Y. Huang, "Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models," *Journal of Systems and Software*, 80(4):606-615, April 2007.
14. N. Tadayon, "Neural network approach for software cost estimation," in *Proceedings of International Conference on Information Technology: Coding and Computing*, pp. 815- 818, April 2005.
15. W. E. Wong and Y. Qi, "Effective program debugging based on execution slices and inter-block data dependency," *Journal of Systems and Software*, 79(7):891-903, July 2006.
16. χSuds User's Manual, Telcordia Technologies, 1998.
17. A. Zeller, "Isolating cause-effect chains from computer programs," in *Proceedings of the 10th ACM SIGSOFT Symposium on Foundations of Software Engineering*, pp. 1-10, Charleston, South Carolina, November 2002.
18. A. Zeller, R. Hildebrandt, "Simplifying and isolating failure-inducing input," *IEEE Transactions on Software Engineering*, 28(2):183-200, February 2002.
19. A. X. Zheng, M. I. Jordan, B. Liblit, M. Naik, and A. Aiken, "Statistical debugging: simultaneous identification of multiple bugs," in *Proceedings of the 23rd international Conference on Machine Learning*, pp. 1105-1112, Pittsburgh, Pennsylvania, June 2006.

Temporal Software Change Prediction Using Neural Networks

Mehdi Amoui, Mazeiar Salehie and Ladan Tahvildari
Software Technologies Applied Research Group
Department of Electrical and Computer Engineering
University of Waterloo, Ontario, Canada
{mamouika, msalehie, ltahvild}@uwaterloo.ca

Abstract

Software change prediction plays a key role in software maintenance and evolution. It is primarily utilized to know "where" the most change-prone entities are, and how the change will be propagated through a system. The results of the prediction are used to plan different tasks in maintenance and evolution, such as re-factoring operation. This paper argues that knowing "when" the changes may happen can give more insight to managers and developers for planning the maintenance activities. To address this issue, a Neural Network-based Temporal Change Prediction (NNTCP) framework is proposed. Such a novel framework determines "where" the changes would be applied (as hot spots), and then adds time dimension to predict "when" it may occur. As a proof of concept, the NNTCP framework is applied to Mozilla as a large-scale open source software. We provide a short discussion on obtained prediction results.

1 Introduction

Software systems are continuously being changed during their lifecycle. The changes may stem from different categories of maintenance/evolution, as discussed in IEEE standard [5]. *Corrective* for fixing bugs, *adaptive* for adapting the software to new environments, *perfective* for updating the software according to requirements' changes, and finally *preventive* for making the software more maintainable. Moreover, a key characteristic of the evolution of large software systems is that changing becomes increasingly difficult over time, the code degrades, and maintenance becomes increasingly hard and expensive through time. These problems are well-known as software aging and code decaying, which have gained increasingly importance both in academia and industry.

Change prediction can be performed for different purposes like reverse engineering and cost estimation [1]. It often ends up to determining the change-prone entities and change propagation patterns of the product [17]. This can be used for future resource allocation, cost estimations as well as specifying *where* maintenance/evolution tasks are "better" to start. This issue in large-scale systems is crucial in order to detect early the potential changes that may occur in future, and consequently reduce the costs and the risks of applying those changes.

The motivation for this work is that knowing *where* the change-prone entities are, may not be enough for the prediction objectives. Knowing *when* the changes would occur can help managers and developers better to plan for later revisions of change-prone entities, which can result in prioritization of changes for those entities. This paper sets out to propose a framework to predict the future change date of a system's entities. To achieve this goal, the proposed framework uses the history data of software changes from the software repository to measure applicable development metrics. These metrics after manipulation is fed in to an artificial neural network which has the output of future change date of the given entity. As a proof of concept, the framework has been tested on Mozilla open source project.

The rest of this paper is organized as follows. Section 2 reviews some works related to main theme of this paper. Section 3 overviews the proposed framework for software change prediction. Section 4 describes our case study and discusses on the obtained results. Section 5 draws several conclusions and discusses on some future works.

2 Related Works

Related works can be classified based on *what* they try to predict and *how* they perform it. In the "what" facet, there are two main categories, namely: i) prediction of the number of changes, bugs or faults, and ii) prediction the probability or rate of the change for software entities. In the "how" facet, prediction models are either stateless or stateful. The former relies only on the current state of the system, while the latter is based on the past history of the system besides of the current state. In addition the predic-

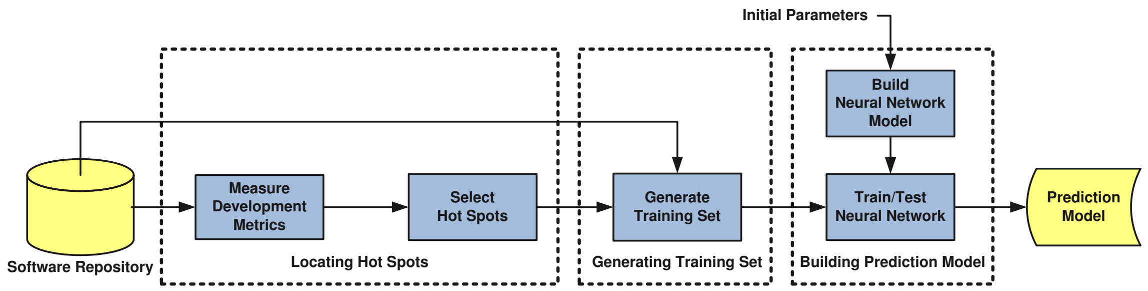


Figure 1. Neural Network-based Temporal Change Prediction (NNTCP) Schema

tion of time-series facts (like software change/fault prediction) can be measured through two approaches: mathematical/statistical models, and AI/soft computing techniques.

For mathematical/statistical approaches there exist several prediction models. Some of these works are based on regression or extended models; among others [2]. One drawbacks for these models is sensitivity to input data. Several works aim to predict probability of change in OO systems using product metrics [11, 15].

There are quite remarkable number of works on software prediction using AI and soft computing methods. These address prediction of quality indicators for example on maintainability and reliability. Khoshgoftar *et al.* [7] use neural networks to predict the number of software development faults for Ada applications by using software metrics as their predictors. In another research, Huang *et al.* [4] proposed a novel neuro-fuzzy constructive cost model (COCOMO) for software cost estimation, or Ramanna [13] used neural networks to predict the number of required changes in a file or a module to achieve the quality assurance standards.

Beside the technique of prediction, the predictors play an important role in an effective prediction. Prior works have identified important predictors for software change predictions (e.g. Khoshgoftaar *et. al.* [6], and Mockus et. al. [10]). The categories of predictors used in prior works are product metrics, development metrics, deployment and usage (DU) metrics, and software and hardware configurations (SH) metrics [9]. In another research, Shepperd [14] compared four prediction techniques: regression, rule induction, nearest neighbor, and neural nets using simulation. Girba *et al.* [1] proposed a meta-model which adds a time layer to structural information to facilitate combining historical data from changelog with other information.

3 Proposed Framework

For addressing the problem, we propose a change prediction framework based on a neural network. We will show how the framework: *Neural Network-based Temporal Change Prediction (NNTCP)* can support the change prediction in a flexible and easy tuning environment. The high level schema of NNTCP is illustrated in Figure 1. This framework is composed from three high-level processes: *locating hot spots*, *generating training data*, and *building prediction model*. The first process, locating hot spots, finds the software entities which are probably change-prone. Generating training data, as the second process, sets out to process change log data of the hot spots and make the appropriate training data set. The last process, building prediction model, is the heart of the prediction framework. This process is responsible to build a neural network prediction model for each hot spot.

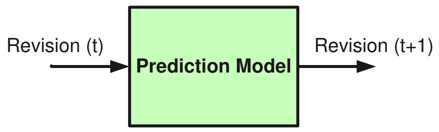


Figure 2. Time-Series based Change Prediction Model

The input of the framework is the software repository. The repository may contain a large set of data that may be useful for locating hot spots as well as prediction. Usually the software repository contains the source code history, the change logs, and the release bundles of the software. On the other hand the output of the framework is a set of prediction models that can predict the revision date of a specific revision for a hot spot entity (See Figure 2). The rest of this section describes the components of this framework in details.

3.1 Locating Hot Spots

Hot sport are software system entities that we are interested to predict their change manner. These entities can be selected by an expert, based on a various implicit properties they have. For example the entities which are recently edited by a new developer, or the ones communicate with a new external interface. But, we are usually interested to put 'hot spot' label on those entities that are proven to have high probability of change.

Probability of change can be measured using wide range of techniques. These techniques address two main approaches. Those which analyze *software product metrics* (like object oriented design metrics) of the last software release, and those that analyze software history data *(software development metrics)* derived from repository logs. For sure we can also think of a third approach which is a combination of both. In our framework, the hot spots will be selected based on development metric values for entities at each level. Pareto law is used to find the change-prone entities based on change-log. This method has been validated by Koru *et al.* [8] for two large-scale case studies including Mozilla, our case study. By Pareto law, we select 20% of entities which have 80% of changes. These entities are hot spots which will be focused for the prediction phase.

3.2 Generating Training Set

In practice, the most valuable information to assist software change prediction is hidden in changelog. It has some similarities with the weather forecast process. We can predict tomorrow's weather based on recent weather changes and the history data of the same day's weather in previous years. In this kind of problems, we are trying to predict a time series based on the retrospective data.

In case of software change prediction, fortunately, the modern source code versioning systems store the source codes and all their corresponding changelog histories. For example, CVS stores the revisions of each file attached by the supporting information on each revision. This supporting information includes the author of each file revision, the date and time stamp, the number of lines added/subtracted from the previous revision, the revision state, and the text description which is provided by the author.

However, considering whole retrospective change information of an entity can decrease the prediction performance. This phenomena is due to the fact that software change rate (software entities, to be more specific) can vary during its lifecycle. There are usually several identical stages in a lifecycle, and each stage has its own change pattern. In addition these stages and their corresponding change patterns are unique for each entity.

Although, we can think of a prediction model that can learn all historical change patterns and predict the current changing pattern, this prediction model will be too complex. To reduce this complexity, we can ignore previous unrelated changing patterns, and narrow down the historical data to the current occurring change pattern and the patterns of the same kind that occurred in the past.

Now the question is how to identify current change pattern and its starting point to cut down the historical data. The most simple and practical answer is to have it as a parameter to be set by a domain expert. The second approach is to analyze the whole software change pattern and be hopeful that the entity we want to predict its changes follows the change pattern of the whole software. This range will be constant for all the entities (hot spots). The final approach is to test the prediction performance (cross validation) with a set of logical training data ranges. The age of the last software release or the average age of all releases is a good example of the value of length parameter.

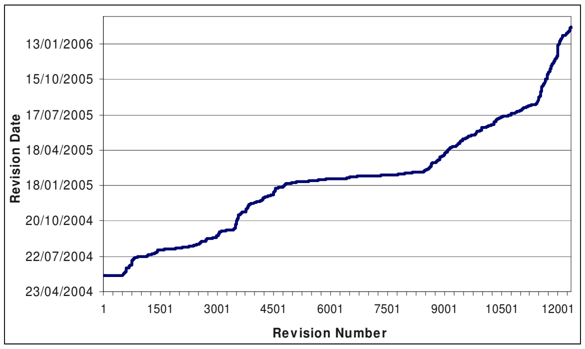


Figure 3. Hibernate3 Change History

As an example, Figure 3 illustrates revision dates of Hibernate open source project[1]. By observing the slope of this function, we can determine the change rate of the software through the time.

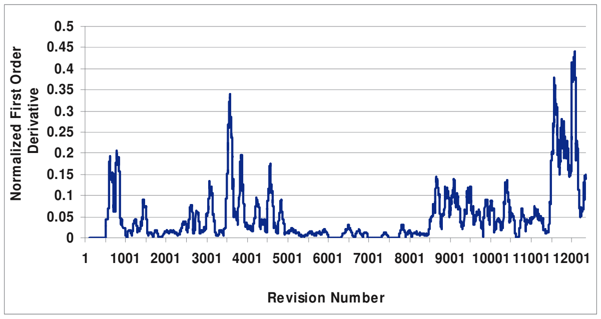


Figure 4. Hibernate3 Consistency Rate

Figure 4 shows the consistency rate as the first derivative of the same function that is normalized and smoothed using a moving average function. The areas with small average values of this function indicate the high change rate periods of software and the high average values indicate the low change rate. For example from Figure 4 we can conclude that the period of time ranging from revision number 5000 to 8500 had a very high change rate. This analysis can assist the user of NNTCP to select the range of revisions for the

[1]http://www.hibernate.org/

training set from a revision number greater or equal to 8500. Moreover, in case of adequate revision numbers the user can set the range from the revision number 11500, where the most recent low change rate state of the software begins.

3.3 Building Prediction Model

The prediction model of NNTCP is based on artificial neural networks. The neural networks are proven to have good performance as prediction models [3, 12]. Although other prediction models, especially the statistical models, can be used for time series prediction, but as we are dealing with a dynamic prediction model for each hot spot, the manual formulation of those approaches is frustrating. On the other hand, the notable strength of neural network lies in its ability to represent both linear and non-linear relationships of input-output sets, and the ability to learn these relationships directly from the data.

The neural network model in NNTCP is called Time-Lagged Feedforward Network (TLFN) [3]. It is a Multi-Layer Perceptron (MLP) with memory components to store past values of the data in the network. The memory components allow the network to learn relationships over time. It is the most common temporal supervised neural network. It consists of multiple layers of neurons connected in a feed-forward fashion.

The training algorithm we used with TLFNs is the Back-propagation Through Time (BPTT) which is more advanced than the standard Backpropagation [3]. In BPTT the network has to be run forward in time until the end of the trajectory and the activation of each neuron must be stored locally in a memory structure for each time step. Then the output error is computed, and the error is backpropagated across the network (as in static backpropagation) through time. This error is used to adjust the weights such that the error decreases with each iteration and the neural model gets closer and closer to producing the desired output. This process is known as *"training"*. The error between the network output and the desired output is computed and fed back to the neural network. The neural network uses this error to adjust its weights such that the error will be decreased. This sequence of events is usually repeated until an acceptable error has been reached or until the network no longer appears to be learning.

Beside the network topology and its training algorithm, there are several other parameters needs to be set for the network. Some of these parameters and their corrsponding values in NNTCP are as bellow:

- **Activation Function:** The BPTT training algorithm requires differentiable, continuous nonlinear activation functions. In our model we use sigmoid function:

$$o = \sigma(s) = 1/(1 + e^{-s}) \tag{1}$$

where:

$$s = \sum_{i=0}^{d} w_i x_i \tag{2}$$

for weights w_i and the inputs x_i.

- **Hidden Layers:** The number of hidden units to use is far from clear. It is suggested to start with a single hidden layer and if the performance was not satisfactory increase the number of hidden layers.

- **Prediction Length:** To use the current and past inputs to predict future desired outputs we have to set the number of samples ahead we want to predict the desired output. In NNTCP we test the prediction performance with prediction length ranging from 2 to 10. In all tests the best performance achieved with the prediction length is set to 2, the minimum valid value.

4 Case Study

As a proof of concept, a case study with a rich CVS history, an open source license, and fairly large-scaled is needed. In order to choose the right case study, we analyze the CVS history of several large scale open source applications. Finally we select the Mozilla Project as our case study.

4.1 Mozilla Overview

The Mozilla project primarily was founded on the basis of Netscape source release, but later on made a new application suite from scratch. The current version of the Mozilla suite has many differences from the initial efforts both in architecture and roadmap. It is now a platform for developers to build new applications, and a modern design in terms of using an advanced architecture for rendering, networking and a general XML-based user interface language. We used Mozilla CVS checkout of 25 Feb. 2007, which includes 111,093 files with 1,033,041 revisions since 1998 (average 9.3 revisions per file).

4.2 Obtained Results

According to NNTCP architecture, the prediction process is based on three main stages. In the first stage we have to identify and select the entities that we are interested to build the prediction model for them. In our case study we perform this task using the Maximum Likelihood Estimation (MLE) model. MLE can be treated as a development metric that uses the counts from the sequences to estimate the distribution. In the MLE model, we compute (predict) the relative frequency of each new event based on the preceding sequence. So, the calculated MLE value for each entity is a good indicator of possible hot spots in a software.

Training Data Range	Train Data Count	From Rev.	Train MSE	Bias	Pred. MSE (Day)
3 Month	54	1.1278	1.54E-03	4.109	16.0
6 Month	115	1.1110.6.40	6.41E-04	2.985	13.0
12 Month	235	1.1186	2.49E-04	2.367	7.0
24 Month	405	1.1036.6.2	2.69E-04	0.466	4.7
48 Month	632	1.824	7.37E-05	3.223	2.5
96 Month (All)	1455	1.1	2.50E-04	13.900	2.7

Table 1. Prediction Results for 'mozilla/layout/base/nsCSSFrameConstructor.cpp' Entity

we computed our MLE probability distributions [16] using this formula:

$$P_{MLE}(E = f_i) = (Count(f_i) + 1)/(N + d) \qquad (3)$$

In (3), $f_i \in D$, N is the size of sequence, $Count(fi)$ is the number of occurrences of f_i and d is the size of domain D. The proportion of times a certain event f_i occurs is called the relative frequency of the event.

We will mark an entity as a hot spot if its P_{MLE} probability exceeds the defined threshold. After detecting the hot spots, we are ready to generate the training data set based on the change history (revision data) of each hot spot. The training set can cover the complete revision history or just a recent part of it. The length of the history data to train the prediction model is arbitrary, but we should make sure that we have enough data to train the network. We also reserve the recent 5% samples of the training data set as a testing data for performance evaluation.

We train and test the prediction model of the hottest detected spot of Mozilla for various data ranges. The results of the change date prediction for this entity using several training data ranges are available in Figure 5 and Table 1. Here we start with full data range using all revision history (96 months), and then shrink the data range to half until there were not enough data to train the network. The results show that for all training data ranges, the calculated train Mean Square Error (MSE) of the desired and the predicted change dates are small. Hence, the neural networks are trained successfully, the trained time series function is not fully overlapped with the actual time series. This results in a shift error between the actual prediction starting point (current date) and the prediction model's starting point. In order to fix this shift error we add a constant bias of this deference to all predicted change dates. The bias value indicates the perception accuracy of the model from the current date, and does not effect the predicted change pattern. The smallest bias value in our test cases was measured for the case of 24 month training data range.

Here we try to predict the next month change dates. The best performance is achieved with the training data of all previous revisions. Though as we shorten the prediction range, we observe that the models based on shorter training sets perform better. We can infer that there should exist an equilibrium date for each entity that minimize the prediction error for a specific range on the training and testing data sets. Although, as far as we meet the required prediction performance, we can set this length to the pre-advised values described in previous section.

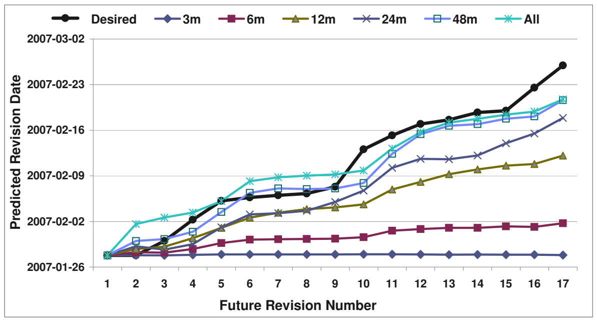


Figure 5. Prediction Results for Mozilla's Hottest Spot Using Various Training Data Ranges

Table 2 lists the current top 10 hot spots of Mozilla detected by MLE. We divide these entities into two sets, and train them using all or the last 24 months change data of Mozilla CVS repository. The results show that in all cases the prediction models follow the change history of the reference entity with a reasonable MSE.

The length of the predictions are set to one month for each entity. This range of test data enables us to count the number of actual and predicted future changes within the range. Therefore, we can also use this model to predict the number or the rate of changes occurred in a specific date range. Here, the prediction range is considered limited, because as we try to predict the change dates farther the error of each change prediction accumulates for the next change prediction. This observation is trivial due to the fact that we are using iterative prediction, and the prediction is based on the previous predicted values which may contain errors.

5 Conclusion

This research presents the Neural Network-based Temporal Change Prediction (NNTCP) to predict the future change dates of software entities. The prediction approach has been performed on a set of hot spots selected based on change-proneness determined by the change history. This selection process relies on the fact that entities (e.g. files) which have

Entity Name	Revision Head	Total Rev.	Training Data Range	MLE	Train Time (sec)	Train MSE	Pred. Bias	Future Change	Pred. Change	Pred. MSE (Day)
layout/base/nsCSSFrameConstructor.cpp	1.1317	1473	All	2.92E-03	72	7.97E-05	36.69	17	36	15.29
layout/html/style/src/nsCSSFrameConstructor.cpp	1.1023	1316	All	2.61E-03	62	2.38E-04	26.03	21	36	6.45
dom/src/base/nsGlobalWindow.cpp	1.911	1273	All	2.52E-03	61	3.05E-04	49.79	13	19	4.33
content/html/document/src/nsHTMLDocument.cpp	3.713	913	All	1.81E-03	40	4.29E-05	26.23	15	18	8.72
layout/html/base/src/nsPresShell.cpp	3.796	962	All	1.91E-03	46	3.14E-04	22.71	18	25	4.40
docshell/base/nsDocShell.cpp	1.829	1077	24 M	2.16E-03	13	2.29E-04	28.79	2	4	4.31
layout/base/nsPresShell.cpp	3.971	1036	24 M	2.05E-03	11	2.82E-04	2.05	6	13	8.31
content/xul/document/src/nsXULDocument.cpp	1.749	965	24 M	1.91E-03	9	2.03E-04	9.61	3	5	4.26
layout/generic/nsBlockFrame.cpp	3.818	946	24 M	1.86E-03	13	1.91E-04	21.33	8	22	5.64
mailnews/imap/src/nsImapMailFolder.cpp	1.750	941	24 M	1.87E-03	7	2.98E-04	3.60	10	16	9.49

Table 2. Prediction Results for Mozilla's Top 10 Hot spots

been changed the most recently can be considered for potential changes in the near future.

The proposed framework generates a prediction model which augments time dimension to the information related to hot spots. This is, similar to a time-series prediction model used in weather forecast and stock-market behavior prediction. The predicted temporal information indicates the pattern of changes, which can give a better picture of the future changes to be planned to managers. This results in prioritizing of tasks and allocating more effectively the resources.

Similar to weather and climate forecasts, respectively for short- and long-term prediction, we may need different predictors and parameters. Our obtained results show clearly this fact for the near and the far future revisions. Another notable point is that a major re-engineering/refactoring stage has a severe impact on the time-based prediction. So, the temporal predicted facts, in their current form, can not be useful in comparison with a regular stage of maintenance/evolution.

Hot spots can be selected using both development and product metrics. Numerous studies have been accomplished for measuring the change-proneness of OO software entities based on product metrics. For instance, Koru *et al.* show that size including lines of code and number of attributes/methods have a remarkable impact on change-proneness [8]. So, one potential extension to the current work is filtering the current set of hot spots using product metrics.

References

[1] T. Gîrba and S. Ducasse. Modeling history to analyze software evolution: Research articles. *J. Softw. Maint. Evol.*, 18(3):207–236, 2006.

[2] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. Predicting fault incidence using software change history. *IEEE Trans. Softw. Eng.*, 26(7):653–661, 2000.

[3] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 1998.

[4] X. Huang, L. F. Capretz, J. Ren, and D. Ho. A neuro-fuzzy model for software cost estimation. In *Proc. of Int. Conf. on Quality Software (QSIC)*, page 126, 2003.

[5] IEEE standard for software maintenance, 1998. URL = http://standards.ieee.org/catalog/olis/se.html.

[6] T. M. Khoshgoftaar, E. B. Allen, K. S. Kalaichelvan, and N. Goel. Predictive modeling of software quality for very large telecom. systems. In *Proc. IEEE Int. Conf. on Communications*, 1996.

[7] T. M. Khoshgoftaar, A. S. Pandya, and H. B. More. A neural network approach for predicting software development faults. In *Proc. of Int. Symposium on Software Reliability Eng.*, pages 83–89, 1992.

[8] A. G. Koru and H. Liu. Identifying and characterizing change-prone classes in two large-scale open-source products. *J. Syst. Softw.*, 80(1):63–73, 2007.

[9] P. L. Li, J. Herbsleb, and M. Shaw. Finding predictors of field defects for open source software systems in commonly available data sources: A case study of openbsd. In *Proc. of IEEE Int. Software Metrics Symposium (METRICS)*, unpaginated, 2005.

[10] A. Mockus, P. Zhang, and P. L. Li. Drivers for customer perceived quality. In *Proc. of Int. Conf. on Software Eng. (ICSE)*, 2005.

[11] G. S. Nikolaos Tsantalis, Alexander Chatzigeorgiou. Predicting the probability of change in object-oriented systems. *IEEE Trans. Softw. Eng.*, 31(7):601–614, 2005.

[12] D. Patterson. *Artificial Neural Networks*. Prentice Hall, Singapore, 1996.

[13] S. Ramanna. Rough neural network for software change prediction. In *Proc. of Int. Conf. on Rough Sets and Current Trends in Computing (TSCTC)*, pages 602–609, 2002.

[14] M. Shepperd and G. Kadoda. Comparing software prediction techniques using simulation. *IEEE Trans. Softw. Eng.*, 27(11):1014–1022, 2001.

[15] A. Sherafat and L. Tahvildari. A probabilistic approach to predict changes in object-oriented software systems. In *Proceedings of the European Conference on Software Maintenance and Reengineering (to appear)*, page TBA, 2007.

[16] F. Stulajter. *Predictions in Time Series Using Regression Models*. Springer Verlag, 2002.

[17] T. Zimmermann, P. Weissgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. *IEEE Trans. on Software Eng.*, (6):429–445, 2005.

Do Neural-Network Question-Answering Systems Have a Role to Play in the Deployment of Real World Information Systems?

Antonio Juarez Alencar, Renata Chaomey Wo, Eber Assis Schmitz and Armando Leite Ferreira
Institute of Mathematics, Electronic Computer Center and The COPPEAD School of Business
Federal University of Rio de Janeiro
P.O. Box 68530 - 21941-590 - Rio de Janeiro - RJ, Brazil
juarezalencar@br.inter.net, renata_wo@yahoo.com.br, eber@nce.ufrj.br, armando@coppead.ufrj.br

Abstract

This article presents a stepwise approach to the construction of web-based hybrid question-answering systems based upon neural-network technologies and natural language processing. These intelligent information systems not only provide high speed answers to questions posed by customers, but also allow customers to receive answers to their questions on a 24/7 basis, provide well conceived standard answers to those questions, and simplify the management of customer support services.

1. Introduction

As Internet users become more numerous, experienced and skillful, and the number of companies embracing e-commerce activities increases worldwide, so does the demand for on-line information about products and services. The nature of information required by Internet customers is varied, ranging from basic technical support to the description of product features, location of outlets and authorized repair services, product warranty coverage, requests for contact with commercial representatives, current status of suggestions and complaints made etc [13].

In order to maintain the balance between the increasing demand for information and the capacity to supply it, companies have resorted to a number of different on-line and off-line strategies, such as: expanding and outsourcing existing customer call centers, installing voice response units (VRUs), making lists of frequently asked questions available to customers, setting up web sites with information about products and services, and providing customer support services over a variety of computerized means of communication such as e-mail, chat services, voice over internet protocol (VOIP) etc [1].

However, recent advances in applied computational intelligence and natural language processing have allowed for the development of question-answering systems (QAS's), i.e. computer systems that automatically detect the existence of requests for information, correctly identify their related content and, using a knowledge base, provide adequate answers to these requests [4]. When a question-answering system seeks human assistance to answer questions that fall beyond its knowledge base, it becomes a hybrid human-machine question-answering system, or hybrid QAS for short.

This article presents a stepwise approach to the construction of web-based hybrid question-answering systems based upon neural-network technology and natural language processing. These systems may be used to meet customers' needs for on-line information when conditions allow, with many advantages over other alternative options. All of this is exemplified by a case study about the construction of a hybrid question-answering system for the "SEBRAE Challenge", a business game involving over 50,000 undergraduate students in five South American countries [12].

2 Conceptual Framework

2.1 Question-Answering Systems

Since Allan Turing (1912-1954) introduced the concept of artificial intelligence while working as a leading cryptanalyst in Bletchley Park, England, during World War II, researchers have passionately pursued the goal of developing computer systems to which one could speak in natural languages and obtain proper answers to questions [5].

However, among the multitude of computer-related subjects that give rise to research and commercial development, QAS has proved to be a particularly challenging topic, where progress has not been made easily. This is partly due to the fact that QAS builds upon the advancements of several scientific fields, including natural language processing, information retrieval and human-computer interaction. Nonetheless, Maybury [11] has identified over ten different types of QAS's that are currently being deployed or researched upon.

2.2 Artificial Neural Networks

Unlike most of the QAS's that one may find in today's commercial and academic world, the QAS proposed in this article makes extensive use of artificial neural network in its inferential reasoning mechanism. In the artificial neural-network paradigm, a mathematical structure composed of values and functions provides a general model for the neuron, a cell that serves as the basic construction block of the human brain and also the brain of many other living beings.

However, while the human neuron receives electrical impulses through its dendrites, deals with them and propagates these impulses to connected neurons, depending on the occurrence of specific neurological conditions, the artificial neural-network neuron receives numerical values through its input nodes, processes these values using a combination of a summation and an activation function, and yields an output, depending on the input values themselves and the weights associated with them.

Figure 1 presents the mathematical structure underlying an artificial neural-network neuron. In the figure, $x_1, \cdots, x_n$ are numerical values that represent input signals, $w_1, \cdots, w_n$ are the weights associated with each input, and f is the activation function that receives the weighted sum of the input signals, i.e. $\sum_{i=1}^{n} x_i\, w_i$ as its parameter

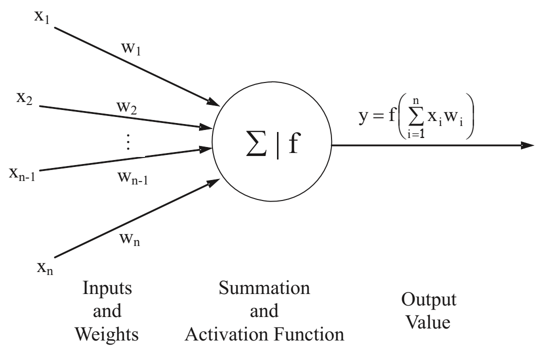


Figure 1. The mathematical structure underlying an artificial neural-network neuron.

Figure 2 presents a feed-forward neural network that receives four input signals, i.e. $x_1, \cdots, x_4$. Each signal has a corresponding input node, whose function is simply to forward the input signals to the nodes in the next layer. In the hidden node layer, the three neurons H_1, H_2 and H_3 process the input signals and send a corresponding output to the next layer. In the output layer the single node O_1 processes the signals sent by the nodes in the hidden layer and generates an output y.

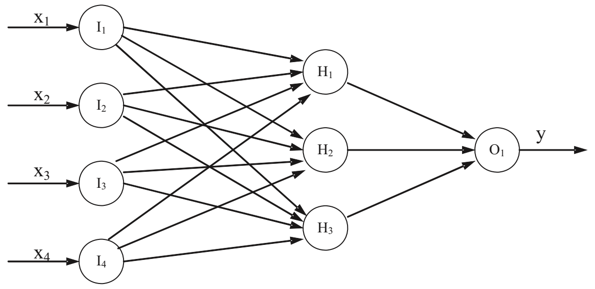


Figure 2. A symbolic representation of an artificial neural network.

The most frequently used mechanism to train an artificial neural network is to randomly choose the weights used by the neurons and present the network to a set of observations about a specific event, together with its corresponding outcome. The difference between the outcome of each observation and the result yielded by the network can then be successively used to adjust the weights in the hidden layers of neurons until an acceptable result is reached.

In this training strategy, called "back-propagation", the change in the weight w connecting unit i to unit j is given by $\Delta w_{j \leftarrow i} = \eta\, e_j\, y_i$, where a unit is either an input, hidden or output node, η is an arbitrarily chosen learning rate, e_j is the error derivative for unit j, and y_i is the output from unit i. The error derivative for unit j is given by $e_i = f'(y_i) \sum_{j=1}^{n} e_j\, w_{j \leftarrow i}$, where f' is the derivative of the activation function. A detailed discussion of back-propagation and other network architectures and training strategies is found in [8]

By choosing the right inputs, number of hidden layers, number of neurons in each layer, number of outputs, kind of activation function and training strategy, one may design an artificial neural network that can precisely approximate any types of function, hence the great usefulness of artificial neural networks. A comprehensive introduction to neural-network technology is provided by Ardib [2].

2.3 The SEBRAE Challenge

The "SEBRAE Challenge" is a virtual business game that simulates decisions that executives face on a daily basis, including the last word on the purchase of raw material, price formation, production, competition analysis, research

& development etc . The game is played in rounds over the Internet by college students of seven different countries in South America, i.e. Brazil, Argentina, Paraguay, Uruguay, Chile, Colombia and Peru. In each round the participants have the opportunity to consider the current situation of their virtual companies, as well as the different forces that act upon the market where they sell products and services [12].

Despite the extensive educational material made available by SEBRAE, participants are free to contact, via e-mail, the team of professionals responsible for the game's execution at any time to resolve questions, make complaints and offer suggestions. To deal with the inflow of e-mails, SEBRAE has brought together a telemarketing service that answers these e-mails as quickly as possible, during office hours. This obviously falls short of satisfying the need for prompt answers to questions that might decide the fate of participants in the game, especially if these questions are addressed to SEBRAE outside office hours.

3 The Method

With a view to providing answers to questions posed by the participants in the SEBRAE Challenge whenever necessary, a hybrid neural-network based question-answering system was built. Figure 3 presents the general structure of this system.

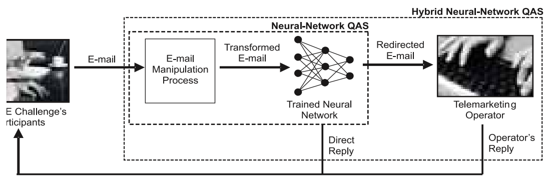


Figure 3. General structure of the SEBRAE Challenge hybrid neural network question answering system.

It should be noted that there are circumstances in which the neural-network question-answering system does not answer questions posed by the participants in the SEBRAE Challenge. Questions that are unlikely to be answered correctly are redirected to a telemarketing operator, who becomes responsible for providing adequate answers.

3.1 The Data

The results presented in this article are based upon a random sample of 1,531 messages sent by the Brazilian participants in the 2004 edition of the SEBRAE Challenge, together with the corresponding reply provided by the telemarketing operators. All messages were written in Portuguese.

The vast majority of these messages contained only a single question posed by participants of the SEBRAE Challenge, while a small number contained double questions, 77 to be precise. These double questions e.mails were later manually transformed into 142 single question messages that became part of the data set used to train the neural network. Therefore, the final training data set consisted of $1,454 + 142 = 1,596$ messages.

When closely examined, the $1,596$ replies provided by the telemarketing operators could be grouped together into 141 distinct answers without any loss of information. Further sample data observation revealed that the 141 distinct answers could be properly replaced by 24 new more general answers and that 5 of these new answers could be used to reply 91.3% of all messages received. Tables 1 and 2 summarize these figures.

Replies	**Quantity**	**Messages Properly Replied**
Originally provided by SEBRAE	1,596	100%
Manually created	24	100%

Table 1. Coverage of the replies to the emails provided by SEBRAE.

Quantity	**Messages Properly Replied**
5	91.3%
19	8.7%
Total	100.0%

Table 2. Coverage of the replies manually crated.

3.2 Data Transformation

Before an e-mail can be presented to a neural network for both training and classification, it must go through a process of transformation. The reasons for this are quite simple: although e-mails are frequently composed of sequences of alphanumeric characters, neural networks are able to deal with numbers only. Moreover, data transformation helps to reduce the complexity involved in the problem of identifying the content of an e-mail.

The data manipulation process to which the e-mails provided by SEBRAE were subjected is composed of six different activities that are executed sequentially, as follows: (a) token separation, (b) translation of abbreviations and foreign language words, (c) translation of words into their phonetic forms, (d) stop-words removal, (e) election of the most relevant words, and (f) codification of words into binary numbers. Figure 4 places these activities into perspective.

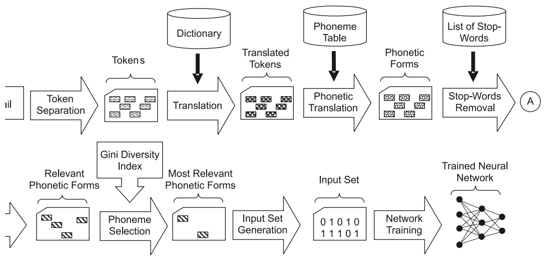


Figure 4. General structure of the SEBRAE Challenge hybrid neural network question answering system.

3.2.1 Token Separation

This consists in separating the different tokens that compose an e-mail, i.e. different sequences of alphanumeric characters without spaces from which the punctuation marks have been removed.

3.2.2 Translations

Although the ability to express the same idea in different forms may initially make a text more interesting to be read, it does add complexity to its understanding. For example, words in a foreign language and abbreviations are frequently used to convey the cultural aspect of a situation and to shorten the text respectively. However, in these circumstances, readers are required to master not only the meaning of words outside their natural tongue, but also the meaning of a considerable range of abbreviations.

Moreover, despite the wide availability of spell-check software on the market, the misspelling of words in e-mails (even those sent by undergraduate students) is not an uncommon event. One way to reduce the extra complexity of having to identify the meaning of words with non-standardized spellings is to translate them into their phonetic form. In this context, for example, “range”, “rrange”, “rannge” etc., may all be translated into “reindэ”, its common phonetic form.

3.2.3 Stop-Words Removal

Stop words are either functional or connective words that lack discrimination power when used to search for information in a data base, classify texts, infer the meaning of electronic messages etc. The following are examples of words that are customarily listed as stop words in English: “a”, “an”, “the”, “in”, “of”, “on”, “are”, “be”, “if”, “into” and “which”. See [6] for lists of stop words in a variety of western languages, including the English language.

3.2.4 Selection of the Most Relevant Words

Despite all the transformations to which the e-mails provided by SEBRAE have been subjected so far, they are likely to contain words with different discrimination power when it comes to the identification of which reply each e-mail requires. While some words may be highly relevant to the task, others will be almost as irrelevant as stop words.

In order to ascertain the discrimination power of a word regarding a specific reply, it suffices to determine how well this word can separate messages that should be properly answered with that reply from the others. This can be achieved with the support of the widely used Gini diversity index. The index was initially proposed by Corrado Gini [7] and later adapted by Breiman *et al.* [3] for the development of classification methods.

In formal terms, for a given set of observations O and a class J of *n* objects, Gini is given by $I(O) = 1 - S$, where $S = \sum_{i=1}^{n} P(j_i|O)^2$ for $j_i \in J$, and $P(j_i|O)$ is the probability of occurrence of objects j_i in *O*. Rokach and Maimon examine both Gini and other alternative ways of estimating the discrimination power of objects in a variety of different circumstances [14].

3.3 The Neural Network

With the support of Neural Dimensions' Neural Solutions software [9] a series of experiments where carried out in order to provide an adequate neural network to be used as the inferential engine that powers the SEBRAE Challenge's hybrid QAS.

All the networks used in these experiments share some common properties. For example, they are all feed-forward networks and have exactly one fully-connected hidden layer of neurons, i.e., every neuron in the hidden layer are connected to all inputs and outputs. Also, following advice from both Sinha [15] and Jefferson *et al.* [10], the number of neurons in the hidden layer is set to be the integer immediately greater or equal to the geometric mean of the number of inputs and outputs[1]. The back-propagation training strategy is used in all experiments.

To offer an unbiased estimation of the error incurred by the neural network in each experiment, the e-mails provided by SEBRAE are divided into two different sets. While the first and larger set is used to train the network, the second and smaller set is used to estimate the number of correct

[1]If p e q are numbers, than $\sqrt{p \times q}$ is the geometric mean of p and q.

answers yielded by the network. Table 9 presents the corresponding figures.

Training Set		Test Set		Total	
Qtd.	%	Qtd.	%	Qtd.	%
1,117	70.0	479	30.0	1,596	100.0

Table 9: Quantity of e-mails used to train and test the neural network.

Because only five standard replies can be used to answer the vast majority of all the e-mails provided by SEBRAE (see Table 4) and one can rely on the support of a telemarketing operator to answers messages for which a standard reply has not been properly provided, in all experiments the neural network was trained to deal with e-mails requiring six different kinds of standard replies.

While five of these replies are precisely those used to answer 91.3% of the e-mails provided by SEBRAE, the sixth provides a pseudo standard reply to the remaining 8.7% of the e-mails. The idea behind the use of a pseudo reply is that the actual reply to certain questions posed by participants of the SEBRAE Challenge should be provided by a telemarketing operator in a customized manner. Therefore, when one of these remaining e-mails is detected, instead of indicating which standard reply should be used by the QAS, the neural network concedes that a telemarketing operator should be called in.

4 The Experiments and Related Results

In the same manner that a human being can still understand the meaning of a sentence from which words or sounds have been removed, an artificial neural network can be trained to identify which reply should be used to answer an e-mail from a reduced subset of its contents. Therefore, a question that developers of neural-network QAS's face is to determine the smallest set of words that hold sufficient discrimination power to allow e-mails to be properly replied to.

With regard to the development of the SEBRAE hybrid QAS, five different experiments have been carried out bearing this objective in mind. In each experiment, the set of most relevant words were gradually enlarged until no further improvement in the performance of the network could be reached.

In order to avoid bias towards certain standard replies to the detriment of others, the same number of most relevant words has been selected for each of the six replies the network was trained to recognize when to use. Table 10 contains the total number of relevant words used in each experiment.

For example, in the first experiment, the twenty-three most relevant words were selected to train the network. In the fifth and last experiment, the fifty most relevant words

Experiment	Most Relevant Words	
	Per Standard Reply	Total
1	5	23
2	8	33
3	10	38
4	12	44
5	15	50

Table 10: Quantity of most relevant words used in each experiment.

have been selected. However, because some words have discrimination power to recognize the need to use more than one standard reply, the total number of relevant words in each experiment is smaller than six times the number of relevant words per standard reply.

It should be noted that because each most relevant word corresponds to a neural-network input, the number of inputs used in each experiment varies accordingly. However, the number of outputs is always the same, i.e. the number of replies the network is trained to recognize (six in this case).

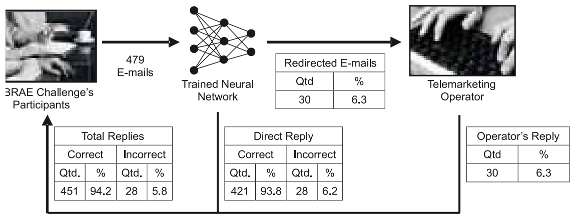
Table 3 presents the accuracy of the answers provided by the neural network in the different experiments. While in the first experiment the network was able to properly answer 91.9% of the e-mails in the test set, this figure rose to 93.8% in the last experiment, in which a larger set of most relevant words was used. Unfortunately, neither further increase in the number of most relevant words nor in the number of hidden neuron layers was followed by an increase in the number of properly answered e-mails.

Experiment	Replies					
	Correct		Incorrect		Total	
	Qtd.	%	Qtd.	%	Qtd.	%
1	397	91.9	35	8.1	432	100.0
2	405	92.5	33	7.7	438	100.0
3	412	93.2	30	6.8	442	100.0
4	414	93.7	28	6.3	442	100.0
5	421	93.8	28	6.2	449	100.0

Table 3. Precision of the answers provided by the neural network in each experiment.

In addition, the data presented in Table 5 reveal two important pieces of information. First, the accuracy of the answers provided by the network increases much more slowly than the increase in the number of most relevant words. This is due to the fact that words with decreasing discrimination power were added to the set of most relevant words as the experiments proceeded.

Second, not all e-mails were answered by the network. Only the e-mails to which an actual standard reply has been provided are directly replied by the network. E-mails that do not fit into this category are forwarded to a telemarketing operator, who becomes responsible for providing the right answer. In the last experiment, for instance, the telemarketing operator was required to answer 30 e-mails, i.e. out of the 479 e-mails used for testing, the neural network answered 449 and the telemarketing operator only 30. Figure 5 summarizes these statistics.



Redirected E-mails	
Qtd	%
30	6.3

Total Replies			
Correct		Incorrect	
Qtd.	%	Qtd.	%
451	94.2	28	5.8

Direct Reply			
Correct		Incorrect	
Qtd.	%	Qtd.	%
421	93.8	28	6.2

Operator's Reply	
Qtd	%
30	6.3

Figure 5. Summary of the last experiment in the pursue of a suitable inferential engine.

5 Conclusions

One of the major differences between the traditional mortar and brick market of products and services we have been accustomed to and the virtual market created with the introduction of the Internet is the easiness with which customers may compare the benefits of acquiring products and services from a multitude of different providers. In the very competitive virtual market, where many products and services are offered in a global scale, it is becoming increasingly difficult for organizations to avoid providing services on any other basis but 24/7. After all, one of the main characteristics of the Internet is its around-the-clock availability.

During the process of product acquisition over the Internet, it is natural that potential buyers may want to get in touch with customer support services virtually in search of information that they could not find in a web site. Failure in promptly providing such information may lead to customer dissatisfaction and cognitive dissonance, which may adversely impact the customer-to-business relationship and, as result, sales.

In this article, we have demonstrated the viability of building hybrid question-answering systems that are able to properly answer questions posed by customers via e-mail in natural language. These systems, that use artificial neural networks as their inferential engine, are not difficult to be built and maintained, yielding accurate results even in the presence of noise. If the necessary data is available, there is no major obstacle preventing companies from enjoying the benefits of hybrid question-answering systems, but their desire to do so.

References

[1] J. Anton and M. Murphy. *Managing Web-Based Customer Experiences: Self-Service Integrated with Assisted-Service*. The Anton Press, August 2003.

[2] M. A. Arbib. *The Handbook of Brain Theory and Neural Networks*. The MIT Press, 2^{nd} edition, November 2002.

[3] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Chapman & Hall/CRC Press, January 1984.

[4] C. Champion. Taking advantage of web self-care to meet client needs. *Customer Interaction Solutions*, 22(5):48–50, November 2003.

[5] J. Copeland. A brief history of computing. Information avalibale on the Internet at www.alanturing.net/turing_archive/pages/Reference%20Articles/BriefHistofComp.html, June 2000. Site last visited on January 20^{th}, 2007.

[6] I. I. d'Informatique - University of Neuchatel. Have a look at the CLEF site (european languages) or NTCIR (asian languages) providing other information about multilingual retrieval. Information available on the Internet at www.unine.ch/info/clef/, Switzerland, 2005. Site last visited on February 8^{th}, 2007.

[7] C. Gini. *Memorie di Metodologica Statistica*, volume I, chapter Variabilità e concentrazione, pages 359–408. Dott. A. Giuffrè, Milano, Italy, 1939. Article written in Italian.

[8] M. M. Gupta, L. Jin, and N. Homma. *Static and Dynamic Neural Networks: From Fundamentals to Advanced Theory*. John Wiley & Sons-IEEE Press, 1^{st} edition, April 2003.

[9] N. Inc. NeuroSolutions. Information available on the internet at www.nd.com, 2006. Site last visited on March 16^{th}, 2007.

[10] M. F. Jefferson, N. Pendleton, S. B. Lucas, and M. A. Horan. *Artificial Neural Networks in Cancer Diagnosis, Prognosis, and Patient Management*, chapter 5: The Use of Genetic Algorithm Neural Network (GANN) for Prognosis in Surgically Treated Nonsmall Cell Lung Cancer, pages 39–53. CRC, 1^{st} edition, June 2001.

[11] M. T. Maybury. *New Directions in Question Answering*, chapter 1: Question Anserwing: An Introduction, pages 3–14. AAI Press / The MIT Press, 2004.

[12] SEBRAE – The Brazilian Micro and Small Business Support Service. Desafio sebrae 2005. Information available on the internet at www.desafio.sebrae.com.br, 2005. Site last visited on February 14^{th}, 2007. Text written in Portuguese.

[13] S. Negasha, T. Ryanb, and M. Igbariab. Quality and effectiveness in web-based customer support systems. *Information & Management*, 40:757–768, 2003.

[14] L. Rokach and O. Maimon. Top-down induction of decision trees classifiers: A survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 35(4):476–487, November 2005.

[15] A. K. Sinha. Short term load forecasting using artificial neural networks. In *IEEE International Conference on Industrial Technology 2000*, pages 548–553, Mumbai, India, 19-22 January 2000. Jaico Publishing House.

Knowledge Conversion in Software Development

Olivier Gendreau, Pierre N. Robillard
Computer engineering department
École Polytechnique de Montréal
Montréal, Québec, Canada
{olivier.gendreau, pierre-n.robillard}@polymtl.ca

Abstract

Software processes can be categorized in two types of approach: engineering-based processes, criticized for restraining creativity, and agile methodologies, criticized for being often unpredictable. This paper proposes a conciliatory view of software processes by analysing human cognitive activities. Our approach, based on the SECI knowledge conversion process, defines eight knowledge conversion types. The approach is then tested on a project developed by a team of undergraduate students enrolled in a capstone project during the 2006 winter semester at École Polytechnique de Montréal. The knowledge perspective of the capstone project mainly stresses the importance of creativity and information sharing in collaborative projects.

1. Introduction

Software processes can be categorized in two types of approach. First, there are engineering-based processes such as Rational Unified Process (RUP), Unified Process for EDUcation (UPEDU) or Model-Based Architecting and Software Engineering (MBASE). These processes are mainly criticized for restraining creativity [1]. Second, there are agile methodologies such as Extreme Programming (XP), Scrum, Dynamic Systems Development Method (DSDM), Adaptive Software Development (ASD), Crystal Methodologies, Lean Development (LD), Feature Driven Development (FDD) and Agile Modeling (AM). These methodologies are mainly criticized for being often unpredictable [2]. In an effort to conciliate these two software process views, an hybrid approach as emerged mainly by mixing known advantages of the two approaches, OpenUP [3] being an example.

The elaboration of a software process implies conformance to different standards, conventions and best practices of the software field. Metamodels such as Software Process Engineering Metamodel (SPEM), OPEN Process Framework (OPF), Software Process Improvement Capability dEtermination for Object-Oriented/Component-Based Software Development (OOSPICE) and LiveNet can be used to assure uniform representation between processes. But in order to assess software quality or maturity, reference models such as ISO 9001, Software Capability Maturity Model (SW-CMM), Capability Maturity Model Integration (CMMI) and ISO/IEC 15504 can be used. Also, software process improvement (SPI) models are available such as Personal Software Process (PSP), Team Software Process (TSP) and Initiating, Diagnosing, Establishing, Acting and Learning (IDEAL).

The SPI field can also be categorized in two approaches : blueprints and recipes [4]. We can see similarities between traditional processes and blueprint SPI as between agile methodologies and recipe SPI. In fact, traditional processes and blueprint SPI insist on process and prescription while agile methodologies and recipe SPI insist on people and adaptation.

By reviewing the software process literature, we can conclude that most of the processes stand in the engineered-based process/agile methodologies debate continuum. Since we know that software development is knowledge-intensive [5], in order to manage this duality, the integration of some knowledge engineering concepts should be a valuable avenue. Recurring problems of feedback loops could be corrected by improving and adding software practices in order to achieve an integrated knowledge management. Of course, some software disciplines are more subject to benefit from this type of approach. In particular, software design is a discipline particularly involved with cognitive synchronisation. This knowledge-centered activity is intended to assure that team-mates share the same mental model, the same representation of concepts [6].

This paper proposes a conciliatory view of software processes by analysing human cognitive activities. To achieve this purpose, we suggest analysing knowledge conversion in software processes. For now, there is not much literature on that matter which leads us to believe in the originality of the approach.

Section 2 presents useful knowledge concepts related to knowledge conversion. Section 3 details the proposed approach to analyze knowledge conversion in software development and section 4 presents the observations from a capstone project.

2. Knowledge concepts

More than two decades ago, Alvin Toffler [7] predicted the imminence of a society based on knowledge as a source of power. Nowadays, we can state that knowledge actually is a strategic tool for enterprises seeking improved profits [8]. Therefore, knowledge management is clearly an important matter.

Information and knowledge are vital forces in today's organizations [9] and particularly software organizations. In fact, information and knowledge are essential during software development lifecycle, predominantly during design. In this regard, Kahkonen and Abrahamsson [10] demonstrated the link between software processes and knowledge creation.

Knowledge is context-specific, meaning it depends on time and space [11]. Information becomes knowledge when it is interpreted by someone, associate to a context and anchored to one's commitments [12]. We can categorize knowledge in two types: explicit and tacit [13]. Explicit knowledge can be expressed in formal and systematic language. It can be processed and stored relatively easily [14], contrarily to tacit knowledge which is highly personal and hard to formalise. It is deeply rooted into one's actions, experience and values [15].

Nonaka, Toyama and Konno [12] developed a dynamic process enabling an organization to create, maintain and exploit knowledge. The Unified Model of Dynamic Knowledge Creation includes three elements: the SECI process, which is a knowledge creation process through tacit and explicit knowledge conversion; *Ba*, which is the knowledge creation context; and knowledge assets including every organization-specific resources essential to value creation.

An organization creates knowledge from the interaction between tacit and explicit knowledge, called knowledge conversion. There are four types of knowledge conversion: socialisation (from tacit knowledge to tacit knowledge); externalisation (from tacit knowledge to explicit knowledge); combination (from explicit knowledge to explicit knowledge); and internalisation (from explicit knowledge to tacit knowledge) [12]. Socialisation relates to the conversion of new tacit knowledge from past experiences. Externalisation is the process of crystallising knowledge by making tacit knowledge explicit. Combination relates to converting explicit knowledge to more complex or systematic explicit knowledge. Internalisation happens when someone embodies explicit knowledge into tacit knowledge.

In an organizational perspective, in order to create knowledge, it is crucial to put strategies in place. Regarding that matter, Choi and Lee [8] found links between knowledge management and SECI knowledge creation process. They conclude that a human strategy is more appropriate for socialisation while a system strategy is more appropriate for combination. As for externalisation and internalisation, a balanced human-system strategy is more appropriate.

Von Krogh, Nonaka and Aben [16] developed four knowledge management strategies depending on knowledge domain and knowledge process. A knowledge domain includes data, information, explicit knowledge and tacit knowledge. A knowledge process can either be knowledge creation or knowledge transfer.

A knowledge gap is a problem without any known solution. When that occurs, key resources are responsible of gathering data and information and to create the necessary knowledge in order to solve the problem. We can easily relate knowledge gap resolution to software design.

A knowledge strategy consists of using knowledge processes (transfer or creation) to knowledge domains (existing or new) in order to achieve strategic goals such as efficiency or innovation. The optimisation strategy is used to transfer knowledge domains already existing in the organization. The expansion strategy is used to create knowledge based on data, information and knowledge already existing. The appropriation strategy is used to build new knowledge domains with external sources. The exploration strategy gives to one or many teams the responsibility to build new knowledge domains from scratch.

To conclude, as said earlier, literature regarding knowledge management application to software processes is sparse which demonstrates the originality of this paper.

3. Proposed approach

We propose to analyse software development by using an approach based on the SECI process developed by Nonaka, Toyama and Konno [12]. Table I specifies eight knowledge types which will allow describing knowledge conversion in software development projects.

TABLE I. KNOWLEDGE TYPES

Knowledge type	Conversion details	Description
KH	No conversion involved	Know-how
CTT	Tacit to tacit	Information sharing
TE	Tacit to explicit	Knowledge crystallisation
CTE	CTT and tacit to explicit	Collaborative knowledge crystallisation
EE	Explicit to explicit	Combination, review
CEE	CTT and explicit to explicit	Collaborative combination
ET	Explicit to tacit	Learning
CET	CTT and explicit to tacit	Collaborative learning

In table I, we can see that each of the four SECI types of knowledge conversion can either occur in individual or collective contexts with the exception of CTT. In our view, tacit to tacit knowledge conversion can not be done individually in software development. This is because such a thing involves "philosophical thinking" which is not relevant in software development. Another particularity is that KH (know-how) does not involve any knowledge conversion because it is related to procedural activities. CTT is a team activity used to exchange or synchronize information. TE and CTE involve knowledge crystallisation, which means that information is formalised such as when structured information is written in a document. EE and CEE are related to activities not requiring much creativity (tacit knowledge) such as reviewing artefacts or coding from a detailed design. Finally, ET and CET are related to learning activities, such as training.

Table II specifies both engineering-based and agile software activities knowledge types.

TABLE II. KNOWLEDGE TYPES AND PROCESSES ACTIVITIES

Knowledge Type	Engineering-based activity	Agile activity
KH	Execute tests, Manage working environment, Integrate system	
CTT	Conduct a meeting, Discuss	
TE and CTE	Design components, Define architecture, Fix major code defect, Plan project's development, Write an artifact	Code, Refactor (major), Plan project's development
EE and CEE	Code, Review, Fix minor code defect, Debug	Refactor (minor), Review, Debug
ET and CET	Attend a training, Learn	

An important concern with knowledge type determination is that knowledge types are not orthogonal. In other words, process activities can be associated with more than one knowledge type. Consequently, the strategy is to determine, for a given activity, the dominant knowledge type. For instance, coding, depending on the process used, can be perceived as a TE or an EE knowledge type. For engineering-based processes, coding mostly involves translating detailed design into code. Therefore, the dominant knowledge type is TE during the design activity and EE during the coding activity. However, in most agile processes, coding is considered a creative activity involving both design and coding. In such a process, the dominant knowledge type is TE and EE is less important than in engineering-based processes. Consequently, for projects needing massive artefacts production, such as in critical systems development, engineering-based processes are more adequate than agile methodologies.

4. Observations from capstone project

The knowledge type approach is tested on a project developed by a team of five undergraduate students enrolled in a capstone project during the 2006 winter semester. It is an optional project-oriented course offered to senior-year students in computer engineering at École Polytechnique de Montréal. The course's particularity is that the project is defined by an industrial partner, this time an international aeronautic company. The project is based on a business needs document supplied by the industrial partner. An engineer from the participating organization meets the students once a week to guide them in developing the software product. The students follow an engineering-based software process derived from the UPEDU.

The methodology used to measure developers effort is a more elaborate version of the effort time slip method popularized by Perry, Staudenmayer and Votta [17] and improved afterwards by Germain and Robillard [18]. Each time a team member executes a task, she/he must log information in a time slip *token* containing the date, start and end time, participants involved in the task, process details and task description. The aeronautic project contained about 1500 *tokens* for a total effort of over one thousand hours.

Table III presents the knowledge type distribution for the project undertaken by the students based on their time slips.

TABLE III. KNOWLEDGE TYPE DISTRIBUTION

Knowledge type	%
KH	7
CTT	14
TE	19
CTE	4
EE	23
CEE	19
ET	12
CET	2

Table III provides some insight into three types of activities that are basic to any software development processes: collaborative activities, creativity and learning.

First, the importance of the collaborative activities spent on this project is obtained by summing up the four knowledge types that involve information exchange: CTT, CTE, CEE, and CET. It is found that although it is an engineering-based project, almost 40% of the team effort is spent on collaborative activities.

Second, creativity is a major endeavour in a software development project. A first level evaluation of the amount of team effort involved in creative activities in this project is to consider all knowledge types that are initiated by tacit knowledge, which are CTT, CTE and TE. These three tacit knowledge types count for 37% of the total team effort. Interestingly, almost half of the creativity effort is done collaboratively.

Finally, in most projects, some learning is needed unless team members are already expert in the field. Learning is characterised by the conversion of explicit knowledge into tacit knowledge. Some of the learning occurred during discussion (CTT) but it is difficult to evaluate its importance. Consequently, for this project, learning activities count for at least 14% (TE and CTE) of the total team effort.

5. Conclusions and future work

Knowledge type approach, based on the SECI process, provides a cognitive perspective to software engineering activities. It defines eight knowledge conversion types (KH, CTT, TE, CTE, EE, CEE, ET, and CET). By recording effort for each process activities, it is possible to evaluate the knowledge type distribution in a project's development.

The knowledge perspective of the capstone project stresses the importance of creativity and information sharing in collaborative projects. More detailed analyses could provide enough insight to enable the design of practices that will be tailored to the creativity needed in projects. Ongoing researches are aiming at refining our approach and at applying it to different software projects in order to analyze correlation between knowledge type distribution and project or software process type.

References

[1] A. J. Bailetti and J. Liu, "Comparing software development processes using information theory," presented at Portland International Conference on Management of Engineering and Technology (PICMET'03), Portland, OR, USA, 2003.

[2] M. C. Paulk, "Extreme programming from a CMM perspective," *Software, IEEE*, vol. 18, pp. 19-26, 2001.

[3] Eclipse Foundation, "Eclipse Process Framework Project (EPF)," 2006.

[4] I. Aaen, "Software process improvement: Blueprints versus recipes," *IEEE Software*, vol. 20, pp. 86-93, 2003.

[5] P. N. Robillard, "The Role of Software in Software Development," *Communications of the ACM*, vol. 42, pp. 87-92, 1999.

[6] P. N. Robillard, "Opportunistic Problem Solving in Software Engineering," *Software, IEEE*, vol. 22, pp. 60-67, 2005.

[7] A. Toffler, *Powershift: Knowledge, Wealth and Violence at the Edge of the 21st Century*. New York: Bantam Books, 1990.

[8] B. Choi and H. Lee, "Knowledge management strategy and its link to knowledge creation process," *Expert Systems with Applications*, vol. 23, pp. 173-187, 2002.

[9] E. Trandsen and K. Vickery, "Learning on demand," *Journal of Knowledge Management*, vol. 1, pp. 169-80, 1998.

[10] T. Kahkonen and P. Abrahamsson, "Digging into the fundamentals of extreme programming building the theoretical base for agile methods," presented at 29th Euromicro Conference, Belek-Antalya, Turkey, 2003.

[11] F. A. Hayek, "The Use of Knowledge in Society," *The American Economic Review*, vol. 35, pp. 519-530, 1945.

[12] I. Nonaka, R. Toyama, and N. Konno, "SECI, ba and leadership: a unified model of dynamic knowledge creation," *Long Range Planning*, vol. 33, pp. 5-34, 2000.

[13] M. Polanyi, "The Tacit Dimension," in *Knowledge in Organizations*. Boston: Butterworth-Heinemann, 1997, pp. 135-146.

[14] R. Williams, "Narratives of knowledge and intelligence ... beyond the tacit and explicit," *Journal of Knowledge Management*, vol. 10, pp. 81-99, 2006.

[15] D. A. Schon, *The Reflective Practitioner*. New York: Basic Books, 1983.

[16] G. Von Krogh, I. Nonaka, and M. Aben, "Making the most of your company's knowledge: A strategic framework," *Long Range Planning*, vol. 34, pp. 421-439, 2001.

[17] D. E. Perry, N. A. Staudenmayer, and L. G. Votta, "People, organizations, and process improvement," *IEEE Software*, vol. 11, pp. 36-45, 1994.

[18] E. Germain and P. N. Robillard, "Engineering-based processes and agile methodologies for software development: a comparative case study," *Journal of Systems and Software*, vol. 75, pp. 17-27, 2005.

A LANGUAGE FACILITATING INFORMAL REASONING ABOUT PROGRAMS

J. Nelson Rushton and Dwayne Towell
Computer Science Department
Texas Tech University

Acknowledgment: This research was conducted under funding from NASA grant 1314-44-C522

Abstract: It is argued that traditional programming languages are not designed to be verifiable. A new language is described, *L*, which is deliberately designed to facilitate reasoning about its programs, and, in particular, informal reasoning as customarily practiced in the discipline of mathematics. Formal and informal logics are discussed for reasoning about *L* programs, and several properties of the language are proven.

1. Introduction

This paper introduces *L*, a functional language designed to facilitate informal reasoning about programs written in it. Section 2 draws a comparison, first attributed to Dijkstra [4], between programming and program verification -- noting that they are essentially the same activity. Section 3 points out that most programming languages are either not designed with verifiability in mind, or are designed for formal verifiability, leaving an open opportunity for languages designed deliberately for informal verifiability, which in practice is the most likely to be used. Section 4 argues that well known high level languages, in particular Lisp, Haskell, and Prolog, do not have a semantics, nor a methodology of use, which facilitates reasoning about programs in a precise way. Section 5 defines the language *L*, and Section 6 discusses formal and informal logics for reasoning about its programs. The resulting language can be used as a specification language, and can be implemented as a server for performing computation and inference, in much the same way SQL acts as a database server. It can also be used as a meta-language for more highly sugared functional languages. It is *not* intended to grow into a language for writing executable programs, like Common Lisp or LPA Prolog, which would invalidate its quintessential properties of verifiability.

2 . Programming as theorem proving.

Computer programming is hard. In 2001 the Standish Group estimated that around 30% of commercial software projects were successful -- defined as being completed within 10 percent of the committed cost and schedule and delivering all of its intended functions. 50% of projects were substantially over budget or schedule, or compromised in functionality -- while the remaining 20% delivered nothing at all. Moreover, it is not just large programs that are difficult. The rates of success, failure, and total failure for commercial projects are about the same, in our experience, as for freshmen students, on programs requiring one or two pages of code and comments. Generally speaking, a person writing a program *essentially different from any they have written before* often they finds that they are in for more work than it first appears -- or, not infrequently, that the task turns out to be infeasible.

This section presents a perspective which helps explain why computer programming is so deceptively difficult. The perspective is not a new one; it goes back at least to a paper of Dijkstra [4], but it bears refreshing. In a nutshell the explanation is this: a program appears as an encoding of an algorithm, along with comments which (hopefully) specify the desired behavior of that algorithm. Neither the code nor the specification may look particularly daunting. But as Dijkstra noted, the code and specification together constitute a mathematical proposition -- namely, that a certain algorithm has a certain property. The real complexity of such a proposition, if it is claimed to be a true one, lies not in its statement but in the *reasoning which convinces us of its truth*. That is, the real complexity if a program lies not in the code, not in the specification, but in the logic which bridges the two. In the current practice of programming, this logic is usually not explicit. This is why programs are *deceptively* complex. It may be imagined that the complexity which meets the eye in the code and comments of a program, compared with that which is hidden in the logical connection between the two, is analogous to the complexity of the statement of a mathematical theorem compared with its proof.

Moreover, the reasoning which connects code with its specification cannot be thought of as an ornament, or an ivory tower ideal. On the contrary, this reasoning, with the implicit complexity surrounding it, is inescapably at the heart of a programmer's thought process. Code, after all, is not written at random. The programmer's job *begins* with a desire to encode an algorithm certain properties -- and is finished if and when it is reasonably believed to have these properties. This is simply what it means to program. Explicit program verification may not be practiced much, but *reasoning about computation* is the essence of the activity of programming. The associated task of *typing code which results from this reasoning* is an indispensable, but comparatively trivial part of the process.

3. Verifiability of programming languages.

In the early days of computer programming, there was little choice on how computations were represented – it was necessary to meet the machine on its own terms. Over time, technology afforded more flexibility in representations, and consequently more and more work was done using "high level" languages – that is, languages more centered around the human's reasoning process, as opposed to the machine. However, the mainstream languages have evolved in small, ad-hoc steps, with each resembling its predecessor, from machine code to assembly, to FORTRAN, C, C++, and Java. Each step in this evolution appeared to its proponents to simplify the encodings of algorithms; but we claim less attention, if any, was paid to the properties of these languages for explicit, precise reasoning about those algorithms. Not surprisingly, programs in these languages are difficult to reason about, as argued by Cooke et. al. [3]

While mainstream languages were evolving in small steps from machine code, a smaller group of scholars was taking a different approach to language design, centered from the outset around functional and logical notation -- time tested methods for writing human-understandable calculations. Notable efforts in this direction have included Lisp, Prolog, and Haskell. Again, however, attention to the proof theories associated with these languages appear to have been side issues, and largely if not entirely post hoc. None of the original papers on Lisp, Prolog, or Haskell, for example, mentions the verification of programs written in the respective languages. In an often-cited paper [5], John Hughs argues, for example, that modularity is the key advantage of functional programming, but makes no explicit connection between modularity and explicit verification. In Section 4, we will argue that this lack of attention has resulted in barriers to reasoning about code in these languages, some related inherently to the syntax and semantics of the languages, and some related to the methodology of their use.

Recently, yet a third group languages has been developed, paying deliberate concern to the formal verifiability of their programs. Notable efforts in this direction include PVS and ACL2[1]. These lines of research have yielded valuable practical results in the design of safety-critical (or, generally, correctness-critical) hardware and software.

At the same time, the use of formal verification for arbitrary software properties is not likely to catch on in general practice, because the discovery of nontrivial formal proofs -- by man, machine, or cooperation of the two -- is *grindingly slow.* PVS and ACL2, for example, require a great deal of expertise in formal logic to even begin learning. Functions in both languages by default fail to compile absent a formal proof they are well-defined and total on their domains. PVS disallows mutual recursion, for reasons related to this.

For such reasons, formal logic is scarcely used, for the actual discovery and verification of properties of computations, even by experts in formal logic. Gödel's famous theorem of incompleteness, for example, is *about* formal logic, but was originally stated and proven informally. A formal proof followed eight years later by Chwistek [1], which was an interesting development, but not because it seriously boosted peoples' confidence in the theorem. The well-checked informal proof was good enough. As a rule, when people -- even logicians -- want to understand the properties of computations, they reason about them informally.

All off the above points to a gap in technology, in terms of programming languages deliberately designed for explicit, precise, but informal reasoning about defined functions -- of the sort customary in the profession of mathematics. The goal of this paper is to define such a language. The following section points out how the paradigms of existing well known functional and logical languages can be improved on in this respect.

4. On traditional functional and logical languages.

We will say that a language is *declarative* to the extent that its primary components (functions, clauses, etc.) resemble propositions which characterize the behavior of the programs in which they appear. Obviously, procedural and procedural/OO languages do not come close to having this property. Lisp is declarative in some cases, for example, the following function

[1] We mention ACL2 as a separate language from Lisp because (1) it has some special syntactic features distinct from pure Lisp, and (2) it is restricted to a subset of Lisp and a programming methodology substantially different from Lisp employed in practically all in other contexts.

```
(defun f (m n)
  (cond (and (>= m 0) (> n 0) (< m n))
  (cond
    ((eq m 0) n)
    (t (f (mod n m) m))
   )))
```

maps simply to a formal proposition

```
m ∈ N & n ∈ N & m>=0 & n>0 & m<=n ->
(
     (m=0 -> f(m, n) = n)
     &
    (~(m=0) ->  f(m,n) = f(m, n mod m))
)
```

which, along with axioms for built in operations, characterizes the function well enough to deduce its essential property (*viz*., that under suitable assumptions it returns the greatest common divisor of *m* and *n*.) For functions written in pure Lisp (without side effects), the mapping is practically trivial, with two exceptions:

(1) Unlike logical statements, guarded expressions appearing as arguments of Lisp's `cond` operator are tested and used in a particular order. Thus associated conditional propositions are systematically, though usually only slightly, more complex than the Lisp definitions.
(2) Lisp functions often in practice omit the types of their arguments, which must be included in the associated propositions to assure correct behavior.

Regarding (1), McCarthy [7] is to be credited with introducing conditionals into formal functional languages. They were, however, introduced with a syntax and semantics that shrinks code at the expense of making the language less declarative, in the sense defined above. Also, in this respect, Lisp's conditionals differ from those of traditional mathematics. Ordinary mathematical conditionals are written, for example, as follows:

```
For x in R, let
f(x) =
     x+5 , when x<= 0
     x-5, when 0< x < 10
     x-1, when x >= 10
```

When evaluating, say, *f*(21) using this definition, the reader is not obliged to read the branches of the definition in any particular order. If a branch is found whose guard is satisfied, other branches can be ignored even if they are written first. The author of a function, on the other hand, is obligated to write a function which is well defined, in the sense that the value computed is independent of the order in which branches are examined. For example, if someone wrote the following:

```
For x in R, let
f(x) =
     x+5 , when x <= 1
     x-5, when  x >= -1
```

then he would be violating his agreement to play the game of mathematics! A syntactically analogous Lisp expression using *cond* is syntactically valid, yields a genuine function, and, in fact, would not be unusual. The conventions of mathematics, it seems, accept the risk of ill-defined functions in trade for a higher level of declarativeness.

The second issue is largely methodological. Our premise is that if one seriously intends to reason about a function definition, then it must be accompanied by conditions on its arguments which guarantee that it returns a value -- that is, a domain must be specified on which the function is total. Both PVS and ACL2 enforce this requirement. The reason is that even the most basic platitudes of formal and informal reasoning, such as *f*(*x*)=*f*(*x*), fail when applied to terms which do not have values. For example, if the Lisp function *f* defined above, the term `(= (f 1 (2 3)) (f 1 (2 3)))` returns not *true,* but an error. Normally, using typed variables takes much of the nuisance out of specifying domains. But because Lisp variables are untyped, one must either leave the functions partial, which impedes reasoning, or explicitly insert the missing type information using Boolean functions as types, which tends to clutter the code.

Haskell, in contrast with Lisp, uses type signatures for functions, which helps manage the complexity of keeping up with their domains. Haskell's conditionals, like those of Lisp, are ordered; and moreover Haskell allows multiple definitions of a function symbol, which are again ordered. For example, in the definition shown here

```
factorial(0) = 1

factorial(n) = n * factorial (n-1)
```

the second equation is ignored in computing factorial(0). On its face, it says something which is false about the behavior of the program, but its real meaning can be extracted considering the fact that the equations are tried in order. Thus the semantics of the equations appearing in a Haskell program depend implicitly on their preceding context. The dependency is simple, and by itself not likely to cause confusion. Moreover, it saves keystrokes in the definition. But we claim the savings in code size is just what is paid back as a cognitive load when reasoning about the code, considering the context dependency of the equations. This explains why mathematicians have never adopted the same convention, despite the more succinct definitions it affords.

Another attribute of Haskell is lazy evaluation. This goes hand in hand with the deliberate use of partial functions, which can be used to do things that might requite more code otherwise. Given Dijkstra's analogy that the code is "half a proposition", and the real complexity is in verification, this raises the question of the effect of deliberate use of Lazy evaluation on the resulting verification logic. Early research in this area was done by Simon Thompson [8]. As usual, this work was after the fact with respect to development of language being reasoned about, in this case Miranda. The resulting logic was both sophisticated and bulky, and Thompson as well as later authors (e.g., [6]) have commented on the degree of complexity added to the logic by lazy evaluation. Even if the formal verification logic were only *equally* complex with, say, predicate calculus, the mere fact that it is highly nonstandard would have serious consequences for informal verification.

Prolog, the best known logical language, is used in practice with side effects, including *assert* and *cut*, which require dynamic logic to be reasoned about -- much as a program in FORTRAN or Java. If we consider the pure form of the language, Prolog is quasi-declarative in the sense that inferences actually made by the interpreter are sound with respect to the *prima facia* interpretation of the code as a set of propositions (modulo only the absence occurs-check in unification algorithm of most implementations). However, the use of depth first SLDNF resolution is incomplete, so that the Prolog programmer must keep the evaluation mechanism in mind while coding, rendering the language largely a procedural one. This is not unexpected since the choice of depth first SLDNF was largely motivated by a desire to allow side effects to be predictably deployed.

5. The language *L*.

We now define a language which builds on the traditions of functional languages, but incorporates some of the insights above in being designed from the ground up to facilitate informal verifiability. We begin with the syntax. Nonterminal symbols appear in italics. Meta-symbols are ::= ( ) * |. Formal symbols in the target language resembling meta-symbols are enclosed in single quotes.

identifier ::= *letter* (*letter* | *digit*)*
functionName ::= *identifier* (^ *identifier*)*
naturalNumber ::= *digit digit**
integer ::= *naturalNumber* | - *naturalNumber*
rational ::= *integer* | *integer* / *naturalNumber*
scalar ::= *identifier* | *functionName* | *rational*
variable ::= ? *identifier*
head ::= *functionName* '(' *variable* (,*variable*)* ')'

term ::= *variable* | *scalar*
| *functionName*'(' *term* (,*term*)* ')'
equation ::= `equals`(*term*, *term*)
guard ::= *equation*
body ::= *term*
programRule ::= `rule`(*head*, *body*, *guard*)
program ::= *programRule**
derivedRule ::= `rule`(*term*, *term*, *term*)
rule ::= *programRule* | *derivedRule*

We consider the following function symbols built in. Numbers following the function names indicate their arities. The following have their usual interpretations on rational numbers:

```
add/2        subtract/2
minus/1      multiply/2
divide/2     modulus/2
floor/1      less/2
greater/2    lessEq/2
greatEq/2
```

For every nonnegative integer *n*, and terms $a_1,\ldots,a_n$, `roster`($a_1,\ldots,a_n$) is interpreted as the set $\{a_1,\ldots,a_n\}$.

The following have their usual interpretations on sets:

```
union/2  setDifference/2 memberOf/2
```

The following Boolean operators have their classical interpretations on the identifiers `true` and `false`:

```
and/2    or/2     not/1
```

The function symbol `equals/2` has its usual interpretation on rational numbers, identifiers, and finite sets formed from these. The unary functions

```
rational/1  set/1 natural/1   int/1
```

return `true` if their argument is a rational number, set, natural number, or integer respectively. Otherwise they return `false`.

The semantic value of a program is a set of equations which are its "consequences", as defined below. Consequences of a program *P* of the form `equals`(*T*, *D*), where *D* is a "literal" (defined below), are thought of as defining the value *D* which an interpreter must return when querying the program *P* with the term *T*.

A ***literal*** is a term written using only scalars and the `roster` function symbol. For any set S of rules and equations, define $\varphi(S)$ to be the closure of S under the substitution law of equality for literals -- precisely, $\varphi(S)$ is the smallest set T of equations and rules containing S, such that if `equals`(a,b) or `equals`(b,a) appears in T, where either a or b is a literal, and X is an equation or rule appearing in T, then Y appears in T where Y is obtained from X by substituting one or more instances of a for b.

An ***instance*** of a program rule R is the term obtained by replacing all variables occurring in R consistently with literals. For a program P and a set S of equations and rules, define $\pi(P,S)$ as the closure of S under application of rules of P -- that is, $\pi(P,S)$ is the smallest set T of rules and equations containing S such that if `rule`($H,B,$`true`) appears in T, `equals`(H,B) appears in T.

An ***axiom of L*** is an equation whose first argument is written with a single built-in function symbol whose arguments are literals, whose second argument is a literal, and which is true in the usual interpretation of the function symbol. If P is a program, the set $C(P)$ of ***consequences of P*** is the closure of the axioms of L under substitutivity of equality and the rules of P. That is, $C(P)$ smallest set T of rules and equations containing P, as well as all axioms of L, and such that $\varphi(T) = T$ and $\pi(P,T) = T$.

6. Properties of L

If R = `rule`(*head*, *body*, *guard*) is a rule in language L, let $\rho(R)$ be the formula $(\forall x_1,\ldots,x_n)($ *guard* `->` `equals`(*head*, *body*)) of first order logic with equality, where $x_1,\ldots,x_n$ are the variables appearing in R. If P is a program in L, $\rho(P) = \{\rho(R) : R \in P\}$. We are now prepared to state

Theorem 1: Let P be an L program. If equation E is a consequence of P, then E is entailed under first order logic with equality[2] by $\rho(P)$, together with the axioms of L.

This result gives some of the formal verification properties of the language L, relating them to a simple, well known formal system. But there is a subtlety involved in the analogous informal reasoning. Informally we would like to write simply $6<9$ for, say `equals(less`(6,9),`true`), and, of course $a=b$ for `equals`(a,b). This means that the formal terms `equals`(a,b) and `equals(equals`(a,b), `true`) have the same informal translation $a=b$. The problem is that in some cases, the consequences of a program P may contain an equation of the form `equals`(a,b), without containing the corresponding term `equals(equals`(a,b), `true`). Therein lies the rub.

It turns out that this unfortunate occurrence happens precisely when one of the function symbols occurring in a or b is a symbol f for a partial function. That is, the guard conditions for the rules defining f are satisfied for arguments for which f cannot be inferred to have a literal value. For example, consider a program attempting to defining a factorial function by

```
rule(factorial(0), 1, true)

rule(
  factorial(n),
  times(n, factorial(minus(n,1))),
  not(equal(n,0))
)
```

In this case, `factorial` is a partial function since the guard condition of its second rule admits the argument -1, for example, while `factorial`(-1) cannot be inferred to have a literal value.

Here is where L departs most substantially from pure Lisp, but resembles traditional mathematics, in its methodology and its semantics. First, methodologically, we insist that partial functions never be written knowingly. These errors cannot be caught by a syntax checker, which would be equivalent to solving the halting problem; but they are to be considered harmful. Second, when evaluating terms, an L interpreter is not required to behave deterministically when evaluating partial functions outside of their domains. We next give the precise specification for an L-interpreter.

For any set S of rules and equations, define $\psi(S)$ to be the closure of S under the substitution law of equality for literals *and non-literals*-- precisely, $\psi(S)$ is the smallest set T of equations and rules containing S, such that if `equals`(a,b) or `equals`(b,a) appears in T, and X is an equation or rule appearing in T, then Y appears in T where Y is obtained from X by substituting one or more instances of a for b. The operator ψ differs from φ only in that substitution may be made for both literals and expressions (that is, nonliteral terms).

Now, if P is a program, define the set $Q(P)$ of ***quasi-consequences of P*** is the smallest set T of rules and equations containing P, as well as all axioms of L, and such that $\psi(T) = T$ and $\pi(P,T) = T$. Essentially, in inferring quasi-consequences of a program, we are allowed to make substitutions of arbitrary terms which have been inferred to be equal, without having inferred that those terms have literal values. From the standpoint of functional language theory, quasi-consequences of a Program are all results obtainable by arbitrary combinations of strict and lazy evaluation.

A program P is said to be ***denotational*** if (1) for every rule `rule`($h,b,$`true`) which is a consequence of P, there is a unique literal x such that `equals`(h,x) is a

[2] as expounded in, e.g., Cohen [2].

consequence of P, and (2) there is a mapping *rank* from terms to ordinals such that whenever `rule`(*head*, *body*, *guard*) is an instance of a rule appearing in P, and `equals`(*guard*, true) is a consequence of P, then every subterm of *body* has lower rank than *head*. It turns out that denotational programs are the formal analog of the informal mathematical notion of *well definedness*. We are now prepared to state

Theorem 2: If P is a denotational L program, then the consequences of P are precisely the quasi-consequences of P.

and,

Theorem 3: Let P be a denotational L program. Then equation E is a consequence of P if and only if E is entailed under first order logic with equality by $\rho(P)$, together with the axioms of L. Moreover, `equals`(a,b) is a consequence of P if and only if `equals`(`equals`(a,b), `true`) is.

We now define an ***L interpreter*** to be an algorithm which, given a denotational program P and a term T such that `equals`(T,x) is a consequence of P for some literal x, halts and returns x. Unlike Lisp and Haskell, we do not specify the results of the interpreter on non-denotational programs. Because the consequences and quasi-consequences of denotational programs are identical, we may use arbitrary inferences of predicate calculus to evaluate and analyze such programs; and we may do so informally, counting on the equivalence of `equals`(a,b) with `equals`(`equals`(a,b), `true`).

7. Conclusions

The semantics of L may be described, from a functional programming standpoint, as nondeterministically strict or nonstrict. The downside is that non-denotational programs may behave differently on different platforms. For many purposes, however, this is not a big worry. Nondeterminism results from a non-denotational program, which is considered a programming error. The error results either from a misunderstanding of the solution, or from a typo; and there is no reason to suppose that the misunderstanding, or the typo, would manifest itself more benignly in a language with strict semantics, such as Lisp,. It may, for example, result in a Lisp runtime error or non-termination, where in L it produces a misunderstood, but non-crashing, result in finite time. Another apparent downside is that L's supposed verification logic cannot be used to reason about programs which are not known to be denotational. It must be realized, however, that this is *in effect* the case in reasoning about programs in strict languages as well, such as Lisp.

An upside is that the L interpreter is free to deploy optimizations at runtime which do not adhere entirely to strict semantics, or entirely to lazy semantics. Like a human performing mathematical computation, but *unlike* a Lisp or Haskell interpreter, the L interpreter may perform arbitrary, *prima facia* valid operations and term substitutions in searching ways to optimize a calculation.

We submit that L fills what is currently a gap in programming language approaches, suiting application areas where programmers intend to reason seriously and precisely, but informally, about the properties of programs.

8. References

[1] Chwistek, Leon (1940). "A formal proof of Gödel's Theorem.", *The Journal of Symbolic Logic*, Vol. 5, (Mar., 1940), pp. 28-30

[2] Cohen, P. (1966). *Set Theory and the Continuum Hypothesis.* Addison-Wesley; 4th edition (August 1966)

[3] Cooke, D.E, Rushton JN, Watson R (2006) : The Evolutionary Role of Variable Assignment and Its Impact on Program Verification. In the proceedings of SEKE 2006 315-320

[4] Dijkstra (1989) "On the Cruelty of really teaching computer science". *Communications of the ACM*, December 1989.

[5] Hughs, John (1984). "Why functional programming matters." *The computer Journal.* Volume 32.

[6] Kieburtz, R.B (2002) "P-logic: property verification for Haskell Programs. Available at. citeseer.ist.psu. edu/kieburtz02plogic.html

[7] McCarthy (1960) Recursive functions of symbolic expressions and their computation by machine. *Communications of the ACM*, 7:184-195.

[8] Thompson, Simon (1989). "A logic for Miranda". *Formal Aspects of Computing* 1, 1989.

Plenary Talk: Towards Seamless Business Process and Dialogue Specification

Dirk Draheim

Software Competence Center Hagenberg
Softwarepark 21, 4232 Hagenberg, Austria
E-mail: draheim@acm.org

Executable specification or automatic programming [12] has always been a major strand of research to improve development and maintenance of Software systems. In the domain of enterprise applications the issue of executable specification is currently addressed by business process execution initiatives. We have seen steady efforts to make business process specifications executable, both in academia [5] and industry [10]. There are two non-mutual and converging research areas that foster this trend, i.e., business process modeling, e.g., [11], and workflow management [7].

Business processes are an issue in enterprises, e.g., [6], even without executable semantics of processes. And therefore, the subject of investigation in the business process modeling community is business process excellence, i.e., understanding best business processes, rather than exact semantics. Business process modeling languages are usually no more than visualizations of spoken language with all its ambiguities. Using a business process modeling language does not guarantee at all an improved exactness of the system description. However, it clearly eases communication between stakeholders and therefore fosters requirement elicitation substantially. Paradoxically, the vague semantics of typical business process modeling languages might be a reason for that. However, the gap remains; there is no canonical mapping between the components that are under the control of workflow technology and the entities addressed by business process modeling. Furthermore, the view of business process modeling is rather a global one, i.e., the net of business activities and exchanged information entities. The view of workflow control is also a local one, looking at the human computer interaction and having a concrete work list paradigm at hand for processing workflows.

Workflow control has its origins in concrete technologies for computer-supported collaborative work based on document processing [2] like Palo Alto's OfficeTalk [4] or Polymer [9] on the one hand and in more general rapid development frameworks based on a work list paradigm like FlowMark [8] on the other hand. A lot of today's commercial so-called BPM (Business Process Management) suites [10] actually started as workflow management products. However, we expect more from business process orientation than rapid development and better system maintainability. Support for advanced techniques like business process monitoring and business process simulation is desired.

Current BPM and workflow technologies are not fully integrated with the application programs that make up the dialogues of an enterprise application. This means that BPM technology controls the workflow states and not the dialogues that bridge the workflow states. The dialogue states are not seen by BPM technology. This means, most importantly, that the dialogues are not amenable to advanced BPM techniques like business process monitoring and business process simulation. Today, BPM technology is successfully used in enterprise application projects in the following sense: some rules in the interplay of existing enterprise applications are identified and these rules are then automized by a BPM product. On the other hand, if a workflow-intensive system should be built with BPM technology from scratch it is not obvious any more how to design the human computer interaction. The problem is to fix the right granularity of workflow versus dialogue states. Unfortunately, despite some heuristics [13] it still lacks a systematic treatment of this question. We follow a different, fundamental approach: we want to unify workflow states and dialogue states so that the aforementioned problem simply does not appear any more. An immediate major benefit of this is that advanced BPM techniques are no longer artificially restricted to some coarse-grained workflow states, they become pervasive. Furthermore, the business logic is partitioned naturally into services of appropriate granularity this way. The decision which parts of the supported business process is subject to workflow technology and which parts make the dialogues is orthogonal to the specification of the business process. The definition can be changed allowing for more flexibility in business process specification.

In order to unify workflows and dialogues it is necessary to investigate advanced role-model concepts from a workflow patterns perspective. There has been a rigorous discussion of workflow patterns in the workflow community [14] that helped in the investigation and analytical comparison of existing workflow technology. We will broaden the workflow pattern discussion by the consideration of different users and roles, because user and role models are at the heart of the workflow paradigm. This way, a human computer interaction viewpoint is brought to the discussion of workflow patterns that refines the current global, i.e., observational viewpoint of an overall action flow. In these efforts the findings of form-oriented analysis [3] serve as a basis. Here, the single user session of a submit/response-style system is defined as typed, bipartite state machine. The human-computer interaction is form-oriented - it consists of an ongoing interchange of report presentations and form submissions. We identified the notion of work list as an interaction pattern in single user session scenarios [1] and want to generalize the defined semantic apparatus to a form-oriented workflow definition language.

References

[1] Sandrine Balbo, Dirk Draheim, Christof Lutteroth, and Gerald Weber. Appropriateness of User Interfaces to Tasks. In (Alan Dix, Anke Dittmar, Eds.): Proceedings of TAMODIA 2005 - 4th International Workshop on Task Models and Diagrams for User Interface Design – For Work and Beyond, ACM Press, 2005.

[2] Dines Bjørner. Documents: A Domain Analysis. Retirement Lecture at The Technical University of Denmark, available at: http://www2.imm.dtu.dk/ db/, 27th March 2007, pp. 1-23.

[3] Dirk Draheim, Gerald Weber. Form-Oriented Analysis - A New Methodology to Model Form-Based Applications. Springer, 2005.

[4] Clarence A. Ellis and Marc Bernal. Officetalk-D: An Experimental Office Information System. ACM SIGOA Newsletter, vol. 3, no. 1-2, June 1982.

[5] Diimitrios Georgakopoulos, Mark Hornick, Amit Sheth. An Overview of Workflow Management. Distributed and Parallel Databases, 3, pp. 119-153, 1995.

[6] Michael Hammer and James Champy. Reengineering the Corporation: A Manifesto for Business Revolution. Harper Business Essentials, 2004.

[7] David Hollingworth. The Workflow Reference Model. Technical Report TC00-1003, Workflow Management Coalition, Lighthouse Point, Florida, USA, 1995.

[8] Frank Leymann and Dieter Roller. Business Process Management with FlowMark. Proceedings of IEEE Compcon, March 1994.

[9] Dirk E. Mahling, Noel Craven and W. Bruce Croft. From Office Automation to Intelligent Workflow Systems. IEEE Intelligent Systems 19(3), 41-47, 1995.

[10] Derek Miers, Paul Harmon, Curt Hall. The 2006 BPM Suites Report. Business Process Trends, 2006.

[11] Object Management Group. Business Process Modeling Notation Specification. OMG Final Adopted Specification, dtc/06-02-01, Object Management Group, February 2006.

[12] David L. Parnas. Software Aspects of Strategic Defense Systems. Software Engineering Notes, ACM Sigsoft, vol. 10, no. 5, ACM Press, October 1985.

[13] Bob Stegmaier, Mike Ebbers, Tomislav Begovac. Image and Workflow Library: FlowMark V2.3 Design Guidelines. IBM International Technical Support Organization, 1998.

[14] W.M.P. van der Aalst, A.H.M. Hofstede, B. Kiepuszewski, A.P. Barros. Workflow Patterns. Distributed and Parallel Databases 14: 5-51, 2003.

Evaluating the Efficiency of Retrieval Methods for Component Repositories

Oliver Hummel, Werner Janjic & Colin Atkinson
Chair of Software Technology, University of Mannheim
{hummel, wjanjic, atkinson}@informatik.uni-mannheim.de

Abstract

Component-based software reuse has long been seen as a means of improving the efficiency of software development projects and the resulting quality of software systems. However, in practice it has proven difficult to set up and maintain viable software repositories and provide effective mechanisms for retrieving components and services from them. Although the literature contains a comprehensive collection of retrieval methods, to date there have been few evaluations of their relative efficiency. Moreover, those that are available only study small repositories of about a few hundred components. Since today's internet-based repositories are many orders of magnitude larger they require much higher search precision to deliver usable results. In this paper we present an evaluation of well known component retrieval techniques in the context of modern component repositories available on the World Wide Web.

1. Introduction

The basic motivation for software reuse is simple. It is about "*creating software systems from existing software rather than building software systems from scratch"* as Krueger [1] put it in 1992. It is expected to reduce the time needed to developed software applications and improve the quality of delivered software products. Krueger's definition includes the potential reuse of any kind of asset during a software development process. Approaches like product line engineering [3] and design patterns [4] have proven particularly successful in this regard. However, the original vision presented by McIlroy [5] in 1968 focussed on the reuse of software components obtained from so-called component markets. The availability of components has clearly grown over the years but component markets have yet to make a significant impact on practical software development. This is for example demonstrated by the limited number of component vending sites available on the Internet and last year's surprising shutdown of the Universal Business Registry for web services [6].

Some successful "in-house" implementations of the component market concept have been reported by companies like IBM [9] or GTE [10] in the past. However, the size of their component repositories was limited to a few hundred components - the same approximate size as the component repository prototypes that were investigated in the 1990s (e.g. [7], [8]). Some experts like Poulin [19] have identified a size of about 200 components as the upper limit for manually maintained component repositories since the content of larger repositories tends to degenerate (the UBR is a good example for this hypothesis). However, researchers have long tried to exceed this limit and automatically populated repositories [20] with more than one hundred thousand assets have been reported more recently [2]. One rationale for this approach is to try to improve the chances of finding a suitable component. Another is the enormous size of publicly available class and function libraries, which often exceed thousands of components, and company version control systems which often contain hundreds of thousands of components. It is obvious that only automatically populated repositories will be able to cope with the numbers of reusable assets available today. The reuse of open source software from the web presents even more challenges to the reuse community and there has been general pessimism in some quarters that the so-called component storage and retrieval problem will be solved in the near future [13] [16].

Very recently however, some commercial search engines focusing on (open) source code from the Internet have demonstrated that repositories with millions of components are now technically feasible. The four major players in this segment are, in the order of appearance on the market: Koders.com, Krugle.com, merobase.com and Google's new code search engine (google.com/ codesearch). We believe that repositories of this size require a shift of priorities in component retrieval research. Where the main problem used to be to find any kind of matching (or at least a similar) component in a small repository, today the problem has shifted to finding the "best" matching component from a variety of candidates. This can only be achieved by improved retrieval techniques. Because of the previous lack of real world reuse repositories there are only a few publications examining the efficiency of these techniques. Even the authors of an often cited survey [16] admitted that their data about retrieval efficiency was largely estimated from common sense and the few meager studies accessible to them at the time. Even worse, the few viable results that are obtainable (e.g. [7]) are all based on repositories with far less than one thousand components. It is questionable whether these results will be scaleable for larger collections and will hold for repositories with millions of assets. The following small example illustrates why.

Consider the interface of a simple stack component that might have the following form in Java:

```
public class Stack {
  public void push(Object o) {}
  public Object pop() {}
  public int size() {}
}
```

Using signature matching [15], a classic and well known retrieval technique the component could be represented as follows:

```
Object -> void
void -> Object
void -> int
```

In a collection of about 1000 components such a signature might appear less then ten times and a stack could be identified with relative ease amongst them. However, performing the same query with the data pool of the merobase search engine (containing almost 10 million components) delivers more than 40.000 results.

Given these new developments it has become apparent that new analyses of the effectiveness of classical retrieval techniques, and combinations thereof, are required to improve the theoretical foundation for component repositories. In section 2 we provide a more detailed overview of the state of the art in component-based software reuse, component retrieval techniques and known evaluations. In section 3 we present our own evaluation of retrieval techniques based on modern code search engines from the web and discuss the results in section 4. Finally, we summarize and conclude our contribution in section 5.

2. Component Retrieval Background

Many attempts have been made to solve the problem of effectively storing large numbers of potential reuse candidates in a component repository. However, not only has this so-called "repository problem" [13] proven hard to solve, the question of how a component should best be represented (called the "reuse representation problem" by [7]) has also been a stumbling block. Given that the existing component repositories on the web are almost all source code centric and offer only basic text-search capabilities (see next subsection) it is difficult to use them for more sophisticated reuse approaches than just "code scavenging" [1]. This practice describes the copying and pasting of small code snippets into the system under development and is discouraged in many publications such as the Anti-pattern book [21]. It requires a lot of effort to find appropriate snippets and their use is more likely to degenerate the design of the system under development than to improve its quality. Although software reuse has been the subject of research for almost four decades, there is still no clear picture of when and how components should be used in a development process and how they should be stored in a repository. Even modern development methodologies contain only very abstract guidelines to select components based on their interface. Kratz et. al. [22] have recently shown that there is indeed some relation between the interface and the functionality of a component. However, since candidates might not match perfectly, a feedback loop may be necessary in which either the design or the candidate have to be adapted. This idea is best described in [11] so far.

Our own experience with reuse repositories indicates that the best kind of component search to use in a development process depends on the point of time at which the search is performed rather than on the nature of the process itself. The earlier reusable components are searched during a system's development the less design work is likely to have been carried out. Hence a general text-based search is more useful in early development phases and can feed back valuable information about potential components and their interfaces into the design process. On the other hand, if component search is carried out at a relatively late point in the development process an interface-driven search approach is required. Furthermore, if binary components or web services are to be discovered there is no source code and thus the search has to use interface descriptions in any case. Considering these differing requirements, a component search engine must be very flexible. However, most of the first generation search engines available today are only able to support keyword-driven searches.

2.1. Component Representation Methods

A repository's component representation format determines the possibilities for searches on this structure. Frakes and Pole [7] identified four basic representation methods briefly explained in the following. *Enumerated classification* originates from library science and separates an area into mutually exclusive, typically hierarchical classes. *Ontologies* in the semantic web community might be considered a modern synonym for this approach. *Facetted classification* [10] and the slightly more general *attribute value classification* approaches are very similar and use a number of facets (resp. attributes) to describe an asset. Each facet comprises a finite set of terms from which one is chosen to describe the asset. In contrast an attribute can contain any arbitrary value. Finally, free text indexing approaches extract textual information from an asset, i.e. the component or its documentation in our context.

There have been a lot of attempts to develop efficient component retrieval techniques for all four approaches. These are best summarized in the well-known survey by Mili et al. [16], but as the authors conclude, *"most* *solutions are either too inaccurate to be useful or too intractable to be usable"*. Since the representation methods are rather intuitive we turn our attention towards component retrieval techniques that more directly influence the query formulation techniques available to users in the next subsection.

2.2. Component Retrieval Techniques

Mili et al [16] distinguish the following fundamental techniques for component retrieval in five (originally six) not fully orthogonal groups:

1. Information retrieval methods
2. Descriptive methods
3. Operational semantics methods
4. Denotational semantics methods
5. Structural methods

Since software retrieval is grounded on information retrieval, a natural first step was to transfer the methods of the latter to the former and to apply a simple textual analysis to software assets. Descriptive methods go one step further and rely on additional textual descriptions of assets like a set of keywords or facet definitions [10]. Operational semantics methods rely on the execution or so-called sampling [8] of the assets. Denotational semantics methods use signatures [15] or specifications of components for retrieval. Finally, structural methods do not deal with functional properties of components but with their structure (i.e. act-alike vs. look-alike). Most of these mechanisms were initially developed for functional languages with an underlying type theory (i.e. type hierarchies) and no implicit variable passing as is common in today's programming languages. Only a few of these ideas (such as (1) and (2)) are easily transferable to object-oriented languages like Java or C#.
Mili et al. describe a sixth group – the so-called topological methods – as an approach to minimize the "distance" between query and reusable candidates. This approach relies on an underlying, "measurable" retrieval technique and hence we prefer to consider it as an approach for ranking the results of a query. The authors assessed these groups according to a scheme with five discrete rates ranging from very low (VL), low (L) through medium (M) to high (H) and very high (VH). Detailed explanations may be found in the referenced source itself. Due to space constraints, we only reproduce a table containing the three technical aspects that are interesting for the remainder of this paper. *Precision* is a measure from information retrieval (IR) theory that describes the ratio of relevant retrieved assets to the total number retrieved, see e.g. [14]. The *recall* also originates from IR and is the ratio of retrieved relevant assets to the total number of relevant assets in the collection. The meaning of the *automation potential* should be obvious.

Method	Precision	Recall	Automation Potential
1. Information Retrieval	M	H	H
2. Descriptive	H	H	VH
3. Operational Semantics	VH	H	VH
4. Denotational Semantics	VH	H	M
5. Structural	VH	VH	VH

Table 1. Overview of retrieval techniques.

It is important to note that Mili et al. themselves state that due to the low number of publications on this topic their values are largely best effort estimations. Moreover, since there were no large component repositories at that time the data is only based on experience with smaller collections of a few hundred components.

2.2.1. Previous Results

Information Retrieval (IR) typically uses recall and precision as defined previously to evaluate the performance of retrieval systems. Since it is important to know the number of relevant documents in a collection, the IR community has created a number of so-called reference collections (again with about 1000 documents, [20]) over the years. These collections are built by experts and hence it is known which documents can be considered relevant for a given query. Consequently, it is simple to test retrieval algorithms and to calculate recall and precision for them. One of the first authors who investigated the efficiency of their retrieval technique (called "behaviour sampling") in that way were Podgurski and Pierce [8]. They used a small library (around 100 components) of C functions where examples could be retrieved by randomized sampling, i.e. input and output values were provided and functions that delivered the expected outputs for given inputs were considered acceptable. The system delivered a precision of 100 percent if exactly the right number of samples (>= 12) was provided. However, it is clear that this technique is too time consuming for repositories with millions of assets. Frakes and Pole performed an evaluation of retrieval efficiency with UNIX commands [7]. Although it is one of the few publications that focused on this topic it is very domain specific and UNIX commands do not have an interface in the sense of components in modern programming languages. Hence, although these experiments can provide some general insights it is questionable whether they can be generalized and applied to today's retrieval systems. Inoue et al. presented and evaluated a retrieval system [2] that was considerably larger (about 120.000 components) than all previous systems. The authors realized that the classic recall measure cannot be calculated for repositories of that size since it is not possible to find all components potentially relevant for a query. Their examination focused on keyword-based searches (e.g. "clock applet") for Java but unfortunately the authors did not make their relevance criterion explicit. However, they claimed precision rates of about 70 percent for their system. One possible approach to estimate the recall for a large repository is injecting known components into it. However this is not feasible for third party search engines on the web. Moreover, results can easily become biased as they require examples that are known to work well with a given configuration.

To briefly summarize the main issues arising in this context: at present, no reference collections of software components are available. Moreover, even if one were available it could not come anywhere near the size and complexity of today's software repositories. It is therefore questionable whether its results would be scaleable to real world situations. Without the knowledge of how many components are relevant for a query it is not possible to calculate the recall of a search engine. Fortunately, it is feasible to calculate the precision by examining a given number of results and determining whether they do what they are supposed to do. We adopted these insights in our experimental design that we describe in more detail in section 3.

2.3. Component Search Engines

Most of the component search engines available today only offer keyword-based search capabilities based on a free-text representation of components (i.e. they use an IR method). The following table lists the most prominent component search engines that we were aware of at the time of writing.

Name	Languages	Components	Representation Methods Used	Support for Interface-Driven Searches
merobase	45	~10 M	all	yes
Google Codesearch (GCS)	44	~5 M	text, facetted	partially via regular expr.
Krugle	37	~5 M	text, facetted	limited, name-based
Koders	32	~1M	text, facetted	limited, name-based

Table 2. Popular code search engines.

To date, only merobase is able to fully support searches on interface descriptions based on a combination of IR- and denotational methods. Koders and Krugle are able to constrain searches to method or class names (we call this name-based) and interface-driven searches can be widely imitated with regular expressions on Google Codesearch. The large number of programming languages supported by the engines in table 2 can be explained by the fact that they not only support usual programming languages such as Java and C#, but also index scripting languages like Javascript or PHP and other artifacts such as makefiles etc. We do not consider other search engines such as Codase.com, Planetsourcecode.com, ucodit.com, jsourcery.com, Codehound.com etc. due to their limited size, range of languages or different search focus. As we have shown in [12], it is also possible to use regular Google or Yahoo for source code searches.

3. Experimental Evaluation

Due to the limitations discussed in section 2.2.1 we focused our investigation on the precision of search engines and retrieval techniques. We reused some of our previous work [12] where we collected query examples from the reuse literature. We derived interface-driven queries from them and inspected the first 25 results for Java (as it is most widely supported) from each query in two different experiments. First we evaluated the retrieval performance of various search. The results are presented in section 3.1. Our second experiment performed a more academic comparison of some retrieval techniques and is discussed in section 3.2.

Our matching criterion was that the required interface was contained verbatim or with simply a change of case in a candidate component. In order to finally decide whether such a candidate component is functionally appropriate and thus relevant we improved the sampling approach of [8] and manually defined meaningful JUnit test cases for each interface. We have already experimented with this technique before and found that interfaces and test cases can be used to describe and retrieve components in a very precise manner. Due to its affiliation with test-driven development we have called this approach "Extreme Harvesting". A proof of concept is also presented in [12].

3.1. Comparison of Search Engines

We limited our comparison to the three component search engines shown in table 3 below since only they offered an API for programmatic access at the time of writing. Additionally, we compared them with the general web search versions of Google and Yahoo which we enhanced with special filetype constraints to better utilize them for software component retrieval. To our knowledge, we made the optimal use of each search engines facilities for mimicking interface-driven retrieval. For instance, requiring the term "randomString" to appear only in method names should deliver more precise results with Koders than allowing it anywhere in the source code. Table 3 below summarizes our results for this experiment.

Query	Google	Yahoo	GCS	Koders	merobase
`copyFile(String, String): void`	1 / 25	2 / 25	7 / 25	0 / 25	18 / 25
`gcd(int,int):int`	10 / 25	7 / 25	12 / 25	2 / 25	17 / 25
`isLeapYear(int): boolean`	8 / 25	12 / 25	3 / 25	2 / 25	14 / 25
`md5(String):String`	0 / 25	0 / 25	4 / 22	0 / 25	12 / 25
`isPrime(int): boolean`	6 / 25	15 / 25	7 / 25	4 / 25	5 / 25
`randomNumber(int, int):int`	0 / 25	3 / 25	2 / 7	0 / 7	14 / 25
`randomString(int): String`	4 / 25	2 / 25	6 / 25	4 / 16	5 / 25
`replace(String, String, String): String`	2 / 25	8 / 25	14 / 25	3 / 25	22 / 25
`reverseArray( int[]):int[]`	1 / 10	3 / 23	1 / 1	0 / 4	5 / 7
`sort(int[]):int[]`	0 / 25	0 / 25	5 / 20	0 / 25	20 / 25
`sqrt(double): double`	5 / 25	4 / 25	4 / 25	1 / 25	11 / 25
`getMinMax(int[]): int[]`	0 / 15	0 / 22	0 / 0	0 / 25	2 / 4
`Stack( push(Object):void pop():Object size():int )`	1 / 25	2 / 25	0 / 0	1 / 25	6 / 25
Average Precision	12.2%	17.9%	29.5%	5.9%	53.7%
Standard Deviation	13.3%	18.9%	26.5%	7.8%	22.4%

Table 3. Comparison of code search engines.

We calculated the mean value and the standard deviation of each engine's precision. Furthermore, we performed t-tests for $\alpha = 0.05$ to measure the statistical difference of the results. Only the results provided by merobase show a significant improvement over the other engines. Google Codesearch (GCS) is also significantly better than Koders; but all other pairwise comparisons reveal no statistically significant difference. It is interesting that the general versions of Google and Yahoo tend to deliver more precise results for code searches than the specialized engine of Koders. However, we believe this can be explained by the different expressiveness of the queries offered by the different search engines. We will support this with more evidence in the next subsection where we directly compare retrieval methods.

3.2. Comparison of Retrieval Techniques

This subsection presents the results of our experiments to compare four retrieval techniques on the component pool of merobase. The experimental process is identical to that used in the last subsection. Since we had access to the data pool of merobase and could implement some dedicated support for these experiments we used this engine for a comparison of the retrieval techniques shown in table 4. However, it would not have been possible to test other known retrieval techniques or the representation models on their own without major changes to index structure. Hence we compared interface-driven search capabilities with pure signature matching and simple keyword-based searches in two distinct forms. Namely, an algorithm that searches keywords in the complete source code of components and a name-based algorithm that is able to constrain searches to method or class names (cf. table 2).

Query	signature matching	text-based	name-based	interface-driven
`copyFile(String, String): void`	0 / 25	3 / 25	16 / 25	18 / 25
`gcd(int,int):int`	0 / 25	20 / 25	11 / 25	17 / 25
`isLeapYear(int): boolean`	0 / 25	9 / 25	7 / 25	14 / 25
`md5(String):String`	0 / 25	0 / 25	0 / 25	12 / 25
`isPrime(int): boolean`	0 / 25	4 / 25	5 / 25	5 / 25
`randomNumber(int, int):int`	0 / 25	0 / 25	0 / 25	14 / 25
`randomString(int): String`	0 / 25	4 / 25	6 / 25	5 / 25
`replace(String, String, String): String`	1 / 25	6 / 25	0 / 25	22 / 25
`reverseArray( int[]):int[]`	0 / 25	0 / 25	2 / 25	5 / 7
`sort(int[]):int[]`	1 / 25	0 / 25	0 / 25	20 / 25
`sqrt(double): double`	0 / 25	2 / 25	4 / 25	11 / 25
`getMinMax(int[]): int[]`	1 / 25	2 / 25	2 / 25	2 / 4
`Stack( push(Object):void pop():Object size():int )`	0 / 25	3 / 25	3 / 25	6 / 25
Average Precision	0.9%	16.3%	17.2%	53.7%
Standard Deviation	1.8%	21.9%	19.3%	22.4 %

Table 4. Comparison of retrieval techniques.

We again performed statistical t-tests for $\alpha = 0.05$ on these results and found all pairwise comparisons significantly different, except for text-based vs. name-based. The results demonstrate that interface-driven searches are far more precise than plain keyword-based queries. This might also explain why Koders tends to be weaker then the general versions of Google and Yahoo where interface-driven searches can be better "simulated" with quoted queries. Google Codesearch and merobase consequently deliver significantly better results when their capabilities for regular expressions or interface-driven searches are used. However the precision remains still roughly between 30 and 60 percent and given the fact that sometimes thousands of results are returned a further increase of the precision is still required. This can be obtained by the use of a final semantic validation as integrated in our Extreme Harvesting approach where we use standard JUnit tests to check the dynamic behavior of components.

Another important requirement for precise searches are so-called search constraints that allow queries to be enhanced with additional metadata (such as the programming language) as is common in most general web search engines today. Thus, we believe it is not only necessary to combine various of the retrieval techniques proposed in the literature to reach acceptable precision on today's component collections, but also to combine a number of representation methods.

4. Conclusion

In the last year or so there has been an explosion in the number of search engines focusing on the discovery of software. However, their search algorithms and degree of precision are generally too weak to deliver a valuable service. One retrieval technique alone is typically not sufficient to guarantee high precision searches on a large repository and hence it makes sense to develop a combination of various techniques as we proposed for Extreme Harvesting [12]. However, this idea has so far been largely based on our intuition since we had no adequate repository at hand to compare the retrieval techniques proposed in the past. The results presented in this paper demonstrate that interface-driven searches deliver significantly better results than simple keyword-based approaches. Furthermore they deliver better candidates that can be checked with a more time consuming retrieval technique such as behavior sampling or the more advanced Extreme Harvesting. Testing of the form advocated in Extreme Harvesting seems to be able to push the precision close to 100%. However, to make Extreme Harvesting practicable we still have to overcome a number of practical problems (such as security and performance concerns). We are currently working on this challenge and will elaborate on further experiments on another occasion.

5. References

[1] C.W. Krueger, "Software reuse", *ACM Computing Surveys*, Vol. 24, Iss. 2, 1992.

[2] K. Inoue, R. Yokomori, H. Fujiwara, T. Yamamoto, M. Matsushita, S. Kusumoto, "Ranking Significance of Software Components Based on Use Relations", *IEEE Trans. on Software Eng*., Vol. 31, No. 3, 2005.

[3] Clemens, P., Northrop, L.: *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2002.

[4] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[5] D. McIlroy, "Mass-Produced Software Components", *Report of a conference sponsored by the NATO Science Committee*, Garmisch, 1968.

[6] Microsoft's UBR shutdown FAQ, 2006, http://uddi.microsoft.com/about/FAQshutdown.htm

[7] W.B. Frakes, T.P. Pole, "An empirical study of representation methods for reusable software components", *IEEE Transactions on Software Engineering,* Vol. 20, Iss. 8, 1994.

[8] Podgurski, A., L. Pierce: "Retrieving Reusable Software by Sampling Behavior", *ACM Transactions on Software Engineering and Methodology*, Vol. 2, Iss. 3, 1993.

[9] M. Lenz, H. Schmid, P. Wolf, "Software reuse through building blocks", in W. Tracz (ed..): *Software Reuse: Emerging Technology*, Computer Society Press, 1987.

[10] R. Prieto-Diaz, "Implementing Faceted Classification for Software Reuse". *Communications of the ACM*, Vol. 34, Iss. 5, 1991.

[11] I. Crnkovic, M. Chaudron, S. Larsson, "Component-based Development Process and Component Lifecycle", *Proceedings of Int. Conf. on Software Engineering Advances*, 2006.

[12] O. Hummel, C. Atkinson, "Using the Web as a Reuse Repository", *Proceedings of the International Conference on Software Reuse*, Torino 2006.

[13] R. Seacord: "Software Engineering Component Repositories", *Proceedings of the Int. Workshop on Component-Based Software Engineering*, 1999.

[14] R. Baeza-Yates, B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.

[15] A.M. Zaremski, J.M. Wing: "Signature Matching: A Tool for Using Software Libraries", *ACM Transactions on Software Engineering and Methodology*, Vol. 4, No. 2, 1995.

[16] Mili, A., R. Mili and R. Mittermeir: "A Survey of Software Reuse Libraries", *Annals of Software Engineering,* Vol. 5, 1998.

[17] K. Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.

[18] Y. Ye., G. Fischer, "Reuse-Conducive Development Environments", *Journal of Automated Software Engineering*, Vol. 12, No. 2, 2005.

[19] J. Poulin, "Populating Software Repositories: Incentives and Domain-Specific Software", *Journal of Systems and Software,* Vol. 30, Iss. 3, 1995.

[20] Y.S. Maarek, D.M. Berry, G.E. Kaiser, "An information retrieval approach for automatically constructing software libraries", *IEEE Trans. on Software Engineering*, Vol. 17, Iss 8, 1991.

[21] W.J. Brown, R.C. Malveau, H.W. McCormick, T.J. Mowbray, AntiPatterns: refactoring software, architectures, and projects in crisis, Wiley, 1998.

[22] B. Kratz, R. Reussner, W.J. v.d. Heuvel, "Empirical Research on Similaritiy Mertrics for Software Component Interfaces", *Journal of Integrated Design and Process Science*, Vol. 8, Iss. 4, 2004.

Benchmarking the RDF(S) Interoperability of Ontology Tools

Raúl García-Castro
Asunción Gómez-Pérez
Ontology Engineering Group, Departamento de Inteligencia Artificial.
Facultad de Informática, Universidad Politécnica de Madrid, Spain
York Sure
Institut AIFB, Universität Karlsruhe (TH), 76128 Karlsruhe, Germany

***Abstract*—The number of ontology tools, such as ontology editors and repositories, is constantly rising. Ideally, one could use them all seamlessly and thus benefit from all the functionalities they offer. As shown in previous EON workshops, interoperability among different development tools is not straightforward since ontology editors rely on specific internal knowledge models which are translated into common formats such as RDF(S). This paper addresses the urgent need for interoperability by providing an exhaustive set of RDF(S) benchmarks and demonstrating in an extensive field study the state-of-the-art of interoperability among six ontology tools. From the field study we have compiled a comprehensive set of best practices which may serve as guidelines. *Tool developers* benefit from having guidelines to design their import and export functionalities and a concrete set of benchmarks against which they can evaluate their import and export functionalities. *Ontology engineers* benefit from our work by having an overview to which extend interoperability is ensured for combinations of specific tools.**

I. Introduction

Ontologies enable interoperability among heterogeneous applications. The development and deployment of ontologies and ontology-based applications follows methodological guidelines and is supported by ontology tools such as ontology editors and repositories. Ideally one could use all existing ontology tools seamlessly and thus benefit from all the functionalities they offer. As shown in previous workshops on Evaluation of Ontology-based Tools (EON), interoperability among different ontology tools is not straightforward. For instance, ontology editors usually rely on specific internal knowledge models which are translated *more or less* into common formats such as RDF(S)[1]. Finding out why interoperability fails is cumbersome and non-trivial as any assumption made for the translation within one tool may easily prevent successful interoperability with other tools.

This paper addresses the urgent need for interoperability. We provide an exhaustive set of RDF(S) benchmarks which have been developed as part of the EU IST Knowledge Web Network of Excellence[2]. The benchmarks were designed to support evaluation and improvement of ontology tool interoperability. In an extensive field study we explored the state-of-the-art in interoperability among six ontology tools. Three of the participating tools are ontology editors (KAON, Protégé, WebODE) and three are repositories (Corese, Jena, Sesame), thus tool support for ontology development as well as deployment is covered. The field study helped us gain a deep understanding of the import and export functionalities of ontology tools. Our findings may serve as guidelines for developing tools and are summarized in comprehensive best practices on interoperability.

Tool developers benefit by having guidelines to design their import and export functionalities and by having a concrete set of benchmarks against which they can (automatically) evaluate their import and export functionalities. *Ontology engineers* benefit from our work by having an overview to which extend interoperability is ensured for combining specific tools. We hope that future generations of ontology tools will offer smooth interoperability and thus fulfil the key promise of ontologies.

This paper is structured as follows: Section II presents the motivation behind *benchmarking* software rather than *evaluating* it and other evaluation initiatives that have taken place. Section III examines how the RDF(S) interoperability benchmarking was conducted and how the RDF(S) benchmarks were designed. Sections IV and V summarize the results of executing the export, import and interoperability benchmarks. Section VI provides the recommendations extracted from benchmarking for ontology and software developers, and for anybody interested in carrying out a benchmarking activity. Finally, Section VII presents the conclusions derived from this work and future lines of work.

II. Related Work

A. Evaluation vs. Benchmarking

Software *evaluation*, according to the ISO 14598 standard [1], is defined as *the systematic examination of the extent to which an entity is capable of fulfilling specified requirements*; considering software not just as a set of computer programs but as the procedures, documentation and data produced.

Software *benchmarking* is a continuous process for improving software products, services and processes by systematically evaluating and comparing them to those considered to be the best. This definition, adapted from the definition given by the business management community [2], is followed by

[1]http://www.w3.org/RDF/
[2]http://knowledgeweb.semanticweb.org/

some authors in the Software Engineering community while others consider benchmarking as a software evaluation method [3].

The reason for benchmarking software products instead of just evaluating them is to obtain several benefits that cannot be obtained from evaluations. A software evaluation shows the weaknesses of the software or its compliance to quality requirements. If several software products are involved in the evaluation, we also obtain a comparative analysis of these products and recommendations for users. When benchmarking several software products, besides all the benefits commented, we gain continuous improvement of the products, recommendations for developers on the practices used when developing these products and, from these practices, those that can be considered best practices.

B. Related Evaluations

We now briefly present two evaluation initiatives closely related to our work. The first is a benchmark suite for evaluating RDF(S) usage, and the second is a previous evaluation of the interoperability of ontology development tools.

The *RDF Test Cases* [4] were created by the W3C RDF Core Working Group. These tests check the correct usage of the tools that implement RDF knowledge bases and illustrate the resolution of different issues considered by the Working Group. The RDF Test Cases could also be used for evaluating RDF(S) importers but, while they provide examples for, and clarification of, the normative definition of the language, our approach aims for an exhaustive evaluation of RDF(S) importers. Another difference is that we distinguish between the benchmarks that depend on the RDF(S) knowledge model and those that depend on the RDF syntax used. Moreover, we take into consideration valid input ontologies only, whereas the RDF Test Cases take erroneous input ontologies and entailment benchmarks.

The *Second International Workshop on Evaluation of Ontology-based Tools (EON 2003)* had as main topic the evaluation of the interoperability of ontology-based tools [5]. The results of the workshop led to significant improvements in various well-known ontology editors. The main reasons for benchmarking the interoperability of ontology tools again are:

- Interoperability is a great problem in the Semantic Web which is still unsolved.
- The workshop experiments (and those of its successors) involved only few tools and focused on editors.
- Some experiments evaluated export functionalities, others import functionalities and only a few evaluated interoperability.
- No systematic evaluation on a technical level was performed because ontology tool developers were just asked to model and interchange a domain ontology and report about findings. Each experiment used different test strategies and interchange languages, and also different principles for modelling ontologies. Therefore, only specific comments and recommendations were made.

We learnt many lessons from the results of the initial EON experiments which enabled us to do a systematic evaluation on a technical level such as the one presented in this paper.

III. RDF(S) Interoperability Benchmarking

The RDF(S) interoperability benchmarking started in Knowledge Web as an effort to improve the interoperability of ontology tools and to provide comprehensive recommendations for industry on how to use these tools. The benchmarking was organized and carried out following a generic software benchmarking methodology developed in Knowledge Web [6]. We now present the main decisions and outcomes of instantiating such methodology.

The goals for benchmarking the interoperability of ontology tools are related to the benefits pursued through it, and these are: to *evaluate and improve* their interoperability, to acquire a *deep understanding* of the practices used to develop the importers and exporters of these tools and to extract from these practices those that can be considered the *best practices*, to produce *recommendations* for users, and to create *consensual processes* for evaluating their interoperability.

These goals involve different communities that are related to the ontology development tools, namely, the research and industrial communities, and tool developers.

Participation in the benchmarking was open to any organisation irrespective of being a Knowledge Web partner or not. To involve other organisations in the process, with the goal of having the best-in-class tools participating, several actions were taken:

- The benchmarking proposal, a document being used as a reference along the benchmarking, was published as a public web page[3], which includes all the relevant information about the benchmarking: motivation, goals, benefits and costs, tools and people involved, planning, related events, and a complete description of the experimentation and the benchmark suites.
- Research was performed on the existing ontology development tools, both freely available and commercial ones, which could export and import to and from RDF(S), and their developers were contacted. In the future, any tool capable of importing and exporting RDF may participate in the benchmarking or benefit from the benchmarks.
- The interoperability benchmarking was announced with a call for participation through the main mailing lists of the Semantic Web area and through lists specific to ontology development tools.

Six tools took part in the benchmarking, three of which are ontology development tools: KAON, Protégé (using its RDF backend), and WebODE; the other three are RDF repositories: Corese, Jena and Sesame. These six tools do not share a common knowledge model and benchmarking was not always performed by tool developers.

The experimentation over the tools aimed to obtain results for interoperability improvement. Therefore, other quality at-

[3]http://knowledgeweb.semanticweb.org/benchmarking_interoperability/

tributes such as performance, scalability, robustness, etc. were not considered. However, an approach for benchmarking the performance and scalability of ontology development tools can be found in [7].

The experimentation took into account the most common way of interchanging ontologies that ontology tools provide, that is, by exporting ontologies from a tool into an interchange language and then importing ontologies into the other tool, using RDF(S) as interchange language and serializing the ontologies into RDF/XML syntax. A future benchmarking activity inside Knowledge Web will cover the case of using OWL[4] as interchange language.

Interoperability of ontology tools using an interchange language depends on the capabilities of the tools to import and export ontologies from/to this language. Therefore, the experimentation included not only the evaluation of interoperability but also that of the RDF(S) import and export functionalities of the tools.

The evaluation criteria must describe in depth the import, export and interoperability capabilities of the tools, whereas the experiments to be performed in the benchmarking must provide data informing how the tools comply with these criteria. Therefore, to obtain detailed information about these capabilities, we need to know: the elements of the *internal knowledge model* of an ontology development tool that can be imported from RDF(S), exported to RDF(S), and interchanged with another tool using RDF(S) as interchange language; the *secondary effects* of importing, exporting and interchanging these components, such as insertion or loss of information; and the *subset of elements* of the internal knowledge models that these tools may use to interoperate correctly.

To obtain these experimentation data, we defined three benchmark suites for evaluating the import, export and interoperability capabilities of the tools [8], which are common for all the tools. As the quality of the benchmark suites to be used is essential for the results of the benchmarking, the first step was to agree on the definition of these suites. Then, we decided to perform the import and export experiments before the interoperability ones, as the results of the first influence those of the second.

The steps to follow for executing the three benchmark suites are similar, and these are: the definition of the expected ontology resulting from importing, exporting or interchanging the ontology described in the benchmark; the import, export, or interchange of the ontology defined in the benchmark; and the comparison of the expected ontology with the ontology imported, exported or interchanged, checking whether there is some addition or loss of information.

The benchmark suites were intended to be executed manually but, as they contain many benchmarks, it is highly recommended to execute them (or part of them) automatically. In the cases of Corese, Jena, Sesame, and WebODE, most of the experimentation was automated. In the other cases, it was performed manually.

[4]http://www.w3.org/TR/owl-features/

The benchmarking web page[3] contains a detailed description of the benchmark suites, all the files to be used in the experiments, templates for collecting the results, and the experimentation results obtained.

A. Benchmark Suites

The benchmark suites check the correct import, export and interchange of ontologies that model a simple combination of ontology components (classes, properties, instances, etc.). Because one of the goals of the benchmarking is to improve the tools, the benchmark ontologies are kept simple on purpose so as to isolate problem causes and to identify problems.

As the ontology tools participating in the benchmarking have different internal knowledge models, both the experimentation and the analysis of the results are based on a common group of ontology modelling primitives, available in RDF(S) and in these tools. However, since tackling this common group exhaustively would yield a huge number of benchmarks, we have only considered the components most used for modelling ontologies in ontology development tools: classes, instances, properties with domain and range, literals, and class and property hierarchies. The rest of the components have not been dealt with so far.

1) The RDF(S) Import Benchmark Suite: contains 82 benchmarks, which define a simple RDF(S) ontology serialized in a RDF/XML file that must be loaded into the ontology development tool.

To isolate the factors influencing the correct import of an ontology, we have defined two types of import benchmarks: those that evaluate the import of the different combinations of components of the RDF(S) knowledge model, and those that evaluate the import of the different variants of the RDF/XML syntax, as stated in the RDF/XML specification.

2) The RDF(S) Export Benchmark Suite: comprises 66 benchmarks, which define an ontology that must be modelled in the tool and saved to a RDF(S) file.

We have defined two types of benchmarks for isolating the two factors influencing the correct export of an ontology. One group of benchmarks evaluates the correct export of the combinations of components of the ontology development tool knowledge model and the other group evaluates the export of ontologies with concepts and properties whose names include characters restricted by RDF(S), such as those not allowed for representing RDF(S) or XML URIs.

3) The RDF(S) Interoperability Benchmark Suite: evaluates the interchange of ontologies from one source tool to a destination one and vice versa. Each benchmark defines an ontology that must be modelled in the origin tool, saved to a RDF(S) file, and loaded into the destination tool.

Since the factors influencing the correct interchange of an ontology (besides the correct functioning of the importers and exporters) as well as the knowledge model used for defining the ontologies are the same as those in the RDF(S) Export Benchmark Suite, the ontologies defined in the RDF(S) Interoperability Benchmark Suite are identical to those of the RDF(S) Export Benchmark Suite.

The evaluation criteria are common for the three benchmark suites and are defined as follows:

- **Modelling** (*YES/NO*). The ontology tool can model the ontology components described in the benchmark.
- **Execution** (*OK/FAIL*). The execution of the benchmark is carried out without any problem, and the tool always produces the expected result. In the case of a failed execution, the following information is required: reasons for the failure of the benchmark execution and, if the tool had been corrected to pass a benchmark, the corrections performed.
- **Information added or lost.** The information added or lost in the ontology interchange.

In the export and interoperability benchmark suites, if a benchmark defines an ontology that cannot be modelled in a certain tool, this benchmark cannot be executed in the tool, being the *Execution* result *N.E.* (Non Executed). In the import benchmark suite, even if a tool cannot model some components of the ontology, it should be able to import correctly the rest of the components.

IV. Import and Export Results

The results obtained when importing from and exporting to RDF(S) depend mainly on the knowledge model of the tool that executed the benchmark suite. The tools that natively support the RDF(S) knowledge model (Corese, Jena and Sesame, essentially the RDF repositories) do not need to perform any translation in the ontologies when importing/exporting them from/to RDF(S). The RDF repositories import and export correctly from/to RDF(S) all the combinations of components, as these operations do not require any translation.

In the case of tools with non-RDF knowledge models (KAON, Protégé and WebODE, the ontology development tools), some of their knowledge model components can also be represented in RDF(S) whereas some others cannot; on the other hand, tools do need to translate ontologies between their knowledge models and RDF(S). Besides, not all the combinations of components of the RDF(S) knowledge model that have been taken into account in the benchmarking can be modelled into all the tools.

We present an analysis of the import and export results of the participating ontology development tools.

A. Import Results

In general, ontology development tools import correctly from RDF(S) all or most of the combinations of components that they model, rarely adding or losing information. The only exceptions are: Protégé, which poses problems only when importing classes or instances that are instances of multiple classes, and WebODE, which causes problems only when importing properties with a XML Schema datatype as range.

When the ontology development tools import ontologies with combinations of components that they cannot model, they lose the information about these components. Nevertheless, they usually try to represent partially these components using other components from their knowledge models. In most cases, the import is performed correctly. The only exceptions are: KAON, which poses problems when it imports class hierarchies with cycles, Protégé poses problems when it imports class and property hierarchies with cycles and properties with multiple domains, and WebODE, which causes problems when it imports properties with multiple domains or ranges.

When dealing with the different variants of the RDF/XML syntax, ontology development tools import correctly resources with different URI reference syntaxes and with different syntaxes (shortened and unshortened) of empty nodes, of multiple properties, of typed nodes, of string literals, and of blank nodes. The only exceptions are: KAON when imports resources with multiple properties in the unshortened syntax; and Protégé when imports resources with empty and blank nodes in the unshortened syntax. The tools do not import language identification attributes (*xml:lang*) in tags.

B. Export Results

In general, ontology development tools export correctly to RDF(S) all or nearly all of the combinations of components that they model without losing information, though KAON poses problems only when exporting to RDF(S) datatype properties without range and datatype properties with multiple domains and with an XML Schema datatype as range, whereas Protégé causes problems only when exporting to RDF(S) classes or instances that are instances of multiple classes and template slots with multiple domains.

When these tools export components that are present in their knowledge model but cannot be represented in RDF(S), such as their own datatypes, they usually insert new information in the ontology though some information is lost.

When dealing with concepts and properties whose names do not fulfil URI character restrictions, each ontology development tool behaves differently. When names do not start with a letter or "_", some tools leave the name unchanged while others replace the first character with "_", spaces in names are replaced by "-" or "_", depending on the tool, and URI reserved characters and XML delimiter characters are left unchanged, replaced by "_", or encoded.

V. Interoperability Results

The RDF repositories (Corese, Jena and Sesame) interoperate correctly among themselves as they always import and export from/to RDF(S) correctly. This causes that interoperability between the ontology development tools and the RDF repositories depends only on the capabilities of the former to import and export from/to RDF(S) and, therefore, the results about this interoperability are identical to those presented in the previous section.

The import and export results presented in the previous sections showed that few problems arise when importing and exporting ontologies. Nevertheless, the interoperability results present more problems.

As a general comment we can say that interoperability between the tools depends on: a) the correct working of their

RDF(S) importers and exporters; and b) the way selected for serializing the exported ontologies in the RDF/XML syntax.

Furthermore, we have observed that some problems in any of these factors affect the results of not just one but of several benchmarks. This means that, in some cases, correcting a single import or export problem or changing the way of serializing ontologies can cause significant interoperability improvements.

Next, we list the components that can be interchanged between the tools.

1) Interoperability using the same tool: Ontology development tools seem to have no problems when the source and the destination of an ontology interchange are the same tool. The only exception resides in Protégé when interchanges classes that are instances of multiple metaclasses and instances of multiple classes, since Protégé does not import resources that are instances of multiple metaclasses.

2) Interoperability between each pair of tools: Interoperability between different tools varies according to the tools. Besides, as the detailed interoperability results show, in some cases the tools are able to interchange certain components from one tool to another, but not the other way round.

When **KAON** interoperates with **Protégé**, both can interchange correctly some of the common components that they are able to model. But the problems arise with classes that are instances of a single metaclass or of multiple metaclasses, with datatype properties without domain or range, with datatype properties whose range is *String*, with instances of multiple classes, and with instances related via datatype properties.

When **KAON** interoperates with **WebODE**, they can interchange correctly almost all the common components that these tools can model. The only exception occurs when they interchange datatype properties with domain whose range is *String*.

When **Protégé** interoperates with **WebODE**, they can interchange correctly all the common components that these tools can model.

3) Interoperability between all the tools: Interoperability between **KAON**, **Protégé** and **WebODE** can be achieved with nearly all the common components that these tools can model. The only components that these tools cannot use are: datatype properties with domain and whose range is *String*, and instances related through datatype properties.

Therefore, interoperability can be achieved among the tools that have participated in the benchmarking using classes, class hierarchies without cycles, object properties with domain and with range, instances of a single class, and instances related through object properties.

4) Interoperability regarding URI character restrictions.: Interoperability is low when tools interchange ontologies containing URI character restrictions in class and property names. This is mainly due to the fact that tools usually encode some or all the characters that do not comply with these restrictions, which provokes changes in class and property names.

VI. Recommendations

A. Recommendations for ontology engineers

This section offers recommendations for ontology engineers which use more than one ontology tool to build ontologies. Depending on the tools used, the level of interoperability may be greater or lower, as can be seen in Section V.

If the ontology is being developed bearing in mind interoperability, the ontology engineers should be aware of the components that can be represented in the ontology development tools and in RDF(S). Also, they should try to use in their ontologies the common knowledge components of these tools in order to avoid the knowledge losses known already.

Ontology engineers should also be aware of the semantic equivalences and differences between the knowledge models of the tools and the interchange language.

It is not recommended to name resources using spaces or any character that is restricted in the RDF(S), URI or XML specifications.

In the case of interoperability in the RDF repositories, although these repositories export and import correctly to RDF(S), ontology engineers should consider the limitations that other tools have when exporting their ontologies to RDF(S) with the aim of interchanging them.

B. Recommendations for tool developers

This section includes general recommendations for improving the interoperability of the tools when developing them. In [9], we offer more detailed recommendations to improve each of the participating tools according to the results and practices found. Although it is not compulsory to follow these recommendations, they help correct interoperability problems as we could observe when analysing the results.

Interoperability between ontology tools using RDF(S) as interchange language depends on how the importers and exporters of these tools work, whereas the way they work depends on the development decisions made by tool developers, who are different people with different needs. Therefore, it is not straightforward to provide general recommendations for developers. Nevertheless, some comments can be extracted from the analysis of the benchmarking results.

Seldom, a development decision will produce an interoperability improvement with some tools but a loss with others. For example, when exporting classes that are instances of a metaclass, some tools require that the class be defined as instance of *rdfs:Class* while some others require the opposite. The collateral consequences of the development decisions should be analysed by the tool developers.

Tool developers should be aware of the semantic equivalences and differences between the knowledge models of their tool and the interchange language; on the other hand, the tools should notify the user when the semantics is changed.

The first requirement for achieving interoperability is that tool importers and exporters are robust and work correctly when dealing with unexpected inputs. Although this is an evident comment, the results show that this requirement is not fulfilled by the tools and that some tools even crash when importing some combinations of components.

Above all, tools should deal correctly with the combinations of components that can be present in the interchange language but that cannot be modelled in them. For example, cycles in class and property hierarchies cannot be modelled in ontology development tools. Nevertheless, these tools should be able to import these hierarchies by eliminating the cycles.

To export components commonly used by ontology development tools, they should be completely defined in the file. This means that metaclasses and classes in class hierarchies should be defined as instances of *rdfs:Class*, properties should be defined as instances of *rdf:Property*, etc.

Exporting complete definitions of other components can cause problems if these are imported by other tools. Not every tool deals with datatypes defined as instances of *rdfs:Datatype* in the file or with *rdf:datatype* attributes in properties.

Every exported resource should have a namespace if the document does not define a default namespace.

C. Recommendations for benchmarking

This section offers recommendations to perform benchmarking activities that were extracted from the lessons learnt while instantiating the methodology.

Benchmarking is not about comparing the results of the tools but the practices leading to these results. Therefore, experimentation should be designed to obtain these practices as well as the results.

Resources are needed mainly in three tasks: benchmarking organisation, experimentation definition and execution, and results analysis. It should be ensured that enough resources are allocated to each of these tasks.

Benchmarking is an activity that takes long time as it requires tasks that are not immediate: announcements, agreements, etc. Therefore, benchmarking activities should start early in time and the benchmarking planning should consider a realistic duration of the benchmarking.

Benchmarking needs the participation of relevant experts in the domain together with the best-in-class tools. Although it is not required that the tool developers participate in the benchmarking and perform the experiments over their tool, their involvement facilitates the execution and analysis of the experimentation results to a large extent. In all the cases where tool developers performed the experimentation on their own tools, they were able to detect problems and improve their tools while executing the benchmark suites.

VII. Conclusions and Future Work

Seamless interoperability among ontology tools greatly facilitates the development and deployment of ontologies. In this paper we present a set of concrete RDF(S) benchmarks and results and best practices obtained after applying them in an extensive field study on a number of well-known ontology tools.

Our benchmarks may be used by any ontology tool developer to improve import and export functionalities and to ensure interoperability with other tools on a very fine-grained level.

Our field study wants to show ontology engineers to which extent state-of-the-art tools are interoperable *right now*. However, ontology engineers may use the benchmarks themselves to benchmark relevant tools, e.g. in early ontology development project stages to facilitate the selection of appropriate tools.

During this benchmarking activity, tool developers sometimes automated the execution of the benchmark suites, but the experimentation was mainly done by hand. Carrying out experiments manually and analysing the results is expensive since both tasks depend on the expertise of the people performing them, and can be influenced by human errors. Therefore, these tasks should be automated as much as possible and mechanisms should be set up to detect human errors.

Future work mainly includes the development of means to automatize experimentation as much as possible and the development of appropriate OWL benchmark suites to be used in a future benchmarking activity that will consider interoperability using OWL as interchange language.

Acknowledgments

This work is partially supported by a FPI grant from the Spanish Ministry of Education (BES-2005-8024), by the IST project Knowledge Web (IST-2004-507482) and by the CICYT project Infraestructura tecnológica de servicios semánticos para la web semántica (TIN2004-02660). Thanks to all the people that have participated in the RDF(S) interoperability benchmarking: Olivier Corby, Jesús Prieto-González, Moritz Weiten, and Markus Zondler. Thanks to Rosario Plaza for reviewing the grammar of this paper.

References

[1] ISO/IEC, *ISO/IEC 14598-1: Software product evaluation - Part 1: General overview*, 1999.
[2] M. Spendolini, *The Benchmarking Book*. New York, NY: AMACOM, 1992.
[3] R. García-Castro, "Keynote: Towards the improvement of the Semantic Web technology," in *Proceedings of the Second International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2006)*, Athens GA, USA, November 2006.
[4] J. Grant and D. B. (eds.), "RDF test cases. W3C recommendation 10 february 2004," W3C, Tech. Rep., February 2004.
[5] Y. Sure and O. Corcho, Eds., *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools (EON2003)*, ser. CEUR-WS, vol. 87, Florida, USA, October 2003.
[6] R. García-Castro, D. Maynard, H. Wache, D. Foxvog, and R. González-Cabero, "D2.1.4 Specification of a methodology, general criteria and benchmark suites for benchmarking ontology tools," Knowledge Web, Tech. Rep., December 2004.
[7] R. García-Castro and A. Gómez-Pérez, "Guidelines for benchmarking the performance of ontology management APIs," in *Proceedings of the 4th International Semantic Web Conference (ISWC2005)*, ser. LNCS, no. 3729. Galway, Ireland: Springer-Verlag, November 2005, pp. 277–292.
[8] ——, "Benchmark suites for improving the RDF(S) importers and exporters of ontology development tools," in *Proceedings of the 3rd European Semantic Web Conference (ESWC2006)*, ser. LNCS 4011. Budva, Montenegro: Springer-Verlag, June 2006.
[9] R. García-Castro, Y. Sure, M. Zondler, O. Corby, J. Prieto-González, E. P. Bontas, L. Nixon, and M. Mochol, "D1.2.2.1.1 Benchmarking the interoperability of ontology development tools using RDF(S) as interchange language," Knowledge Web, Tech. Rep., June 2006.

A Deep Classification of Temporal Versioned Integrity Constraints for Designing Database Applications

Robson Leonardo Ferreira Cordeiro*, Renata de Matos Galante+†,
Nina Edelweiss+ and Clesio Saraiva dos Santos+

*Computer Science Department (ICMC) - University of São Paulo (USP) - São Carlos - Brazil
+Informatics Institute (II) - Federal University of Rio Grande do Sul (UFRGS) - Porto Alegre - Brazil
E-mail: (rlfcordeiro, galante, nina, clesio)@inf.ufrgs.br

Abstract

The version concept was initially proposed for controlling design evolution on computer aided design and software engineering. Time concept has been widely used in software engineering for real-time systems and the ones that need to register historical data together with their validity periods. Recent researches have proposed the use of both concepts in databases, in order to automatically control the storage and retrieval of historical data and project alternatives. However, integrity constrains on this kind of data still represent an almost unexplored research area. This paper proposes a classification for constraints considering the use of time and version concepts in the data and in the constraints themselves. Constraints are deeply analyzed in the paper, considering their origin, substance, specification, application, temporality and versioning aspects. The classification completeness is also analyzed and a case study is presented. The proposed classification is more expressive than others found in literature and, at the same time, simplifies the task of designing database applications that have to deal with historical and evolving data. This work serves as a base for new researches on the definition, specification and optimized maintenance of these constraints.

1. Introduction

The concept of version was proposed for controlling design evolution and co-authoring on computer aided design [11] and software engineering [17]. In those environments, versioning is applied to files, such that different alternatives or revisions of a document are stored on different files handled by the operating system. One of the most common tool for handling this type of versions is the CVS (Concurrent Version System). The time concept has been widely used in software engineering for real-time systems (specially for embedded systems) [3] and database applications that have to deal with historical data and their validity periods [16]. A glossary of temporal database concepts is found in [10].

Recent work have proposed the use of time and version concepts in database applications, named temporal databases with versions support [4, 5, 15]. They use the time concept to control and store historical data while the version concept allows managing several project alternatives and the evolution of different elements of the data.

Considering that any database application represent some part of the real world, it is possible to affirm that, in order to make a faithful representation of reality, the data managed by these applications must obey several integrity constraints provided by the modeled reality [5, 4].

There are many researches and implementations on the integrity maintenance for snapshot databases. These work usually classify constraints based on their common characteristics, in order to define different maintenance methods for constraints in each class, trying to take advantage of this fact. Recent work on constraints for XML (eXtensible Markup Language) and spatial snapshot databases have been made [12, 6]. However, constraints concerning time and versions still represent an almost unexplored research area that must be deeply analyzed [5, 4].

This paper proposes a detailed integrity constraints classification for database applications, considering the use of time and version concepts in data and in the constraints themselves[1]. The classification particularities are presented and its completion is analyzed by comparing it with some other classifications found in literature. A case study that illustrates its applicability is also provided.

The proposed classification is more expressive than others found in literature and, at the same time, simplifies the task of designing database applications that have to deal with historical and evolving data. By grouping constraints with common characteristics in classes, this classification is

*Work performed while studying at UFRGS.

†This work has been partially supported by Fapergs under grant No. 0412264, CNPq No. 550.845/2005-4 and CNPq No. 475.743/2004-0.

[1]The first ideas of this work were shown in [5].

important in several software engineering areas, since it can be seen as a main step for the specification and, specially, for the automatic maintenance of constraints that still must be validated by user applications or auditing trail tools.

The rest of this paper is organized as follows. The second and main Section presents the constraints classification. In Section three a case study that illustrates the classification use and applicability is presented. Section four presents related work analyzing the classification scope. Finally, Section five shows conclusions and ideas for future work.

2. Integrity Constraints Classification

This Section presents a classification of constraints for temporal databases with versions support. Constraints are deeply analyzed, based on their characteristics, in order to provide an important base for their specification and automatic maintenance, making it easier the design of database applications through software engineering processes.

A constraint specification must have the following components: (i) *restrictive*, a set of components, present or not in the database, used to build restrictions on the database data; (ii) *restricted*, a set of database components whose contents are restricted by the constraint; (iii) *restrictive condition*, a generic logic expression relating the restricted and restrictive components; (iv) *verification points*, that define the starting points of its validity verification, and (v) *violation actions*, possible actions executed over the database immediately after the constraint violation in order to establish again the data integrity. Thus, the data integrity maintenance related to a constraint is based on the validity verification of its restrictive condition on its verifying points.

Based on these components, constraints are analyzed by the aspects: *origin, substance, specification, application, temporality* and *versioning*. Each one of these aspects, described in the following Sections, has several criteria and sub-criteria in order to make possible all classes definition.

Due to the space limitation, aspects not related to time or versions (*origin, substance* and *application*) are briefly described in a unique Section. Only time and versions aspects are detailed, since they are more important to this paper.

2.1. Origin, Substance and Application

The *origin* aspect defines that the possible generating agents of integrity constraints are the *environment*, the *enterprise*, the *data model*, and the *implementation*.

The *substance* aspect allows anyone to classify constraints based on the kind of condition imposed on data. It has several criteria commented as follows. The components of the restrictive and restricted sets of a constraint are classified by their *kind*. This kind can be *schema, entity, relationship* or *domain*. Also the *scope* of these sets is considered to evaluate the database parts related to the constraint. Through the *aspect* criterion, constraints can restrict a database by its *data cardinality, internal composition, temporality* or *versioning*, besides *schema behavior* or *structure*. Also, by the *completion* criterion, a constraint can be *complete*, when it completely represents a constraint of the modeled reality, or *incomplete*, when it represents, in conjunction with others, a single real constraint. According to [1, 4], the *state scope* criterion classifies *static constraints*, when they use only one database state during their verification processes, or *dynamic constraints*, when they use more than one state. The last ones can also be *transition* or *non-transition* constraints. Diverging of other work [2, 13], here *static* constraints can be *temporal*, referring to a unique instant in the past, present or future time. *Dynamic* constraints are always *temporal*, because, even if time is not referenced explicitly, its concept is implicitly used when distinct database states are analyzed.

The *application* aspect takes into account the possible integrity maintenance forms related to the existent constraints. Its criteria are commented as follows. The *verification points* of a constraint can be *based on update operations* executed over the database or *based on the occurrence of events* related to *temporal* or *non-temporal* facts. The first ones are divided in *immediate* and *postponed*, respectively related to the execution of individual operations or complete ACID transactions. The *verification* of a constraint can be *ordered*, when it must respect some ordination, based on the verification of other(s), or *free*, on the contrary way. Constraints can also be *active* and demand data to respect it or, *inactive*, temporarily, not influencing the integrity maintenance. Moreover, the *activation and deactivation* of some constraints can be *dependent*, occurring obligatorily together, or *independent* [7]. The possible *treatments* received by constraints in order to maintain the data integrity are: (i) *prevention*, based on the execution of actions to prevent violations; (ii) *detection*, that only accuses the violations; (iii) *complementation*, executing violation actions after any violation, or (iv) *reconstitution*, rolling back the database to a valid state not influenced by the violation operation or transaction. Finally, the possible vehicles used to maintain the data integrity are: *operational system, database management system, application programs* and *auditing trail tools*.

2.2. Specification

The *specification* of a constraint is the process that incorporates it in the database schema. After that, the constraint must be respected in an unconditional way [4].

It is important to notice that this aspect classifies a constraint according to the temporality and versioning of the constraint itself, without considering any data characteristic. These ones are considered by the following aspects.

A constraint *declaration* can be: *implicit*, when it is inherent in the data model, *explicit*, when it is explicitly represented on the specification, or *derived*, when it is partially inherent in the data model and complemented by explicit statements. According to its *self-temporality scope*, a constraint can be *global*, when it must be respected during the whole database lifetime, or *local*, when it is considered only in some part of this lifetime. The *local* scope can also be related to moments in the *past, present* or *future*. Details are found in Figure 1a. The *self-temporality type* (Figure 1b) of a constraint can be: (i) *non-temporal*, for constraints classified as *global* in the last criterion; (ii) *temporal determination*, for the ones that use time referring to exact moments, or (iii) *temporal indetermination*, when they use time referring to moments in an inaccurate way. Constraints on these two last classes store their update history and may be valid only in specific periods. Also, a constraint can have a *non-versioned self-versioning type*, having no versioning control over itself, or *versioned*, on the contrary way (Figure 1c).

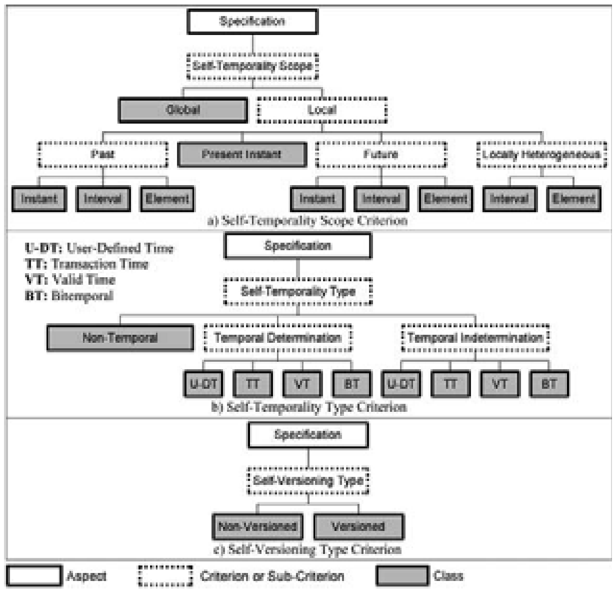


Figure 1. Part of the Specification aspect.

2.3. Temporality

The *temporality* aspect groups constraints based on the temporality of their restricted and restrictive sets. The temporality of the constraint itself is not considered here. Thus, a constraint can be *non-temporal*, enclosing non-temporal restrictive and restricted sets, or *temporal*, when these sets use the temporality in an implicit or explicit way. The latter are sub-classified by several sub-criteria. Temporal constraints depend on the existence of a *"clock"* whose flow is independent of any action executed on the database. Thus, when directly related to the time, their violations are irreversible due to the irreversibility of the flow of time [2].

References in restrictive conditions relate restricted sets to restrictive ones limiting the range of possible values on the first set, based on the second one. These *references* (Figure 2a) can use two distinct time notions: (i) *absolute time*, when there is a quantitative relation between both sets and, absolute values are defined for events duration or for their temporal distances, and (ii) *relative time*, defining a qualitative relation by the use of special operators like the following ones: *before, into* and *after* [7, 10]. The *temporal granularity* (Figure 2b) of constraints can be *homogeneous* or *heterogeneous*, according to the metric unit (*chronon*) used in their temporal references.

The sub-criteria *restrictive* and *restricted temporality types* are classified as the *self-temporality type* (Section 2.2), but considering only data characteristics. *Non-temporal* components can be schemas, entities or relationships, defined as non-temporal in the data modeling, or *domain kind* components (only for restrictive sets) that do not explicitly relate the database to the time. All other components are *temporal*. Figure 2c shows a detailed description of this criterion. Finally, in order to make possible the scope analysis of many database states during its whole lifetime, the sub-criteria *restrictive* and *restricted temporality scopes* (Figure 2d) are defined. They classify constraints as the *self-temporality scope* (Section 2.2), except for not being related to a constraint itself, but for its restrictive and restricted sets. Also, the restrictive one considers the *indefinite* temporal scope, for restrictive *domain kind* components not related to the time and, the possibility of a *local* temporal scope *independent on the current time*.

2.4. Versioning

Besides the self-versioning, considered in the *specification* aspect, restrictive and restricted sets can also make use of the version concept. Thus, the *versioning* aspect classifies *non-versioned* constraints, with restrictive and restricted sets that do not use this concept, and *versioned* ones, on the opposite way. *Versioned* constraints are classified by the sub-criteria described as follows.

Considering their *restrictive* and *restricted versioning types* (Figure 3a), *versioned* constraints have components defined as versioned on the data model or explicitly referring to the versioning of a database part. All other constraints are *non-versioned*. The *restrictive* and *restricted versioning scopes* (Figure 3b, divided, due to the space limitation) of a constraint are classified as *global*, enclosing all versions of database components, or *local*, enclosing only part of these versions. The *indefinite scope* is also possible for restrictive sets with components of the *domain kind*, not directly related to the database versioning. Also, the *local* scope is sub-classified considering the number and the states of the enclosed versions. These sub-criteria consider the scope of different versions of database components

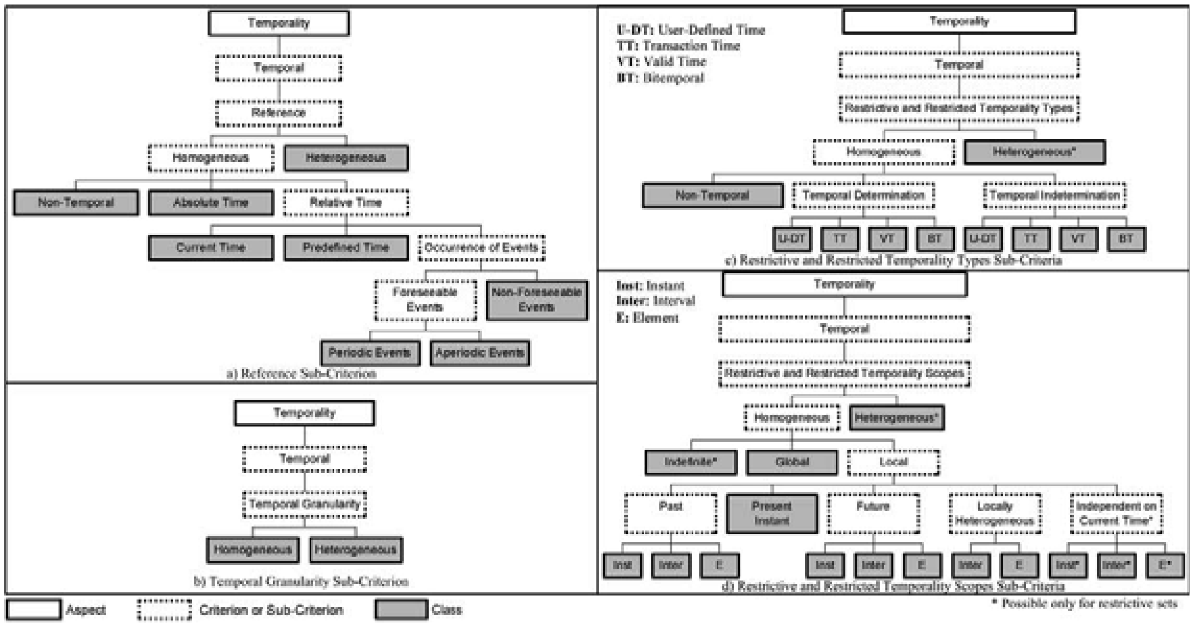


Figure 2. Temporality aspect.

making it possible, together with the other (*binding*) *scope* criteria described before, a complete scope classification of constraints for any temporal database with versions suport.

3. Case Study

The main goal of this Section is to present a case study in order to illustrate the classification use and applicability in temporal databases with versions support.

The system of a software production company is considered here. Actual constraints on this reality are specified in colloquial language and classified according to their time and versions characteristics. The Temporal Versions Model (TVM) [15] is used in the case study. TVM is an object oriented data model for temporal databases with versions support. Time, represented by transaction and valid times, can be related to objects, versions, attributes and relationships.

Figure 4 presents the UML class diagram used by the application example[2], based on the TVM model and on the minute granularity. In this diagram, the classes *analyst* and *programmer* inherit by extension (a special TVM relationship) all characteristics of the *employee* class. The salary of an employee and the base salary of his/her category are defined by temporal relationships with the *salary_level* class. All software projects must be coordinated by an analyst and managed by a workgroup, composed by analysts and programmers. Temporal relationships are used in order to register the historical updates on these relations.

Based on the presented reality, the following constraints are classified: (i) *the salary of an employee cannot be lower than the base salary of his/her category, after three months of employment*, and (ii) *the analyst that coordinates a project must participates in the workgroup that manages it*. The classifications for the *specification*, *temporality* and *versioning* aspects are shown in Figure 5, through trees whose leaf nodes (in bold) represent classes of constraints. Considering that the first constraint depends on legal aspects that can be easily changed in time (specially the *"three months"*), its *self-temporality* was defined as *bitemporal* (transaction and valid times) with *local scope*, in order to control its validity periods and change history. Also, considering the possibility of different experience periods for employees in distinct categories, the *self-verisoning type* of this constraint is *versioned*, in order to have different constraint versions related to the employee categories at the same time. The *self-temporality* of the other constraint is *non-temporal* with *global scope* and its *self-versioning* is *non-versioned*, since it is considered stable. Finally, the classifications for the temporality and versioning aspects are based on the components in the constraints restrictive and restricted sets, defined in the data modeli.

4. Related Work and Classification Scope Analysis

Work related to constraints for temporal databases with versions support are rare and the existing ones usually consider only one of the analyzed features: time or versions. An exception is the paper [4], were Cordeiro et al. propose an specification language (TVCL) for constraints considering both time and versions. However, a classification was not defined and, since it lacks a deep inspection on constraints

[2]In the diagram, the template TV represents temporal versioned classes. Temporal attributes and relationships are defined, respectively, by the labels $<< T >>$ and $<< Temporal >>$.

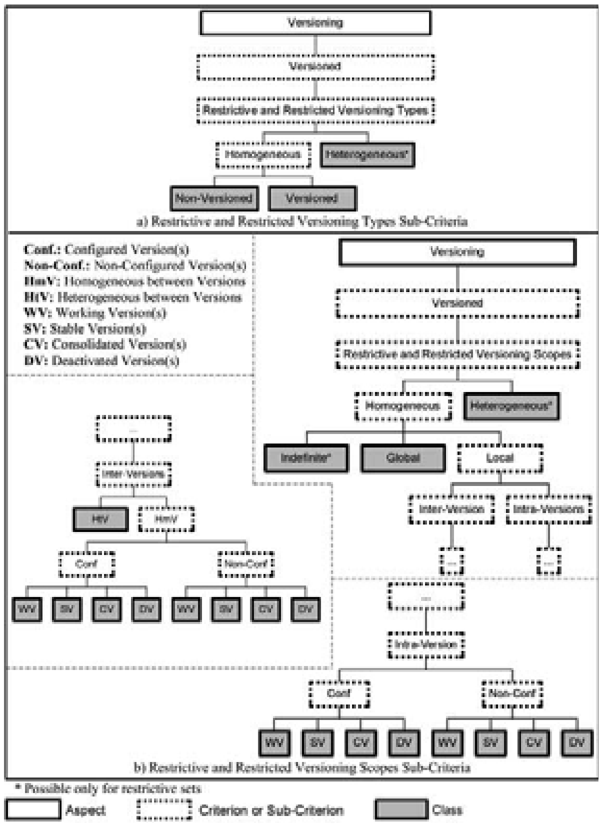

Versioning
Versioned
Restrictive and Restricted Versioning Types
Homogeneous
Heterogeneous*
Non-Versioned
Versioned
a) Restrictive and Restricted Versioning Types Sub-Criteria
Conf.: Configured Version(s)
Non-Conf.: Non-Configured Version(s)
HmV: Homogeneous between Versions
HtV: Heterogeneous between Versions
WV: Working Version(s)
SV: Stable Version(s)
CV: Consolidated Version(s)
DV: Deactivated Version(s)
Restrictive and Restricted Versioning Scopes
Indefinite*
Global
Local
Inter-Version
Intra-Version
HtV
HmV
Conf
Non-Conf
WV
SV
CV
DV
b) Restrictive and Restricted Versioning Scopes Sub-Criteria
* Possible only for restrictive sets
Aspect
Criterion or Sub-Criterion
Class

Figure 3. Versioning aspect.

characteristics, no maintenance process was proposed.

From the best of the authors knowledge, there is no integrity constraints classification that considers the application of time and version concepts in data and in constraints themselves. Thus, the main goal of this research is to define a complete classification for any temporal database with versions support. Ideally, formal methods should be used to prove the completeness of the proposed classification. These methods are usually used to define semantics of system and languages. However, there is no specific language or system in the classification to be evaluated by a mathematical model that describes its behavior. Thus, although the advantages of a formal treatment, its use is not possible in order to prove the classification completeness.

The alternative way used to analyze the completion of the classification was to compare our proposal with other found in literature. To compare these classifications we mapped the equivalent classes of each one of the classifications found in literature to the classification proposed here. This analysis is briefly described as follows.

In the paper [8], Doucet et al. classify temporal constraints according to their *scope* and *binding* on objects, classes and versions. The *state scope* and *reference* are also considered. However, important characteristics are not treated, as the *self-temporality, self-versioning types and scopes, temporality scope, temporality and versioning types, temporal granularity,* and others. Dayal et al. [7] present a good classification, considering the *treatment, activation and deactivation type, state scope, verification points, precedence order, reference, restrictive and restricted scopes* and *binding scopes* only of temporal constraints. Böhlen and Martín [1, 13] present a classification, considering *restrictive and restricted temporality scopes and types, state scope* and *reference* of temporal constraints. In [2], Chomicki analyzes the *state scope, reference* and *treatment* of real-time constraints. Medeiros, Jomier and Cellary [14] present a good classification according to the *origin, vehicle, state scope, restrictive and restricted aspects* and *scopes*, considering only versioning characteristics. Finally, Goonetillake et al. [9] classify versioned constraints by their *origin* and *treatment*.

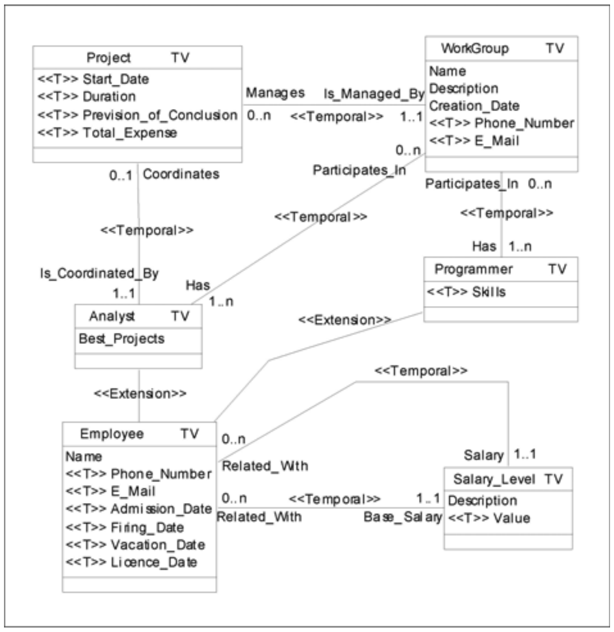


Figure 4. Modeling example.

The result of this experience was that our classification contains, besides others, all the classes represented in the classifications proposed in these work. This result does not prove the classification completeness, but it shows that our classification is more complete than any other analyzed, since it takes into account features that were never considered before, besides all aspects analyzed by these work. Thus, future work on these constraints will have several benefits using this classification as a base.

5. Conclusions and Future Work

This paper presented a detailed classification of integrity constraints considering the application of time and version concepts in data and in constraints themselves. By grouping constraints in classes, based on constraints com-

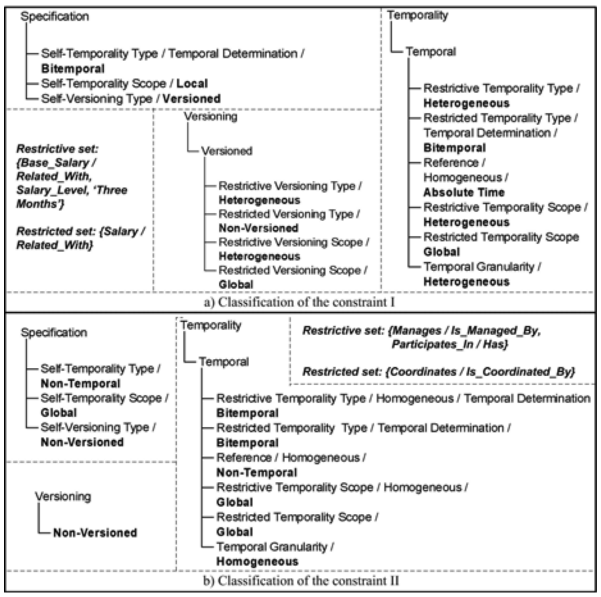

Specification
Self-Temporality Type / Temporal Determination / Bitemporal
Self-Temporality Scope / Local
Self-Versioning Type / Versioned
Temporality
Temporal
Restrictive Temporality Type / Heterogeneous
Restricted Temporality Type / Temporal Determination / Bitemporal
Reference / Homogeneous / Absolute Time
Restrictive Temporality Scope / Heterogeneous
Restricted Temporality Scope Global
Temporal Granularity / Heterogeneous
Restrictive set: {Base_Salary / Related_With, Salary_Level, 'Three Months'}
Restricted set: {Salary / Related_With}
Versioning
Versioned
Restrictive Versioning Type / Heterogeneous
Restricted Versioning Type / Non-Versioned
Restrictive Versioning Scope / Heterogeneous
Restricted Versioning Scope / Global
a) Classification of the constraint I
Specification
Self-Temporality Type / Non-Temporal
Self-Temporality Scope / Global
Self-Versioning Type / Non-Versioned
Temporality
Temporal
Restrictive set: {Manages / Is_Managed_By, Participates_In / Has}
Restricted set: {Coordinates / Is_Coordinated_By}
Restrictive Temporality Type / Homogeneous / Temporal Determination Bitemporal
Restricted Temporality Type / Temporal Determination / Bitemporal
Reference / Homogeneous / Non-Temporal
Restrictive Temporality Scope / Homogeneous / Global
Restricted Temporality Scope / Global
Temporal Granularity / Homogeneous
Versioning
Non-Versioned
b) Classification of the constraint II

Figure 5. Example constraints classification.

mon characteristics, this work is a base for researches related to integrity maintenance processes for any temporal database with versions support, independently of other DBMS (Databases Management System) or data model characteristics. Moreover, due to the existence of specific aspects for temporality and versioning, this classification can be easily adapted and used for snapshot, temporal or versions databases. As shown in Section four, this classification is more complete than others already published since it contains all the classes represented in these work, adding aspects not considered before.

It is crucial to notice that this work provides an important base for the automatic maintenance of constraints that still must be validated by user applications or auditing trail tools, making it easier the definition of software engineering processes for designing database applications that have to deal with historical and evolving data.

In order to get a good integrity maintenance process, based on this classification, the next step consists on the definition of optimized methods for the verification of constraints with common characteristics. Hence, experiments and implementations are being made using the TVM model [15] and the TVCL specification language [4], based on triggers and on a layer to emulate a TVM DBMS under a relational DBMS. The main problem faced is to define an optimized algorithm to verify constraints with event based verification points (Section 2.1). Finally, researches to demonstrate the minimalism of the classification are being made.

References

[1] M. Bohlen, "Valid time integrity constraints," University of Arizona, Tucson, AZ, Tech. Rep. 94-30, 1994.

[2] J. Chomicki, "Real-time integrity constraints." in *PODS*. ACM Press, 1992, pp. 274–282.

[3] J. E. Cooling and J. Cooling, *Software Engineering for Real-Time Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[4] R. L. F. Cordeiro and et al., "Tvcl - temporal versioned constraint language." in *SBBD*, C. A. Heuser, Ed. UFU, 2005, pp. 55–69.

[5] R. L. F. Cordeiro, C. S. Santos, and N. Edelweiss, "Integrity constraints for temporal versions model: classification, modeling and verification," in *Workshop de Teses e Dissertações em Banco de Dados*, Brasília, DF, Brazil, 2004, in conjunction with SBBD.

[6] C. A. Davis and et al., "Deriving spatial integrity constraints from geographic application schemas." in *Encyclopedia of Database Tech. and Applications*, L. C. Rivero and et al, Eds. IG, 2005, pp. 176–183.

[7] U. Dayal and et al., "The hipac project: Combining active databases and timing constraints." *SIGMOD Record*, vol. 17, no. 1, pp. 51–70, 1988.

[8] A. Doucet and et al., "Using database versions to implement temporal integrity constraints." in *CDB*, ser. Lecture Notes in Computer Science, V. Gaede and et al., Eds., vol. 1191. Springer, 1997, pp. 219–233.

[9] J. S. Goonetillake and et al., "An integrity constraint management framework in engineering design," *Comput. Ind.*, vol. 48, no. 1, pp. 29–44, 2002.

[10] C. S. Jensen and et al., "The consensus glossary of temporal database concepts - february 1998 version." in *Temporal Databases, Dagstuhl*, 1997, pp. 367–405.

[11] R. H. Katz, "Toward a unified framework for version modeling in engineering databases," *ACM Comput. Surv.*, vol. 22, no. 4, pp. 375–409, 1990.

[12] A. T. Lazzaretti and R. dos Santos Mello, "A domain integrity constraint control for xml documents." in *SBBD*, 2005, pp. 115–129.

[13] C. Martín, "Analyzing temporal integrity constraints to obtain the minimum number of transition rules," Department of Llenguatges i Sistemes Informàtics, Tech. Rep. LSI-01-52-R, 2001.

[14] C. B. Medeiros and et al., "Maintaining integrity constraints across versions in a database," in *SBBD*. PB: SBC, 1993.

[15] M. M. Moro and et al., "A temporal versions model for time- evolving systems specification." in *SEKE*, 2001, pp. 252–259.

[16] R. Snodgrass and C. Jensen, *Temporal Databases*. Morgan Kaufmann Publishers, 2006.

[17] B. Westfechtel and et al., "A layered architecture for uniform version management," *IEEE Trans. on Software Eng.*, vol. 27, no. 12, pp. 1111–1133, 2001.

Generating Linear Temporal Logic Formulas for Pattern-Based Specifications

Salamah Salamah, Vladik Kreinovich, and Ann Q. Gates
Dept. of Computer Science, University of Texas at El Paso
El Paso, TX 79968, USA
isalamah, vladik, and agates@utep.edu

Abstract

Software property classifications and patterns, i.e., high-level abstractions that describe program behavior, have been used to assist practitioners in specifying properties. The Specification Pattern System (SPS) provides descriptions of a collection of patterns. Each pattern is associated with a scope that defines the extent of program execution over which a property pattern is considered. Based on a selected pattern, SPS provides a specification for each type of scope in multiple formal languages including Linear Temporal Logic (LTL). The Property Specification tool (Prospec) extends SPS by introducing the notion of Composite Propositions (CP), that classify sequential and concurrent behavior over pattern and scope parameters.

In this work, we present an approach to support the automated generation of Linear Temporal Logic (LTL) formulas for complex pattern-based software specifications that use CPs. We define general LTL formulas for the Response pattern, and provide formal descriptions of the different CP classes. In addition, we formally describe the Response pattern and the different scopes that use CPs.

1 Introduction and Motivation

Although formal verification methods such as model checking [6], theorem proving [12], and runtime monitoring [14] have been shown to improve the dependability of programs, software development professionals have yet to adopt them. The reasons for this hesitance include the high level of mathematical sophistication required for reading and writing formal specifications needed for the use of these approaches [5].

Linear Temporal Logic (LTL) is a prominent formal specification language. LTL's popularity stems from the fact that it is highly expressive and that it is widely used in formal verification tools. Such tools include, the popular model checker SPIN [6] that has been used in the verification of a variety of systems such as security protocols [1] and flight software [8]. In addition, LTL is used by the Model checkers NuSMV [2] and Java Path-Finder [7]. LTL is also used in the runtime verification of Java programs [14].

Specification Pattern System (SPS). The problem of generating formal specification is difficult, and the temporal nature of LTL makes it even harder to write specifications. The Specification Pattern System [3] defines patterns and scopes to assist the practitioner in formally specifying software properties.

Patterns capture the expertise of developers by describing solutions to recurrent problems. Each pattern describes the structure of specific behavior, defines the pattern's relationship with other patterns, and defines the scope over which the property holds.

The main patterns defined by SPS are: *Universality(P), Absence(P), Existence(P), Precedence(P,Q)*, and *Response(P,Q)*. $Universality(P)$ states that P is true in every point of the execution; $Absence(P)$ states that P is never true during the execution; $Existence(P)$ states that P is true at some point in the execution; $Precedence(P,Q)$ states that if P holds, then Q must hold before P; and $Response(P,Q)$ states that if P holds, then Q must hold at a future state. Response properties represent a temporal relation called cause-effect between two propositions. SPS restricts the specification of sequences to precedence and response patterns.

In SPS, each pattern is associated with a *Scope* that defines the extent of program execution over which a property pattern is considered. There are five types of scopes defined in SPS: $Global$, $Before\ R$, $After\ L$, $Between\ L\ And\ R$, and $After\ L\ Until\ R$. $Global$ denotes the entire program execution; $Before\ R$ denotes the execution before the first time R occurs, i.e., R holds; $After\ L$ denotes execution after the first time L occurs; $Between\ L\ And\ R$ denotes the execution between intervals defined by L and R; and $After\ L\ Until$ denotes the execution between intervals defined by L and R and, in the case when R does not occur, until the end of execution.

The SPS website[4] provides a formal description of patterns and scopes in several specification languages including Linear Temporal Logic (LTL), Computational Tree Logic (CTL), and Graphical Interval Logic (GIL). These formulas are provided for patterns and scopes involving *single propositions*, i.e., patterns and scopes in which P, Q, L, and R each occur at a single moment of time. SPS also provides formulas for the cases of multiple cause and single effect (when P is made of multiple propositions and Q is single) and of single effect and multiple cause (when P is single and Q is composed of multiple propositions).

Composite Propositions (CP). In practical applications, we often need to describe properties where one or more of the pattern or scope parameters are made of multiple propositions, i.e., composite propositions (CP). For example, the property that every time data is sent at moment t_i the data is read at moment $t_1 \geq t_i$, the data is processed at moment t_2, and data is stored at moment t_3. This property can be described using the Response pattern where P stands for "Data is sent', and Q is composed of q_1, q_2, and q_3 (data is read, data is processed, and data is stored, respectively).

To describe such patterns, Mondragon et al. [10] extended SPS by introducing a classification for defining sequential and concurrent behavior to describe patterns and scopes parameters. Specifically, the work formally described several types of composite propositions (CP) and showed how these descriptions can be translated into LTL.

Some of the corresponding patterns can be described in a Future Interval Logic (FIL) language, a language which is similar to LTL, but less expressive than LTL. For example, in FIL, we cannot describe a practically important property that an event p must hold at the next moment of time. The corresponding translations have been implemented in the Property Specification tool (Prospec) [9] that uses patterns and scopes involving composite propositions to generate formal specifications in FIL. Similarly to LTL specifications, FIL specifications can also be used to formally verify software. However, in comparison to LTL, FIL has two limitations: first, due to the limited expressiveness of FIL, not all patterns and scopes involving composite propositions can be represented; second, FIL is not as widely used in formal verification tools, so the use of FIL restricts the software engineer's ability to use the resulting specifications.

It is, therefore, important to provide a translation of all possible patterns and scopes involving composite propositions into the more expressive (and more widely used) language LTL. It is also important to show that these translations are indeed correct for all patterns and scopes. In this paper, due to page limitations, we concentrate on the case of the most widely used $Response$ pattern [3]. The matter of proving the correctness of these translations is left as a future work.

The rest of the paper is outlined as follows. We start with a brief description of LTL. Section 3 provides a formal description of the different CP classes. Section 4 formally describes the $Response$ pattern, and Section 5 provides the description of the LTL formulas for the $Response$ pattern within the $Global$ scope. Sections 6 and 7 provide formal definition of the other scopes and describe the LTL formulas for the $Response$ pattern within these scopes.

2 LTL: A Brief Reminder

Formulas of LTL are constructed inductively from elementary propositions $p_1, p_2, \ldots$ by applying Boolean connectives $\neg$, $\vee$, and $\wedge$ and temporal operators X (*next*), U (*until*), $\diamond$ (*eventually*), and $\Box$ (*always*). These formulas assume discrete time, i.e., moments $t = 0, 1, 2, \ldots$ The meaning of the temporal operators is straightforward. The formula XP holds at the moment t means that P holds at the next moment of time $t+1$. To check whether $P\,U\,Q$ holds at the moment t, we must find the first moment of time $s \geq t$ at which Q is true; then, the truth of $P\,U\,Q$ means that P is true at all moments of time t' for which $t \leq t' < s$ (if Q never happens at moment t or later, then $P\,U\,Q$ is false). The formula $\diamond P$ holds at moment t means P is true at some moment of time $t' \geq t$. Finally, the formula $\Box P$ holds at moment t if P is true at all moments of time $t' \geq t$.

3 Composite Propositions: A Formal Description

We consider the following 8 CP classes: $AtLeastOne_C$, $AtLeastOne_E$, $Parallel_C$, $Parallel_E$, $Consecutive_C$, $Consecutive_E$, $Eventual_C$, and $Eventual_E$. CP classes of type T_C (called *condition type*) are defined as follows:

- $AtLeastOne_C(p_1, \ldots, p_n)$ means that at least one of p_i holds at a given moment of time t, i.e., that $p_1 \vee \ldots \vee p_n$ holds;
- $Parallel_C(p_1, \ldots, p_n)$ means that all p_i hold at time t, i.e., $p_1 \wedge \ldots \wedge p_n$;
- $Consecutive_C(p_1, \ldots, p_n)$ means that p_1 holds at moment $t_1 = t$, p_2 holds at moment $t_2 = t+1$, …, and p_n holds at moment $t_n = t + (n-1)$; the corresponding LTL formula $Consecutive_C^{LTL}(p_1, \ldots, p_n)$ is $(p_1 \wedge X(p_2 \wedge X(\ldots \wedge X(p_n)) \ldots))$;
- $Eventual_C(p_1, \ldots, p_n)$ means that p_1 holds at $t_1 = t$, p_2 holds at some moment $t_2 > t_1$, …, and p_n holds at some moment $t_n > t_{n-1}$; the corresponding LTL formula $Eventual_C^{LTL}(p_1, \ldots, p_n)$ is

$$p_1 \wedge X(\neg p_2\, U\, (p_2 \wedge X(\neg p_3\, U\, (\ldots \wedge X(\neg p_n\, U\, p_n)) \ldots))).$$

CP classes of the type T_E (called *event type*) can be defined in terms of a new class of auxiliary formulas $T_H(p_1, \ldots, p_n)$. The main motivation for T_H is that in T_C we only required each p_i to hold at a certain moment of time t_i, and we do not make any assumptions about other propositions p_j ($j \neq i$) at this moment t_i. In some practical applications, it is important to require that p_i become true in the prescribed order, i.e., that not only p_i becomes true at moment t_i, but that it also remains false until then. In precise terms, we have the following:

Definition 1

- *By a* CP class*, we mean one of the following four terms: $AtLeastOne$, $Parallel$, $Consecutive$, and $Eventual$.*
- *By a* type of proposition*, we will mean C, E, or H; the type H will be called* auxiliary.
- *By a* composite proposition *P, we mean an expression of the type $T_y(p_1, \ldots, p_n)$, where T is a CP class, y is a type of proposition, and $p_1, \ldots, p_n$ are (single) propositions.*

Definition 2

- *For T = AtLeastOne and for T = Parallel, $T_H(p_1, \ldots, p_n)$ means the same as $T_C(p_1, \ldots, p_n)$, and $T_H^{LTL}(p_1, \ldots, p_n)$ is defined as $T_C^{LTL}(p_1, \ldots, p_n)$.*
- *For T = $Consecutive$ and for T = $Eventual$, $T_H(p_1, \ldots, p_n)$ means*

  $$T_C(p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n, p_2 \wedge \neg p_3 \wedge \ldots \wedge \neg p_n, \ldots, p_n),$$

  and $T_H^{LTL}(p_1, \ldots, p_n)$ is defined as

$$T_C^{LTL}(p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n, p_2 \wedge \neg p_3 \wedge \ldots \wedge \neg p_n, \ldots, p_n).$$

For example, $Consecutive_H(p_1, \ldots, p_n)$ holds at the moment t if:

- at the moment of time t, the proposition p_1 holds and all the further propositions $p_2, \ldots, p_n$ are false;
- at the next moment of time $t+1$, the proposition p_2 holds and all the further propositions $p_3, \ldots, p_n$ are false; ..., and
- at the moment $t + (n-1)$, the proposition p_n holds.

In other words, $Consecutive_H(p_1, \ldots, p_n)$ holds at the moment t if the following formula for T_H^{LTL} holds at moment t:

$$(p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge \ldots \wedge \neg p_n \wedge$$
$$X(p_2 \wedge \neg p_3 \wedge \ldots \wedge \neg p_n \wedge$$
$$X(\ldots \wedge X(p_{n-1} \wedge \neg p_n \wedge X(p_n)) \ldots)))$$

Definition 3 *We say that a composite proposition $T_E(p_1, \ldots, p_n)$* holds *at the moment t if at the moment t, all propositions p_i are false, and they remain false until some moment t' when the composite proposition $T_H(p_1, \ldots, p_n)$ becomes true.*

For example, a composite proposition $AtLeastOne_E(p_1, \ldots, p_n)$ holds at moment t if all the propositions $p_1, \ldots, p_n$ are false at the moment t, and at least one of these propositions $p_1, \ldots, p_n$ is true at some future moment of time $t' > t$.

Definition 4 *By an* LTL formula *$T_E^{LTL}(p_1, \ldots, p_n)$ for $T_E(p_1, \ldots, p_n)$, we mean the formula*

$$(\neg p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n) \wedge$$
$$((\neg p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n)\, U\, T_H^{LTL}(p_1, p_2, \ldots, p_n)).$$

Theorem 1 *For every composite proposition P and for every moment of time t, P holds at the moment t if and only if the corresponding LTL formula P^{LTL} holds at this moment t.*

Comment. Due to page limitations, detailed proofs are given in [13].

4 Response Pattern within Global Scope: Case of Composite Propositions

Patterns such as $Response$ were introduced in [3] for single propositions. In particular, "q responds to p within global scope" means that every time the property p holds, the property q must hold either after it or at this same moment of time. To extend this description to composite propositions, we therefore need to extend the notion "after" to such propositions.

Single propositions describe a single moment of time. In general, composite propositions deal with a time interval (although, of course, this time interval may be degenerate, i.e., it may consist of a single moment of time). Specifically, for every composite proposition $P = T(p_1, \ldots, p_n)$, there is a starting moment b_P – the first moment of time when one of the propositions p_i becomes true, and the ending moment e_P – the first moment of time when the condition T is fulfilled. These moments can be defined as follows.

Definition 5

- *For a composite proposition P of the type $AtLeastOne_C(p_1, \ldots, p_n)$ that holds at the moment t, we take $b_P(t) = e_P(t) = t$.*
- *For a composite proposition P of the type $AtLeastOne_E(p_1, \ldots, p_n)$ that holds at the moment t, we take, as $e_P(t)$, the first moment of time $t' > t$ at which one of the propositions p_i becomes true, and $b_P(t) = e_P(t) - 1$.*

- *For a composite statement P of the type $Parallel_C(p_1, \ldots, p_n)$ that holds at the moment t, we take $b_P(t) = e_P(t) = t$.*
- *For a composite proposition P of the type $Parallel_E(p_1, \ldots, p_n)$ that holds at the moment t, we take, as $b_P(t) = e_P(t)$, the first moment of time $t' > t$ at which all the propositions p_i become true.*
- *For a composite proposition P of the type $Consecutive_C(p_1, \ldots, p_n)$ that holds at the moment t, we take $b_P(t) = t$ and $e_P(t) = t + (n - 1)$.*
- *For a composite proposition P of the type $Consecutive_E(p_1, \ldots, p_n)$ that holds at the moment t, we find the first moment of time $t' > t$ at which p_1 becomes true, and take $b_P(t) = t' - 1$ and $e_P(t) = t' + (n - 1)$.*
- *For a composite proposition P of the type $Eventual_C(p_1, \ldots, p_n)$ that holds at the moment t, we take $b_P(t) = t$, and as $e_P(t)$, we take the first moment of time $t_n > t$ at which the last proposition p_n is true and the previous propositions $p_2, \ldots, p_{n-1}$ were true at the corresponding moments of time $t_2, \ldots, t_{n-1}$ for which $t < t_2 < \ldots < p_{n-1} < t_n$.*
- *For a composite proposition P of the type $Eventual_E(p_1, \ldots, p_n)$ that holds at the moment t, we find the first moment of time t_1 at which p_1 becomes true, and take $b_P(t) = t_1 - 1$; as $e_P(t)$, we take the first moment of time t_n at which the last proposition p_n becomes true.*

Definition 6 *Let P and Q be composite propositions. We say that Q* responds to *P* within global scope *if once P holds at some moment t, then Q also holds at some moment t' for which $b_Q(t') \geq e_P(t)$.*

5 LTL Formulas For Response Pattern Within Global Scope: Case of Composite Propositions

To describe LTL formulas $\mathcal{P}^{LTL}$ corresponding to patterns $\mathcal{P}$ with composite propositions, we need to describe new "and" operations. If we consider a non-temporal formula A as a particular case of LTL formulas, then A means simply that the statement A holds at the given moment of time, and the formula $A \wedge B$ means that both A and B hold at this same moment of time.

For a general LTL formula A, the fact that A holds at a moment of time t means that some "subformulas" of A hold at this same moment of time, while some other subformulas may hold at different moments of time. For example, the formula $p_1 \wedge X p_2$ means that p_1 holds at the moment t while p_2 holds at the moment $t + 1$. It is therefore desirable to come up with a different "and" operation that would ensure that B holds at all the moments of time at which different "subformulas" of A hold. For example, for this new "and" operation, $(p_1 \wedge X p_2)$ and B would mean that B holds both at the moment t and at the moment $t + 1$.

We will denote this new "and" operation by $\&_r$. We will also need the operation $A \,\&_l\, B$, which will indicate that B holds at the *last* of A-relevant moments of time. Let us give formal definitions of these operations. We only give the definition for the particular cases needed in our patterns.

Definition 7

- *When P is of the type $T_C(p_1, \ldots, p_n)$ or $T_H(p_1, \ldots, p_n)$, with T = Parallel or T = AtLeastOne, then $P \,\&_r\, A$ is defined as $P \wedge A$.*
- *When P is of the type $T_C(p_1, \ldots, p_n)$, with T = Consecutive or T = Eventual, then $P \,\&_r\, A$ is defined as $T_C(p_1 \wedge A, \ldots, p_{n-1} \wedge A, p_n \wedge A)$.*
- *When P is of the type $T_H(p_1, \ldots, p_n)$, with T = Consecutive or T = Eventual, then $P \,\&_r\, A$ is defined as*

$$T_C(p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n \wedge A, \ldots, p_{n-1} \wedge \neg p_n \wedge A, p_n \wedge A).$$

- *When P is of the type $T_E(p_1, \ldots, p_n$, then $P \,\&_r\, A$ is defined as*

$$(\neg p_1 \wedge \ldots \wedge \neg p_n \wedge A) \wedge$$

$$((\neg p_1 \wedge \ldots \wedge \neg p_n \wedge A)\, U\, (T_H(p_1, \ldots, p_n \wedge A)))).$$

Definition 8

- *When P is of the type $T_C(p_1, \ldots, p_n)$ or $T_H(p_1, \ldots, p_n)$, with T = Parallel or T = AtLeastOne, then $P \,\&_l\, A$ is defined as $P \wedge A$.*
- *When P is of the type $T_C(p_1, \ldots, p_n)$, with T = Consecutive or T = Eventual, then $P \,\&_l\, A$ is defined as $T_C(p_1, \ldots, p_{n-1}, p_n \wedge A)$.*
- *When P is of the type $T_H(p_1, \ldots, p_n)$, with T = Consecutive or T = Eventual, then $P \,\&_l\, A$ is defined as*

$$T_C(p_1 \wedge \neg p_2 \wedge \ldots \wedge \neg p_n, \ldots, p_{n-1} \wedge \neg p_n, p_n \wedge A).$$

- *When P is of the type $T_E(p_1, \ldots, p_n)$, then $P \,\&_l\, A$ is defined as*

$$(\neg p_1 \wedge \ldots \wedge \neg p_n) \wedge$$

$$((\neg p_1 \wedge \ldots \wedge \neg p_n)\, U\, (T_H(p_1, \ldots, p_n) \,\&_l A)).$$

Definition 9 *An LTL formula corresponding to Response (P, Q) is*

$$\Box(P^{LTL} \rightarrow (P^{LTL} \,\&_l\, \Diamond Q^{LTL})).$$

For example, if P is of type $Consecutive_C(p_1, p_2)$ and Q is of type $Parallel_E(q_1, q_2)$, then $\mathcal{P}_1^{LTL}$ is:

$$\Box((p_1 \wedge Xp_2) \rightarrow (p_1 \wedge X(p_2 \wedge \Diamond$$

$$((\neg q_1 \wedge \neg q_2) \wedge ((\neg q_1 \wedge \neg q_2)\, U\, (q_1 \wedge q_2))))))$$

Theorem 2 *For the formula "Q responds to P within a global scope", this formula holds at the moment t if and only if the corresponding LTL formula holds at this moment t.*

Comment. Similar results hold for all other patterns from Section 1.1.

6 Other Scopes: Motivations and Definitions

In the previous text, we considered all the propositions in the "global scope", when, e.g., the existence of P means that P holds for some moment of time t. In practice, we are often only interested in moments of time that occur before or after a certain event. For single propositions, the following scopes were proposed in [3]: "before R", "after L", "between L and R", and "after L until R".

We want to extend the above definitions of patterns to the case of scopes. For example, we want to define "Q precedes P within a scope s", meaning that every time P holds within the scope s, Q should hold at a preceding moment of time within the same scope.

In general, a composite proposition P holds at a moment t if several single propositions hold at different moments of time. It is therefore reasonable to say that P occurs within a scope if all these single propositions occur within this scope, i.e., if the whole time interval $[t, e_P(t)]$ is within the scope s. Let us formally describe what this means.

Since we consider composite propositions P, it is natural to allow L and R to be composite propositions as well. A composite proposition P, in general, does not occur at a single moment of time, it usually has a starting moment b_P and the ending moment e_P. So, for composite propositions, "before R" can be naturally interpreted as "before the starting moment of R", and "after L" can be naturally interpreted as "after the ending moment of L". To describe this formally, we will use the above definitions (of section 4) of the starting and ending moments.

Definition 10 *By a* scope statement*, we mean a statement of the type "before R", "after L", "between L and R", and "after L until R", where L and R are composite propositions.*

Definition 11 *By a* scope *corresponding to the scope statement, we mean the following time interval:*

- *For a scope statement "before R", there is exactly one scope – the interval $[0, b_R(t_f))$, where t_f is the first moment of time when R becomes true.*
- *For a scope statement "after L", there is exactly one scope – the interval $[e_L(t_f), \infty)$, where t_f is the first moment of time when L becomes true.*
- *For a scope statement "between L and R", a scope is an interval $[e_L(t_L), b_R(t_R))$, where t_L is a moment of time at which L holds and t_R is the first moment of time $> e_L(t_L)$ when R becomes true.*
- *For a scope statement "after L until R", in addition to scopes corresponding to "between L and R", we also allow a scope $[e_L(t_L), \infty)$, where t_L is a moment of time at which L holds and for which R does not hold at any moment $t > e_L(t_L)$.*

Definition 12 *Let P and Q be composite propositions, and let s be a scope.*

- *We say that P s*-holds *at a moment $t_P \in s$ if P holds at the moment t_p and the ending moment of time $e_P(t_p)$ belongs to the same scope s.*
- *We say that Q* responds to P within the scope *s if once P s-holds at some moment t, then Q also s-holds at some moment t' for which $b_Q(t') \geq e_P(t)$.*

Definition 13 *We say that a pattern* holds *within a scope statement if it holds within each scope corresponding to this scope statement.*

Now that we have defined what it means for a pattern to hold within the different types of scopes, we are ready to provide the LTL description of the five patterns within the scopes ("before R", "after L", "between L and R", and "after L until R").

7 Response Pattern For Scopes Other Than Global: Case of Composite Propositions

Definition 14 *Let $\mathcal{P}$ be a response pattern, i.e., "Q responds to P", and let $\mathcal{P}_{<R}$ denote this pattern in the scope "before R", i.e., a pattern "Q Responds to P Before R". Then, the corresponding LTL-formula $\mathcal{P}_{<R}^{LTL}$ has the following form:*

- *when R is of the type $T_C(r_1, \ldots, r_n)$, then $\mathcal{P}_{<R}^{LTL}$ is*

$$\neg((\neg R^{LTL})\, U\, ((P^{LTL} \&_r \neg R^{LTL})$$

$$\&_l((\neg(Q^{LTL} \&_r \neg R^{LTL}))\, U\, R^{LTL})));$$

- *when R is of the type $T_E(r_1, \ldots, r_n)$, then $\mathcal{P}_{<R}^{LTL}$ is*

$$\neg((\neg((\neg r_1 \wedge \neg r_2 \wedge \ldots \wedge \neg r_n) \wedge X(R_H^{LTL})))U$$

$$((P^{LTL} \&_r \neg R_H^{LTL}) \&_l$$

$$((\neg(Q^{LTL} \&_r \neg R_H^{LTL}))U R_H^{LTL}))).$$

For example, the formula $\mathcal{P}_2^{LTL}$ for Q responds to P $Before$ R where R is $AtLeastOne_C(r_1, r_2)$, P is $Consecutive_C(p1, p2)$, and Q is $Parallel_C(q_1, q_2)$ is:

$$\neg((\neg(r_1 \vee r_2))\, U\, (((p_1 \wedge (\neg(r_1 \vee r_2)) \wedge X((p_2 \wedge \neg(r_1 \vee r_2))$$

$$\wedge(((\neg((q_1 \wedge q_2 \wedge \neg(r_1 \vee r_2))))\, U\, (r_1 \vee r_2))))))))$$

Pattern formulas for the scopes "After L", "Between L and R", and "After L until R" can be generated using the formula $\mathcal{P}^{LTL}$ for the Global scope and the formula $\mathcal{P}_{<R}^{LTL}$ for the scope "before R":

Definition 15 *For a pattern $\mathcal{P}$ in the "After L" scope, the corresponding LTL formula is:*

$$\neg((\neg L^{LTL})\, U\, (L^{LTL} \&_l \neg \mathcal{P}^{LTL})).$$

Definition 16 *For a pattern $\mathcal{P}$ in the "Between L And R" scope, the corresponding LTL formula is as follows:*

- $\square((L^{LTL} \&_l \neg R^{LTL}) \rightarrow (L^{LTL} \&_l \mathcal{P}_{<R}^{LTL})$ *if R is of type C;*
- $\square(L^{LTL} \rightarrow (L^{LTL} \&_l \mathcal{P}_{<R}^{LTL})))$ *if R is of type E.*

Definition 17 *For a pattern $\mathcal{P}$ in the "After L until R" scope, the corresponding LTL formula is:*

- *if R is of type C, then*

$$\square((L^{LTL} \&_l \neg R^{LTL}) \rightarrow$$

$$(L^{LTL} \&_l ((\mathcal{P}_{<R}^{LTL} \wedge ((\neg \diamond R^{LTL}) \rightarrow \mathcal{P}^{LTL})))));$$

- *if R is of type E, then*

$$\square(L^{LTL} \rightarrow$$

$$(L^{LTL} \&_l ((\mathcal{P}_{<R}^{LTL} \wedge ((\neg \diamond R^{LTL}) \rightarrow \mathcal{P}^{LTL}))))).$$

Theorem 3 *For every response formula $\mathcal{P}$ and for every scope s, the formula $\mathcal{P}$ holds within the scope s at the moment t if and only if the corresponding LTL formula holds at this moment t.*

Comment. Similar results hold for all other patterns from Section 1.1.

8 Conclusions

In many real-life situations, software specifications involve complex relations between events and conditions at different moments of time. Such specifications can be formally described in terms of patterns and scopes; however, at present, there is no practically useful general tool for checking such specifications.

In this paper, we describe a (reasonably general) class of such patterns-and-scopes specifications. We prove that specifications from this class can be reformulated in terms of the equivalent formulas of Linear Temporal Logic (LTL). Since there exist efficient tools for checking LTL-based specifications, this reformulation thus enables us to check the original complex specifications.

Acknowledgments. This work is funded in part by NSF grant EAR-0225670, by Texas Department of Transportation grant No. 0-5453, and by the Japan Advanced Institute of Science and Technology (JAIST) International Joint Research Grant 2006-08. The authors are thankful to the anonymous referees for important suggestions.

References

[1] Audun, J., "Security Protocol Verification using SPIN", *Proceedings of the First SPIN Workshop*, October 1995.

[2] Cimatti, A., E. Clarke, F. Giunchiglia, and M. Roveri, "NUSMV: a new Symbolic Model Verifier", *Int'l Conf. on Computer Aided Verification CAV*, July 1999.

[3] Dwyer, M. B., G. S. Avrunin, and J. C. Corbett, "Patterns in Property Specification for Finite State Verification," *Proc. of the 21st Int'l Conf. on Software Engineering*, Los Angeles, CA, 1999, 411–420.

[4] http://patterns.projects.cis.ksu.edu/

[5] Hall, A., "Seven Myths of Formal Methods," *IEEE Software*, September 1990, 11(8)

[6] Holzmann G. J. *The SPIN Model Checker: Primer and Reference Manual,* Addison-Wesley Professional, 2004.

[7] Havelund, K., and T. Pressburger, "Model Checking Java Programs using Java PathFinder", *Int'l J. on Software Tools for Technology Transfer*, 2(4), April 2000.

[8] Glück, P, R., and G, J. Holzmann, "Using SPIN Model Checking for Flight Software Verification" *IEEE Aerospace Conf.*, March 2002

[9] Mondragon, O., A. Q. Gates, and S. Roach, "Prospec: Support for Elicitation and Formal Specification of Software Properties," in O. Sokolsky and M. Viswanathan (Eds.), *Proc. of Runtime Verification Workshop, ENTCS*, 89(2), 2004.

[10] Mondragon, O. and A. Q. Gates, "Supporting Elicitation and Specification of Software Properties through Patterns and Composite Propositions," *Int'l J. of Software Engineering and Knowledge Engineering,* 14(1), Feb. 2004.

[11] Manna, Z. and A. Pnueli, "Completing the Temporal Picture," *Theoretical Computer Science,* 83(1), 1991, 97–130.

[12] Rushby, J., "Theorem Proving for Verification," *Modelling and Verification of Parallel Processes,* June 2000.

[13] Salamah, I. S., *Defining LTL formulas for complex pattern-based software properties*, University of Texas at El Paso, Department of Computer Science, PhD Dissertation, May 2007.

[14] Stolz, V. and E. Bodden, "Temporal Assertions using AspectJ", *5th Workshop on Runtime Verification,* July 2005.

Ontology Based Classification Generating Method for Browsing-Based Component Retrieval

Ge Li, Lu Zhang, Bing Xie[*], Weizhong Shao
Institute of Software, School of Electronics Engineering and Computer Science, Peking University
Key laboratory of High Confidence Software Technologies, Ministry of Education, P. R. China
{lige, zhanglu, xiebing, wzshao}@sei.pku.edu.cn

Abstract

Reuse repository is an essential element in component-based software development (CBSD). To facilitate reusers to retrieve components efficiently, it is typical for reuse repository to provide a classification to represent the components. Based on this classification the reusers can retrieve components by browsing the repository (called browsing-based retrieval). Although browsing-based retrieval is superior to querying-based retrieval in some aspects, the tedious retrieval process is its main drawback, because the classification, by which the browsing process navigated, is usually inefficient. In this paper, we proposed an ontology based approach to generate efficient classification using the components' indexing information. According to our experimental results on real data, the classification can navigate the browsing-based component retrieval efficiently.

1 Introduction

Component-based software development (CBSD) has been widely viewed as a promising way to improve both the productivity and quality of software development [5],[6].During the process of CBSD, the components are usually picked up from a reuse repository, where the software components are stored and searched. An important prerequisite for successful CBSD is the selection of many suitable components from the reuse repository.

To facilitate reusers to retrieve components efficiently, it is typical for reuse repository to provide a classification to represent the components in them. Based on this classification the reuses can retrieve component by browsing the repository, which is called browsing-based retrieval. Browsing-based retrieval usually works stepwise, and requires no search key [3, 2], so it is helpful for an inexperienced retriever, who has no precise expression of the requirements of his or her desired components. However, there is an obvious disadvantage for browsing-based retrieval: the retrieval process usually involves long retrieval sequences and becomes tedious. This is because the classification, by which the browsing process navigated, is usually inefficient and can not classify the components effectively. This paper focuses on the problem of how to build a efficient classification to navigate the browsing-based retrieval.

[*]Corresponding author

2 Motivation

The classification in reuse repository is a hierarchical tree structure, in which each node includes a set of components and is labeled by a word, the parent node includes all the components in its children nodes. Navigated by this structure, the browsing-based retrieval process becomes a process to follow a route in the classification.

In browsing-based component retrieval process, the later browsing step is a refinement based on the returned result set of the previous step, if more components are returned in the previous step, more browsing steps may be needed in the subsequent browsing; and if fewer components can be returned in the previous step, fewer browsing steps may be needed in the subsequent browsing. Thus, in an efficient browsing sequence, the number of returned components should be confined as much as possible in each of the retrieval steps, especially the early steps.

As mentioned above, the browsing-based retrieval is navigated by the classification in the reuse repository, if the distribution of the components under the classification is coarse and unbalanced, more components would be returned during the browsing progress; on the contrary, if the distribution of the components under the classification is refined and balanced, then the number of returned components would be confined in browsing process, and the browsing-based retrieval would become efficiently. So, it is necessary to build an refined and balanced classification for browsing-based retrieval.

3 The Proposed Approach

As mentioned above, each node in the classification is labeled by a word. So, a classification can be seen as a category of words, in which a relationship between a parent node and a child node is a representation of one kind of relationships between two words. A words category (or a classification) can be built if a set of words and the possible relationships among them were predefined. Furthermore, different word categories or classifications can be built using the same set of words and their possible relationships, because different children words can be selected for a given word in a certain environment.

Actually, different parent-children words groups in the predefined set usually have different capability (which is referred to as the "importance" in this paper) in distribute the retrieved component set. More important groups can distribute the components more refined and balanced. Thus, if more important groups can be used earlier in the retrieval process, the retrieved component set can be confined to be smaller, then the number of returned components should be confined in browsing steps, and the browsing-based retrieval would become efficiently. Therefore, the primary concern of our method is how to build the classification by using the most important parent-children words group firstly.

In our approach, the predefined words and their possible relationships will be stored in an ontology, and then, an optimal or sub-optimal balanced classification that can distribute the components efficiently will be generated using this ontology. In the following sections, the definition of the ontology and the generated classification will be introduced firstly, and then we will propose our new approach.

3.1 Definition of Ontology

In our approach, the ontology, which contains the predefined words and their relationships, consists of 3 elements $\{$ $Words$, $Relations$, $Axioms$ $\}$, where $Words$ represents a set of words used in the possible classifications; $Relations$ represents a set of the relationships among the words in $Words$, each relationship represents a binary parent-child relationship between two words; $Axioms$ represents a set of axioms, each axiom is a constraint on the relationships between words, each constraint can be expressed like a Prolog-like rule [1]. This ontology can be represented in any ontology representation language such as RDF(S), DAML+OIL and others. In this paper, we will focus on the words, relations of an ontology, the axioms will be useful when ontology based reasoning is required.

3.2 The Generated Classification

In our approach, the classification is defined as a tree structure, denoted as $N = (V, E)$, where V is the set of nodes labeled with the words in the ontology, and E=$\{\langle v_1, v_2 \rangle \mid v_1, v_2 \in V\}$ is the set of edges. For an edge $\langle v_1, v_2 \rangle$, it shows that the word labeled on node v_1 is a parent of the word labeled on v_2.

3.3 Classification Generating Method

Based on the above approach, the problem of how to generate the efficient classification can be reduced into the problem of how to select the most important parent-children words group from the predefined ontology. In the following, we will present a method of building the classification based on information entropy.

3.3.1 Parent-children Words Group's Importance

The concept of entropy originates in physics through the second law of thermodynamics [4]. Information entropy [8, 9, 7] is a measure of how much information the answers for a specific question can provide. For a given question Q, if there are n possible answers for the question, and the probability of the occurrence of the ith answer is p_i, the information entropy of Q is defined in formula 1.

$$E(Q) = -\sum_{i=1}^{n} p_i \log_2(p_i) \tag{1}$$

Accordingly, supposing c is a word with n distinct children in the predefined ontology, and there are s components that can be represented by word c. Supposing $subc_i$ is the $ith(i = 1, 2, \ldots, n)$ child of c, and denote the relationship between the $subc_i$ and the jth component as $v_i Rc_j$, which shows that word $subc_i$ represents the jth component. Supposing the probability of each component being requested is $p_j (j = 1, 2, \ldots, s)$. Thus, the entropy of the parent-children words group (denoted as A) can be calculated using formula 2.

$$E(A) = -\sum_{i=1}^{n} \frac{\sum_{subc_i Rc_j} p_i}{\sum_{j=1}^{s} p_i} \log_2\left(\frac{\sum_{subc_i Rc_j} p_i}{\sum_{j=1}^{s} p_i}\right) \tag{2}$$

In this paper, we assume the probability of each component being the requested component is the same. Supposing the $ith(i = 1, 2, \ldots, n)$ child-word can represent C_i components, we have $\sum_{i=1}^{n} C_i = s$, the entropy $E(A)$ can be defined as formula 3.

$$E(A) = \log_2 s - \sum_{i=1}^{n} \frac{C_i}{s} \log_2 C_i \tag{3}$$

Table 1. Classification Generating Algorithm

```
Input: R, Ontolgoy
Output: the Classification
Step 1: Set Classification to be empty;
        Create a node N;
        Set N as the root of Classification;
Step 2: BuildSubNodes(R, Ontolgoy, N, Classification);
```

Table 2. The Algorithm of BuildSubNodes

```
Input: R, Ontology, N, and Classification
Step 1: $A' \leftarrow Ontology.Words$;
Step 2: Find the route r from the root of Classification to N;
Step 3: $CurrentComponents \leftarrow \emptyset$;
        For each element R[i] in R
          If all the relations in r are in Ontology
            $CurrentComponents \leftarrow CurrentComponents \cap \{i\}$;
Step 4: If CurrentComponets is empty
          Return;
Step 5: Find the most important word a in $A'$ using CurrentComponents;
        $A' \leftarrow A' - \{a\}$;
        $N.concept \leftarrow a$;
Step 6: If $A'$ is empty
          Return;
Step 7: Find the children of word a in Ontology, denoted as words;
        For each element c in words
        Begin
          Create a node $N'$;
          Link $N'$ to $N$ as a sub-node of $N$;
          BuildSubNodes(R, $A'$, Ontology, $N'$, Classification);
        End
```

From formula 3, we know that if a parent-children words group has more children and the distribution of the components under these children is less skewed, the more important the group is. When there is no component under the ith child-word (i.e. $C_i = 0$), we should assign the value 0 to $\frac{C_i}{s}\log_2 C_i$ in formula 3, otherwise formula 3 will be meaningless.

3.3.2 Classification Generating Algorithm

Supposing there are n components in the repository, R is a set of *(w, componentnum)* pairs, in which the $componentnum$ shows the number of the components represented by the word w. The algorithm of calculating the classification is depicted in Table1, and the algorithm of recursive procedure $BuildSubNodes$ is depicted in Table2.

4 EXPERIMENTAL STUDY

We selected 445 components from SourceForge, and used all the key words related to these components to set up an experimental repository. As in SourceForge, each component in this experimental repository is labeled with eight words: “Development Status”(DS), “Environment”(En), “Intended Audience”(IA), “License”(Li), “Natural Language”(NL), “Operating System”(OS), “Programming Language”(PL) and “Topic”. The ontology about the components in our experimental repository can be represented according to the ontology definition in 3.1, the following is part representation of this ontology. Based on this ontology and the words in the experimental repository, we generated a classification using the algorithm depicted in Table1.

$Ontology = \{Words, Relations, Axioms\}$

- $Concepts = \{DevelopmentStatus, Environment,$
  $IntendedAudience, License, NaturalLanguage,$
  $OperatingSystem, ProgrammingLanguage, Topic,$
  $OS, Desktop\_OS, Handheld\_OS,$
  $Workstations\_OS, Windows9.X, WindowsCE, OS/2,$
  $Provider, Microsoft, Apple, IBM, \ldots\}$
- $Relations = \{IsaKindof(Topic, System),$
  $IsaKindof(Topic, DataBase), \ldots,$
  $IsaKindof(DevelopmentStatus, 5-Production/Stable),$
  $IsaKindof(DevelopmentStatus, 6-Mature), \ldots,$
  $IsaKindof(Environment, Handhelds/PDA's),$
  $IsaKindof(Environment, WebEnvironment), \ldots,$
  $IsaKindof(IntendedAudience, InformationTechnology),$
  $IsaKindof(IntendedAudience, Developers), \ldots,$
  $IsaKindof(License, OSIApproved),$
  $IsaKindof(License, PublicDomain), \ldots,$
  $IsaKindof(NaturalLanguage, English),$
  $IsaKindof(NaturalLanguage, Spanish), \ldots,$
  $IsaKindof(ProgrammingLanguage, VisualBasic),$
  $IsaKindof(ProgrammingLanguage, Java), \ldots,$
  $IsaKindof(OperatingSystem, Desktop\_OS),$
  $IsaKindof(OperatingSystem, Handheld\_OS),$
  $IsaKindof(OperatingSystem, Workstations\_OS),$
  $IsaKindof(Desktop\_OS, MacOSX),$
  $IsaKindof(Desktop\_OS, Windows9.X),$
  $IsaKindof(Handheld\_OS; WindowsCE),$
  $IsaKindof(Workstations\_OS; Solaris),$
  $IsaKindof(Provider, Apple),$
  $IsaKindof(Provider, IBM),$
  $IsaKindof(Provider, Microsoft), \cdots\}$
- $Axioms = \{\exists r_i(c_p, c_q) \Longrightarrow \neg\exists r_i(c_q, c_p)\}$

The classification generated in this experimental study is depicted in Fig. 1. Due to the limited space, some nodes and/or edges of the classification are omitted and replaced by an ellipsis. As shown in Fig. 1, each node in the classification is labeled by a word; and each outgoing edge of a node is labeled by a relation between the parent-word and the child word. So every path from the root node to the leaf node represents a browsing sequence.

To evaluate the performance of our approach, we compare it with random generated classifications, which always randomly selects a relevant word in the algorithm of recursive procedure $BuildSubNodes$. We use all the components in the experimental repository to test the performance of our classification and that of random generated classifications. When locating each component in the experimental repository, the classification is used one time, while 100 different random generated classifications are used as comparison. The information of the browsing process for each component are recorded, including the number of browsing steps using our classification and the average number of browsing steps of the 100 random generated classifications. Based on the performance data of retrieving each component, we can compare the overall performance of these two approaches. In this experiment, once the number of retrieved components is under the threshold 5, we will regard the retrieval is completed or the component is located. The results of experimental study are presented in the following figures.

Fig. 2 demonstrate the relationships between the number of browsing steps and the number of components which can be retrieved in each step. It shows that, guided by our classification, the retriever can find the same desired components using shorter retrieval sequences, then have fewer numbers of browsing steps. Fig. 3 shows the relationships between the number of retrieval steps and the average number of components resulted in each browsing step. It shows that, guided by our classification, fewer components can be returned in the previous step, also the number of returned components can be confined as much as possible in each of the retrieval steps, and then the retriever can pay more attention to fewer components in each retrieval step. In conclusion, the experimental study support our claim that the classification generated by our method can navigate the browsing-based component retrieval efficiently.

5 ACKNOWLEDGMENTS

This research was sponsored by the National Grand Fundamental Research 973 Program of China under Grant No. 2005CB321805, the National Key Technology R&D Program of China under Grant No.2006BAH02A02, and the National Natural Science Foundation of China under Grant No.60473059,60473064 and 90412011.

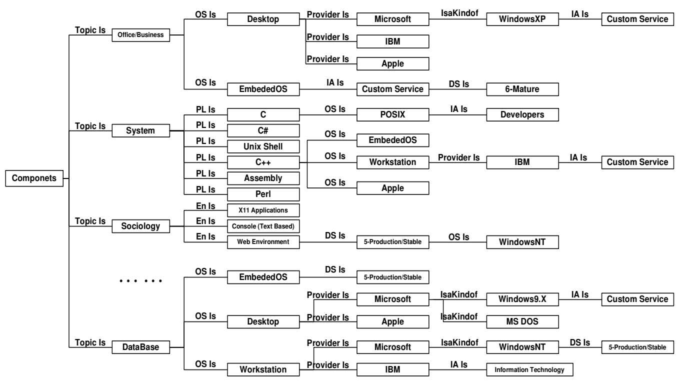


Figure 1. Generated Classification

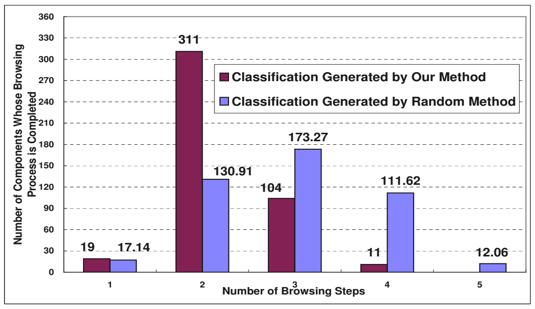


Figure 2. Browsing Steps Comparison

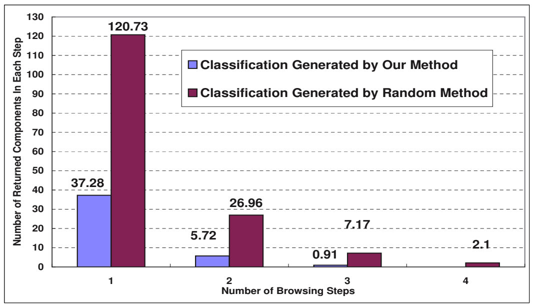


Figure 3. Components Numbers Comparison

References

[1] I. Bratko. *PROLOG Programming for Artificial Intelligence*. Pearson Education Limited, third edtion edition, 2000.

[2] B. Fischer. Specification-based browsing of software component libraries. *Journal of Automated Software Engineer*, 7(2):179–200, 2000.

[3] M. Hertzum and E. Frokjaer. Browsing and querying in online documentation: A study of user interfaces and theinteraction process. *ACM Transaction on Computer-Human Interaction*, 3(2):136–161, 1996.

[4] J. C. Maxwell. Theory of heat. 2001.

[5] H. Mili, E. AhKi, R. Godin, and H. Mcheick. An experiment in software component retrieval. *Information and Software Technology*, 45(10):633–649, 2003.

[6] H. Mili, F. Mili, and A. Mili. Reusing software: Issues and research directions. *IEEE Transactions on Software Engineering*, 21(6):528–562, 1995.

[7] A. Renyi. *Probability theory*. Springer Verlag, 1970.

[8] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[9] R. C. Tolman. *The principles of statistical mechanics.* Dover, 1979.

A Context-Dependent Semantic Distance Measure

Ahmad El Sayed, Hakim Hacid, Djamel Zighed
ERIC laboratory – University of Lyon 2
{asayed, hhacid, dzighed}@eric.univ-lyon2.fr

Abstract

A major lack in the existing semantic similarity measures is that they don't take into account the context or the considered domain. However, two concepts similar in one context may appear completely unrelated in another context. We propose a Context-Dependent method that takes the taxonomy as a principal knowledge resource and a text corpus as a similarity adapter pattern to the target context. Experiments have shown a very interesting correlation ratio of 86% with human similarity ratings.

1 Introduction

The end-goal of any computational model is to achieve a certain degree of "intelligence" that makes it comparable to humans intentions over objects. Thats obviously a hard task especially that two objects sharing any attribute(s) in common may be related by some abstract human-made relation. The same problem remains for text. Beyond managing synonymy and polysemy, many applications need to measure the degree of semantic similarity between 2 words; lets mention: Information retrieval, question answering, automatic text summarization and translation, etc...Taxonomies are widely used for that purpose. Their success came essentially from the fact that they represent the ultimate tool to reflect human intention over instances since most of them are an expert-handicraft resources.

On the other hand, a major lack in the existing semantic similarity methods is that no one takes into account the context or the domain under study. However, two concepts similar in one context may appear completely unrelated in another context. A simple example for that: While *blood* and *heart* seem to be very similar in a general context, they represent two widely separated concepts in a domain specific context like medicine.

We propose a taxonomy-based method that depends essentially on the target context detected from a text corpus. We suggest also to combine it with a feature-based measure in an attempt to take profit from the whole 'knowledge package'.

Section 2 provides a quick overview on the existing taxonomy-based semantic similarity measures. In section 3, we describe our context-dependent approach. Experimental results are shown in section 4 and then compared to other results in section 5. We conclude in section 6.

2 Semantic Similarity Between Concepts

A number of successful projects in computational linguistic have led to the development of some widely used taxonomies like Wordnet[1] (generic taxonomy) and Mesh[1] (for the medical domain). Since their 'semantic' structure, these taxonomies offer a reliable way to calculate semantic similarities between concepts[2]. Taxonomy-based measures can be grouped into edge-based measures and node-based measures.

2.1 Edge-based Measures

First, calculating similarities simply relied on counting the number of edges separating two nodes by an *'is-a'* relation [2]. While this approach can work well in a domain specific taxonomy, its limitations was clear since it can give results such *'person/animal'* closer than *'cat/dog'*. Since specific concepts may appear more similar than abstract ones, the depth was taken into account by calculating either the maximum depth in the taxonomy [3] or the depth of the most specific concept subsuming the two compared concepts [4]. Hirst [5] considers that two concepts are semantically similar if they are connected by a path that is not too long and that does not change direction too often. The problem with edge-based measures is that they consider the taxonomy as a simple structure with uniform distances.

[1]http://www.nlm.nih.gov/mesh/

[2]In this article, we'll use the term concept instead of words since we consider that taxonomy-based measures deal practically with the concept associated to a word, and not the word itself.

2.2 Node-based Measures

These measures came to overcome the unreliability of edge distances and based its similarities on the information associated with each node. This information can be either a node description (Feature based measures) or a numerical value augmented from a text corpus (Information content measures):

2.2.1 Feature based measures

A part of their simple conceptual structure, taxonomies like Wordnet provide users with additional resources which describe most entities. Information in Wordnet is organized around logical groupings called synsets. Each synset is attached to a description set, a list of synonyms, antonyms, etc...[6] proposes a measure that considers that the more common characteristics two objects have and the less non common characteristics they have, the more similar the objects are. This measure, if applied on rich resources like Wordnet, can provide very good results:

$$sim(c1, c2) = \frac{|C1 \bigcap C2|}{|C1 \bigcap C2| + k\,|C1/C2| + (k-1)\,|C2/C1|} \tag{1}$$

2.2.2 Information Content measures

The Information content (IC) approach was first proposed by Resnik[7]. It considers that the similarity between two concepts is "the extent to which they share information in common".

Information content for a concept c is actually calculated by the probability $p(c)$ of encountering an instance of c in a corpus C. An instance of c could be a word w representing c or any subconcept of c [3]. Then, IC is calculated by the log likelihood : $IC(c) = -\log p(c)$

Thus, as frequency increases, informativeness decreases, so the more abstract a concept, the lower its information content. After assigning IC values for each concept, Resnik defines the similarity between two concepts as the IC value of their Most Informative Subsumer (MIS). Thus, the more informative parent two concepts share in common, the more similar they are. By using this theory, Resnik have outscored the edge-based methods by attaining a success rate of 79%[4].

Nevertheless, this method have shown some imprecisions. In practice, it would be enough for two couple of concepts to share the same MIS, to obtain the same similarity, which is irrelevant in many cases. The Jiang[8] model came next in order to overcome the Resnik limitations. It is is derived from the edge-based notion by adding the information content as a decision factor. [8] assigns a link strength (LS) for each *is-a* link in the taxonomy. An LS is simply the difference between the IC values of two nodes. LS takes into account another factors like the local density and the node depth. Then, the distance between two concepts is obtained by summing the edges weights along the shortest path linking two nodes. Jiang has reached a success rate of 84.4% which led it to outperform all the other taxonomy-based measures. Another effective measure was proposed by Lin[9] which reached a 82.1% of precision. As Jiang, Lin doesn't only consider the IC of the most informative subsumer but the IC of the compared concepts too.

[3]In rich taxonomies like Wordnet a concept is represented by a set of words.

[4]The success rate is evaluated by means of correlation with human ratings on a well-defined set of words pairs

3 A Context-Dependent Approach

3.1 Context: a Decisive Factor for Similarity

Context definition varies from one research area to another. Similarity judgments are made with respect to representations of entities, not with respect to entities themselves [10]. Thus, having a changeable representation, one can make any two items similar according to some criteria. To prevent this, a context may be used in order to focus the similarity assessment on a certain features of the representation excluding irrelevant information. Barsalou[11] presents a nice example supporting the context-dependency; it explains the instability of similarity judgments.

In text similarity, comparing two words/terms doesn't make any sense if we ignore the actual context. Let's take the example of *heart* and *blood*. In a general context, these two concepts can be judged to be very similar. However, if we put ourselves in a medical context, *heart* and *blood* define two largely separated concepts. They will be even more distant if the context is more specifically related to *anatomy*.

Our context-dependent approach suggest to adapt semantic distances[5] to the target corpus since it's the entity representing the context in the target applications. It is inspired from the information content theory [7] and from the Jiang[8] measure seen in the above section.

[5]Our method deals with distance which is the inverse of similarity.

3.2 Problems with Information Content Measures

As we've seen earlier, IC measures are mainly based on the concept frequency in a text corpus. According to the measure's purpose, we can show two main difficulties for that approach:

On concept informativeness We believe that it's imprecise to consider infrequent concepts as more informative than frequent ones. We follow Nuno's [12] point of view assuming that the taxonomic structure in WordNet is organized in an enough meaningful way to measure IC. We can simply say that the more hyponyms a concept has the less information it expresses. Nuno have shown that at least similar results can be obtained without using a text corpus.

On context-dependency If the motivation behind measuring the IC from a text corpus is to consider the actual context, we argue that the probability of encountering a concept in a corpus is not an enough adaptive way to determine whether it's representative for a given context. Thus, IC cannot significantly reflect the target context.

3.3 A New Context-Dependency Based Measure

Our approach tends to compute semantic distance[6]. In order to represent the context, we assign weights for taxonomy's concepts according to their Context-Dependency CD to a corpus C. The goal is to obtain a weighted taxonomy, where 'heavier' subtrees are more context representative than 'lighter' subtrees.

As we said earlier, it's clear that a concept's frequency alone is not enough to determine its context-dependency. A very frequent concept in some few documents and absent in many others cannot be considered as "well" representative for the corpus. Thus, the number of documents where the concept occurs is another important factor that must be considered. In addition to that, frequency distribution for a concept over a set of documents can be another important factor for dependency decisions. Suppose we have 2 concepts c_1 and c_2 sharing the same frequency over a corpus C (20 occurrences for instance) and present in the same number of documents (5 documents). Let the frequency of c_1 over the 5 documents be perfectly distributed where 4 occurrences are found in each document $(4-4-4-4-4)$; and let the frequency of c_2 have this distribution $(2-5-9-3-1)$. It's most likely that c_2 -with its heterogeneous distribution among documents - is more discriminative than c_1, whose monotone repartition can reveal less power of discrimination over the target domain (Experimentations made assess our thesis).

[6]We consider distance by its dissimilarity which is the inverse of similarity

Consequently, we introduce our CD measure which is an adapted version of the standard $tf.idf$. Given a concept c , $CD(c)$ is a function of its total frequency $freq(c)$, the number of documents containing it $d(c)$, and the variance of its frequency distribution $var(c)$ over documents in C :

$$CD(c) = \frac{log(1+freq(c))}{log(N)} . \frac{log(1+d(c))}{log(D)} .(1+log(1+var(c))) \tag{2}$$

Where N and D denote respectively the total number of concepts in C and the total number of documents in C. The log likelihood seems adaptive to such purpose as it provides understandable normalized values by reducing big margins. This formula ensures that if a concept has a 0 frequency, its CD will equals 0 too. It ensures also that if c has an instance in C, its CD will never be 0 even if $var(c) = 0$.

Note that the CD of a concept c is the sum of its individual CD value with the CD of all its subconcepts in the taxonomy. The weights propagation from the bottom to the top of the hierarchy is a natural way to ensure that a concept even with a low individual CD will be considered as highly context-dependent if its children are well represented in the corpus(see Figure 1). In other terms, if a corpus C is more specialized in domain D_1 than D_2, we'll most likely encounter more specific concepts related to D_1 in text than to D_2; thus, the subtree S_1 representing D_1 will be more weighted than S_2 representing D_2.

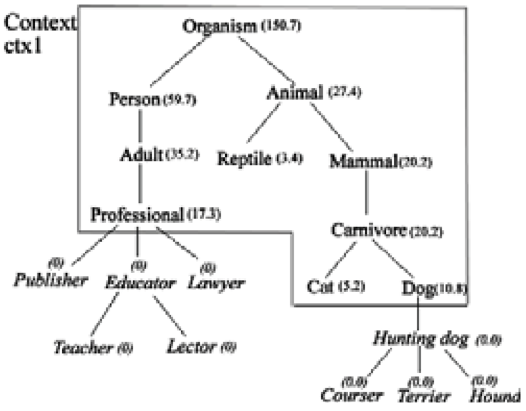


Figure 1: A taxonomy extract showing CD values assigned in the context $ctx1$

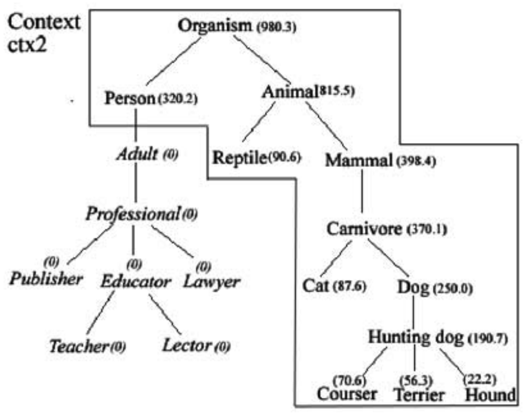


Figure 2: A taxonomy extract showing CD values assigned in the context $ctx2$

That is, to compare two concepts using the CD values, we assign a Link Strength (LS) to each 'is-a' link in the taxonomy. Assume that c_1 subsumes c_2, the LS between c_1 and c_2 is then calculated as follows:

$$LS(c_1, c_2) = CD(c_1) - CD(c_2)$$

Then our Context-Dependent Semantic Distance is calculated by summing the log likelihood of LS along the shortest path separating the two concepts in the taxonomy.

$$Dist(c_1, c_2) = \sum_{c \in SPath(c_1,c_2)} log(1 + LS(c, parent(c)))$$

Where $SPath$ denotes the shortest path between c_1 and c_2.

The method is best illustrated with an example. Consider the taxonomy extract shown in Figure 1. The related context $ctx1$ for a corpus C_1 is represented by the subtree where CD values are greater than 0. In $ctx1$, we can notice that the corpus is likely 'general' (talking about persons, professionals, carnivores,etc.). Let's consider the semantic distance between Cat and Dog in $ctx1$ given the previous formula:

$$Dist_{ctx1}(Cat, Dog) = 2.7 + 2.2 = 4.9$$

Now, consider the context $ctx2$ illustrated in Figure 2 where it seems to be more specialized in animals. Let's calculate the distance between the same two concepts:

$$Dist_{ctx2}(Cat, Dog) = 5.6 + 4.8 = 10.4$$

This states that Cat and Dog are closer in $ctx1$ than in $ctx2$ which respect human intuitions given the two different contexts.

4 Evaluation and Results

4.1 Implementation

In this study, Wordnet is used as the main semantic knowledge base for evaluating the distances. A text corpus have the role to adapt distances to its target context. We've built a corpus of 30,000 web pages using a crawler found on the net. The web pages were mostly taken from News web sites (reuters.com, cnn.com, nytimes.com...). The corpus was then analyzed using Natural Language Processing (NLP) techniques along with WordNet to detect concepts. Evaluation is done next on the basis of the computed CD values.

4.2 The Benchmark

The most intuitive way to evaluate a semantic similarity/distance is to compare machine ratings and human ratings on a same data set. A very common set of 30 word pairs is given by M&C [13]. M&C asked 38 undergraduate students to rate each pair on a scale from 0 (no similarity) to 4 (perfect synonymy). The average rating of each pair represents a good estimate on how similar the two words are. The correlation between individual rating of human replication was 0.90 which led many researchers to take 0.90 as the upper bound ratio. For our evaluations, we've chosen a M&C subset of 28 words pairs which is the most commonly used subset for that purpose. Note that since our measure calculates distance, the M&C distance will be: $dist = 4 - sim$ where 4 represents the maximum degree of similarity.

4.3 Evaluation Strategies

When comparing our distance results with the R&C human ratings, the context-dependency CD method gave a correlation of 0.83 which seems to be a very promising rate (See table 2).

In view of further improvements, we investigate to combine our context-dependent method with a feature-based measure. The lexical taxonomy, till now, is used basically as a semantic network with 'is-a' relations. It's obvious that by ignoring all the other information(description, synonym, antonyms, meronyms, holonyms...), we're not fully exploiting the knowledge resource and we're missing a reliable source that can improve our measure. That's why, we study the utilization of concept's features as a complementary measure. Stop words are eliminated first, then the number of common words is computed using the Tversky formula 1. The distance will be : $dist = 1 - sim$ since Tversky's similarity interval is [0,1].

With our implementation, the feature-based measure used alone gave us a correlation rate of 0.619 over the data set. In order to combine it with the CD measure, we've tried three non linear strategies:

S1: $Dist = CD.Feat$ Gave a correlation of 0.83. No amelioration is noticed in the CD rate.

S2: $Dist = log(1 + CD).log(1 + Feat)$ We obtain a considerable amelioration with a correlation of 0.868.

S3: $Dist = Feat.\sqrt{CD}$ A comparable ratio with the previous strategy is obtained with a correlation of 0.867.

5 Comparison and Discussion

Our method show an interesting results whether on an individual or on a combination scale (See table 1). A part of its interesting correlation coefficient of 0.83, our CD method has the advantage to be context-dependent, which means that our results varies from one context to another. We argue that our values could perform better if we 'place' human subjects in our corpus context. In other terms, our actual semantic distance values reflect a specific context that don't necessarily match with the context of the human subjects during the R&C experiments.

By considering the different aspects of the taxonomy (its hierarchical structure and its nodes information), an optimal correlation was reached with a rate of 0.868 which is not too far from human correlations of 0.901.

6 Conclusion and Future Work

We have shown the importance of considering the context when calculating semantic similarities between words/concepts. We've proposed a Context-Dependent method that takes the taxonomy as a principal knowledge resource and a text corpus as a similarity adapter pattern to the target context. The results obtained from the experiments show the effectiveness of our approach. A much better way to evaluate the method and compare it with others is to perform a context-driven human ratings, where human subjects will be asked to rank a same set of words pairs in different contexts. The machine correlation computed next according to each context will be able to show more significantly the added-value of our approach.

References

[1] G. A. Miller, "Wordnet: A lexical database for english." *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[2] R. Rada, H. Mili, E. Bicknell, and M. Blettner, "Development and application of a metric on semantic nets," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, no. 1, pp. 17–30, 1989.

[3] C. Leacock, M. Chodorow, and G. A. Miller, "Using corpus statistics and wordnet relations for sense identification," *Computational Linguistics*, vol. 24, no. 1, pp. 147–165, 1998.

[4] Z. Wu and M. Palmer, "Verb semantics and lexical selection," in *32nd. Annual Meeting of the Association for Computational Linguistics*, New Mexico State University, Las Cruces, New Mexico, 1994, pp. 133 –138.

[5] G. Hirst and D. St-Onge, "Lexical chains as representation of context for the detection and correction malapropisms," 1997.

[6] A. Tversky, "Features of similarity." *Psychological Review*, vol. 84, pp. 327–352, 1977.

[7] P. Resnik, "Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language." *J. Artif. Intell. Res. (JAIR)*, vol. 11, pp. 95–130, 1999.

[8] J. J. Jiang and D. W. Conrath, "Semantic similarity based on corpus statistics and lexical taxonomy," 1997.

[9] D. Lin, "An information-theoretic definition of similarity," in *Proc. 15th International Conf. on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 1998, pp. 296–304.

[10] D. Medin, *Psychological essentialism.* Cambridge University Press, 1989.

[11] L. Barsalou, *Intraconcept similarity and its application for interconcept similarity.* Cambridge University Press, 1989.

[12] N. S. And, "An intrinsic information content metric for semantic similarity in wordnet."

[13] G. A. Miller and W. Charles, *Contextual Correlated of Semantic Similarity*, 1991, vol. 6.

Similarity method	Type	Correlation with M&C
Human replication	Human	0.901
Rada	Edge-based	0.59
Hirst and St-Onge	Edge-based	0.744
Leacock and Chodorow	Edge-based	0.816
Resnik	Information Content	0.774
Jiang	Information Content	0.848
Lin	Information Content	0.821
CD	Context-Dependent	0.830
Combined(CD + Feat)	Combination	0.868

Table 1: Comparison between the principal measures and their correlation with M&C human ratings

Word Pair	M&C	CD	Feat	S1	S2	S3
car-automobile	0,08	1	0,52	0,52	0,29	0,52
gem-jewel	0,16	1	0,768	0,768	0,395	0,768
journey-voyage	0,16	3,783	0,847	3,203	0,96	1,647
boy-lad	0,24	1,635	0,81	1,325	0,575	1,036
coast-shore	0,3	1,426	0,862	1,229	0,551	1,03
magician-wizard	0,5	1	0,768	0,768	0,395	0,768
midday-noon	0,58	1	0,554	0,554	0,305	0,554
furnace-stove	0,89	14,182	0,886	14,182	1,885	3,766
food-fruit	0,92	8,489	1	8,489	1,56	2,914
bird-cock	0,95	3,606	0,858	3,094	0,946	1,629
bird-crane	1,03	4,286	0,86	3,687	1,034	1,781
tool-implement	1,05	2,01	0,85	1,708	0,678	1,205
brother-monk	1,18	1,473	0,905	1,333	0,583	1,098
crane-implement	2,32	8,982	1	8,982	1,595	2,997
lad-brother	2,34	12,745	0,842	12,643	1,694	3,262
journey-car	2,84	25,653	1	25,653	2,276	5,065
monk-oracle	2,9	13,64	1	13,64	1,86	3,693
food-rooster	3,11	13,53	1	13,53	1,855	3,678
coast-hill	3,13	7,24	1	7,24	1,462	2,691
forest-graveyard	3,16	21,004	0,902	18,939	1,987	4,133
shore-woodland	3,37	15,095	0,903	13,635	1,788	3,509
monk-slave	3,45	11,302	1	11,302	1,74	3,362
coast-forest	3,58	14,736	0,898	13,235	1,766	3,448
lad-wizard	3,58	11,853	1	11,853	1,77	3,443
chord-smile	3,87	15,701	1	15,701	1,952	3,963
glass-magician	3,89	17,276	1	17,276	2,014	4,156
noon-string	3,92	16,53	1	16,53	1,985	4,066
rooster-voyage	3,92	24,853	1	24,853	2,254	4,985
Correlation	**0.905**	**0.830**	**0.619**	**0.830**	**0.868**	**0,867**

Table 2: Distances Results from the different strategies and their correlation to M&C Means.

A Semantical Change Detection Algorithm for XML

Rodrigo Cordeiro dos Santos
Universidade Federal do Parana, Brazil
rodrigosantos@celepar.pr.gov.br

Carmem Hara
Universidade Federal do Parana, Brazil
carmem@inf.ufpr.br

Abstract

XML diff algorithms proposed in the literature have focused on the structural analysis of the document. When XML is used for data exchange, or when different versions of a document are downloaded periodically, a matching process based on keys defined on the document can generate more meaningful results. In this paper, we use XML keys defined in [5] to improve the quality of diff algorithms. That is, XML keys determine which elements in different versions refer to the same entity in the real world, and therefore should be matched by the diff algorithm. We present an algorithm that extends an existing diff algorithm with a preprocessing phase for pairing elements based on keys.

1. Introduction

XML has become the standard format for data exchange on the Web. It helps the process of publishing data, especially when the underlying information is constantly being updated. The data consumer, on the other side, may be interested not only on the current contents of a web site, but also on the updates that have been made since his last access. As an example, one may want to know what are the new products in a catalog, or which products had their prices changed. To help the task of comparing two versions of XML documents, a number of diff algorithms have been proposed in the literature [1, 2, 15, 16].

The majority of these algorithms are based on a structural analysis of the documents. Similar to diff algorithms on strings, their main strategy is based on finding large fragments of data that are identical in both versions of a document and match them. After finding these matchings, a sequence of operations that transforms the old version of the document to the new one is generated. This is called an edit script or delta. In many applications, XML documents are not arbitrary tree structured data, but have well defined structure and semantics. A strategy based solely on structural and value similarities can generate erroneous matchings of elements.

We have conducted an experimental study to analyze the results of two diff algorithms: XyDiff[1], and X-Diff[15]. The experiments consisted of modifying an XML tree by inserting, deleting, modifying, and moving both internal and leaf nodes in the tree, and then analyzing how semantically meaningful the changes detected by these algorithms were. The results showed the following: 1) both algorithms are extremely sensitive to changes in the structure of the document, especially when they involve insertion and removal of internal nodes; 2) the existence of several similar or identical subtrees in a document may induce the algorithms to erroneously match them, and these erroneous matchings are propagated to their ancestors and descendants. We illustrate these problems below.

Example 1: Consider the two XML documents, represented as XML trees, depicted in Figure 1. They contain information about university professors. Each `professor` has a `name`, an `office`, and optionally a `phone` or `email`. Comparing the old version with the new one, we can observe that both `John` and `Mary` have their offices changed, and moreover, that `Mary` has moved to `John`'s former office. If the strategy of the diff algorithm is to find the largest common subtree in both versions, it will match `John`'s old office (node 9) with `Mary`'s new one (node 49). This matching can then be propagated upwards by matching `professor` elements (nodes 4 and 45). That is, the `professor` element node that corresponds to `John` is matched with the one that corresponds to `Mary`. As a consequence, the edit script contains an update on the `name` of the `professor` who works in the matched office, while our expected result is an update on `professor`'s office. In particular, if we know that `name` uniquely identifies a `professor` in the document, that is, `name` is a key for `professor`, then it is always the case that matches of `professor` elements based on their names produce more meaningful results than matches based on similar subtrees.

Observe that not only values have been modified, but the structure of the document has also changed. In the old version, professors are organized by their universities, while in the new one, they are placed under their departments. In our experimental study, the edit scripts generated by both XyDiff and X-Diff consisted of operations that remove all subtrees rooted at `professor` elements, followed by in-

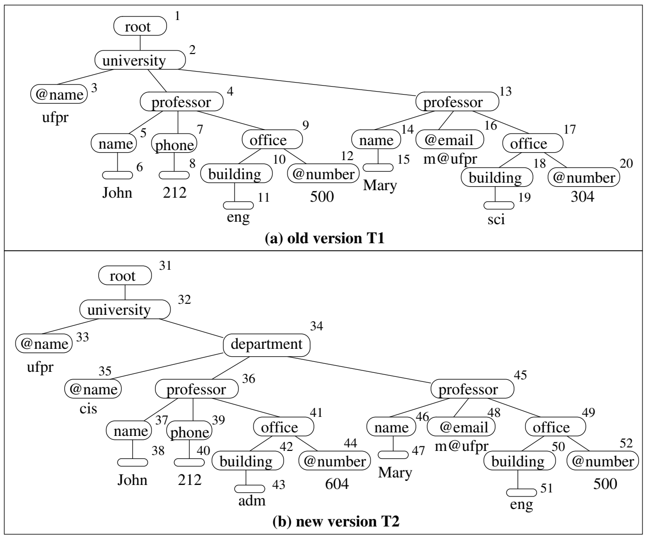


Figure 1. Two versions of XML trees

sertions of the same subtrees under the new `department` node. This is because in both algorithms only nodes reached by following exactly the same path from the root can be matched. Clearly, this is not the expected results from the semantical point of view, which is the creation of a new level in the tree. Similar to the previous case, if we know that `name` uniquely identifies a `professor` *no matter where the* `professor` *node occurs in the tree* then the correct matchings of `professor` elements would be found, avoiding their removals in the edit script. □

In this paper, we propose an algorithm, called XKeyMatch, that uses XML keys[5] to guide the comparison of XML documents. That is, first elements in the two versions are matched based on their key values, and then a structural analysis is performed to determine their differences. In Example 1 we would give as input to the diff algorithm that `name` uniquely identifies a `professor` in the entire document. A strategy based on keys is natural when comparing two relational databases. Since XML has become a standard for data exchange, it is natural to apply the same strategy for this format.

One area in which matchings based on semantics is especially important is data cleansing[11]. One of the main goals of data cleansing is to detect inconsistencies on input data. As an example, consider a datawarehouse that maintains data imported from external sources. Periodically these external sources update and publish new versions of their database. In order to keep the datawarehouse up-to-date, it is necessary to determine what are the differences of the new version compared to the previous one. If the previous version had been through data cleansing, one would like to know if previously detected mistakes have been corrected, either to prevent redoing the cleansing process, or to report the error to the external source. Ideally, a diff algorithm for helping this task should first identify which pieces in the two versions of imported data refer to the same entity and then determine what has been changed.

Contributions. The main contributions of this paper are:

- A proposal of a semantical approach in the context of diff algorithms for XML;
- An algorithm for matching elements in two versions of XML documents based on XML keys.

To the best of our knowledge this is the first algorithm that introduces keys in the context of XML diff algorithms, and that generates semantically meaningful results when there are changes in the structure of the document.

Related Work. A number of diff algorithms have been proposed in the literature both for text documents[8] and for tree structures[13, 14, 17]. XyDiff[1] is one of the earliest algorithms proposed for XML. It was designed for datawarehouses that store huge amounts of data, and therefore it was designed to be efficient, both in time and space. The algorithm is based on an ordered tree model. When the documents are accompanied with a DTD[4], attributes defined as identifiers (ID) are used to match elements according to their values. X-Diff[15] is an algorithm based on an unordered tree model. The distinguishing feature of diffX algorithm[2] is that it looks for matches in isolated fragments of the XML tree, instead of pairing nodes along a tree traversal. A different strategy for finding matches is applied by KF-Diff[16]. It is based on defining unique paths starting from the root for each node in the tree. Whenever such a path cannot be found, which happens when a node has more than one child with the same label, these labels are replaced by *key fields*. Key fields are values contained in the subtrees rooted at these nodes that can distinguish them among those with the same label. Although the idea of key fields is used in this algorithm, it is applied as a technique for deriving unique paths. They are not defined by the user, and therefore are not meant to capture the semantics of the document. Comparative studies of existing XML diff algorithms can be found in [7] and [12].

Organization. The rest of the paper is organized as follows. Section 2 defines XML keys, and presents definitions related to diff algorithms. Section 3 describes our proposal of a semantical diff algorithm, followed by our conclusions in Section 4.

2. XML Keys and Diff Algorithms

This section presents some definitions used throughout the paper.

Tree model and XML keys. XML documents can be modeled as trees. Nodes in the tree can be of three types: element, attribute, and text, where attribute labels are prefixed by “@”. Based on node types, we define function $lab(n)$, and $val(n)$ as follows: if n is an element node then $lab(n)$

denotes the name of the element and $val(n)$ is undefined; if n is an attribute node then $lab(n)$ denotes the name of the attribute and $val(n)$ is its associated string value; if n is a text node then $lab(n) =$ "S", and $val(n)$ is its string value. Two examples of XML trees are illustrated in Figure 1.

To define a key we specify three things: 1) the *context* in which the key must hold; 2) a *target* set on which we are defining a key; and 3) the *values* which distinguish each element of the target set. For example, the key specification of Example 1 has a context of the `root` (the entire document), a target set of `professor`, and a single key value, `name`. Specifying the context node and target set involve path expressions.

The path language we adopt is a common fragment of regular expressions [10] and XPath [6]:

$$Q \quad ::= \quad \epsilon \quad | \quad l \quad | \quad Q/Q \quad | \quad //$$

where ϵ is the empty path, l is a node label, "/" denotes concatenation of two path expressions (*child* in XPath), and "//" means *descendant-or-self* in XPath. To avoid confusion we write $P//Q$ for the concatenation of P, $//$ and Q.

Following the syntax of [5] we write an XML key as:

$$K: \ (C, (T, \{P_1, \ldots, P_p\}))$$

where K is the name of the key, path expressions C and T are the context and target path expressions respectively, and $P_1, ..., P_p$ are the key paths. For the purposes of this paper, we restrict the key paths to be simple paths (without "//"). A key is said to be *absolute* if the context path C is the empty path ϵ, and *relative* otherwise.

Example 2: Using this syntax, some constraints on XML trees in Figure 1 can be written as follows:

- $k_1 : (\epsilon, (university, \{@name\}))$: within the context of the entire document (ϵ denotes the root), a `university` is identified by its `name`.
- $k_2 : (university, (//professor, \{name, phone\}))$: within the context of each subtree rooted at a `university` element, a `professor` is uniquely identified by the values of its subelements `name` and `phone`. The `professor` can appear anywhere in the subtree rooted at `university`.

□

To define the meaning of an XML key, we use the following notation: in an XML document (tree), $n[\![P]\!]$ denotes the set of node identifiers that can be reached by following path expression P from the node with identifier n. $[\![P]\!]$ is an abbreviation for $r[\![P]\!]$, where r is the root node of the tree. As an example, in Fig. 1(a), $[\![university]\!] = \{2\}$, $2[\![professor]\!] = \{4, 13\}$ and $[\![//name]\!] = \{5, 14\}$.

The formal definition of the meaning of an XML key is given next.

Definition 1 *An XML tree satisfies an XML key* `(Q, (Q', {`$P_1, \ldots, P_n$`}))` *iff for any n in $[\![Q]\!]$, and any n_1 and n_2 in $n[\![Q']\!]$, if for all* `i`, $1 \le$ `i` $\le n$ *there exists z_1 in $n_1[\![P_i]\!]$ and z_2 in $n_2[\![P_i]\!]$ such that $z_1 =_v z_2$, then $n_1 = n_2$.*

The definition above involves value equality on trees ($=_v$), so we formalize this notion below.

Definition 2 *Given an XML tree T, and two nodes n_1 and n_2 in T, we say that they are value equal, denoted as $n_1 =_v n_2$ iff the following conditions are satisfied: 1)* `lab(`n_1`) = lab(`n_2`)`*; 2) if n_1 and n_2 are attribute or text nodes then* `val(`n_1`) = val(`n_2`)`*; 3) if n_1 and n_2 are element nodes then: an attribute A is defined for n_1 iff it is also defined for n_2, and $val(n_1.A) = val(n_2.A)$; moreover, if* `[`$d_1, \ldots, d_k$`]` *are subelements of n_1 then n_2 has subelements* `[`$d'_1, \ldots, d'_k$`]`*, and for all $i \in [1, k]$ there exists $j \in [1, k]$ such that $d_i =_v d'_j$.*

For example, XML tree of Fig. 1(a) satisfies key $(//professor, \{name\})$ since $[\![//professor]\!] = \{4, 13\}$, and $4[\![name]\!] \neq_v 13[\![name]\!]$.

Diff Algorithms. The essential goal of a diff algorithm is to find an edit script such that given a version of a document in time $t - 1$ and the edit script, it is possible to obtain its version in time t. The number of operations in a edit script is called the *edit distance*. Although the edit distance is often used to define the cost of the transformation, finding a minimum edit script is not always the best strategy for generating semantically meaningful results. In the next Section, we propose using XML keys to guide the node matching process of a diff algorithm.

3. A Semantical Diff Algorithm

In this section we present XKeyMatch, an algorithm for matching elements in two versions of XML trees based on XML keys. This algorithm is designed to be executed before a diff algorithm that compares the contents in both versions. The architecture of the system is depicted in Figure 2.

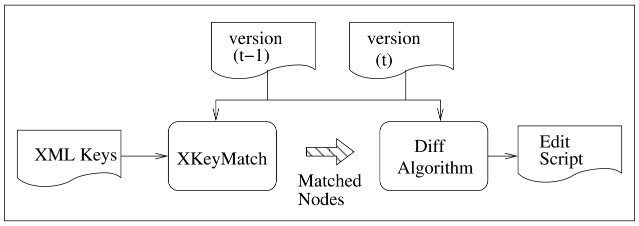


Figure 2. A diff algorithm with a preprocessing phase with XKeyMatch

Algorithm XKeyMatch receives as input two versions, v_{t-1} and v_t, of XML trees, and a set of XML keys Σ that are known to be satisfied by both versions. The output is a set Γ of node pairs $[n_1, n_2]$, where n_1 is a node in v_{t-1} that refers to the same entity as node n_2 in v_t according to Σ. The set Γ is then given as input to a diff algorithm that, based on these matchings, compares versions v_{t-1} and v_t and generates an edit script.

Algorithm XKeyMatch is based on the construction of a deterministic finite automaton (DFA) from the set Σ of XML keys, denoted as KeyDFA(Σ). Using this DFA, it is possible to process all keys in Σ at the same time, and all matchings based on Σ can be generated with a single pre-order traversal on v_{t-1} and v_t. More specifically, each state of the DFA represents a set of paths, and it stores information on all keys that can be defined on nodes reached by following these paths. Therefore, each step of the XML tree traversal corresponds to a change of state in the automaton; information on keys stored at each state is used to collect nodes that are candidates for matchings based on these keys.

After collecting all candidates, algorithm XKeyMatch pairs nodes in both versions according to their key values. These steps of the algorithm are depicted in Figure 3. Given XML trees T_1 and T_2, and a set of XML keys Σ, the algorithm first builds KeyDFA(Σ) (Line 1). Then candidates are selected by calling function `get_candidates` for both trees (Lines 2 and 3). The resulting set of matches is computed by function `match`, which compares the candidates previously collected (Line 4). Next, we present each of these steps in detail.

```
Function XKeyMatch
Input: XML trees $T_1, T_2$, and a set of XML keys $\Sigma$
Output: a set of matched node pairs
1. KeyDFA:= DFA($\Sigma$);
2. candidates($T_1$):= get_candidates($T_1$, KeyDFA);
3. candidates($T_2$):= get_candidates($T_2$, KeyDFA);
4. return (match(candidates($T_1$), candidates($T_2$), $T_1$, $T_2$);
```

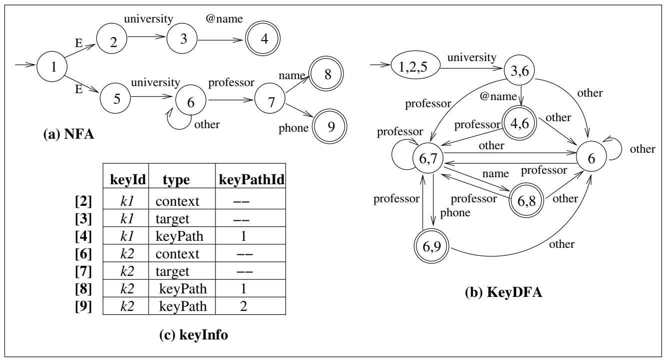
Figure 3. Algorithm XKeyMatch

DFA Construction. Given a set Σ of XML keys, this step of algorithm XKeyMatch generates a deterministic finite automaton, denoted as KeyDFA(Σ), where each state stores information for processing every key in Σ with a single traversal on an XML tree T.

Let $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$, where each σ_i is of the form $(Q_i, (Q'_i, \{P_i^1, \ldots, P_i^{n_i}\}))$. We first describe the construction of a non-deterministic finite state automaton (NFA) associated with each key σ_i in Σ. We start with the construction of a NFA for each path p in $\{Q_i, Q'_i, P_i^1, \ldots, P_i^{n_i}\}$, defined as $M(p) = (N_p, L_p \cup \{other\}, \delta_p, S_p, F_p)$, where N_p is a set of states, L_p is the alphabet, δ_p is the transition function, S_p is the start state, and F_p is the set of final states. Here, "*other*" is a special character that can match any character. These automaton have "linear structure"; that is, if $p = l_1/\ldots/l_m$ then $\delta_p(S_p, l_1) = q_1$, for each j, $1 \leq j < m$, $\delta_p(q_j, l_{j+1}) = q_{j+1}$, and $q_m = F_p$. If p contains "//" then there exists a transition from a state back to itself with "*other*". That is, if $p = \ldots//l_j\ldots$ for some j then $\delta_p(q_{j-1}, other) = q_{j-1}$, where $q_0 = S_p$. The final states of these NFAs carry information about the key considered for its construction, denoted as *keyInfo*. Let $F = \{F_{Q_i}, F_{Q'_i}, F_{P_i^1}, \ldots, F_{P_i^{n_i}}\}$. For each $f \in F$, $keyInfo[f]$ contains the following information:

- `keyId`: σ_i's identifier;
- `type`: the value of this field is *context* if $f = F_{Q_i}$, *target* if $f = F_{Q'_i}$ and *keyPath* otherwise;
- `keyPathId`: a identifier for each key path. This field is defined only when $keyInfo[f].type = keyPath$; in this case, if $f = F_{P_i^j}$ then $keyInfo[f].keyPathId = j$.

The NFA for key σ_i is obtained by making the final state of $M(Q_i)$ coincide with the start state of $M(Q'_i)$, and the final state of $M(Q'_i)$ coincide with start states of $M(P_i^k)$, $1 \leq k \leq n_i$. The NFA for all keys in Σ, $M(\Sigma)$ is finally obtained by creating a new start state with ϵ-transitions to the start states of all $M(\sigma_i)$, $1 \leq i \leq n$. An example of the resulting NFA for $\Sigma = \{k_1$ `:(university, {name}),` k_2 `:(university, (//professor, {name, phone}))`$\}$ is depicted in Figure 4(a), and the corresponding *keyInfo* structure is given in Figure 4(c).



	keyId	type	keyPathId
[2]	*k1*	context	--
[3]	*k1*	target	--
[4]	*k1*	keyPath	1
[6]	*k2*	context	--
[7]	*k2*	target	--
[8]	*k2*	keyPath	1
[9]	*k2*	keyPath	2

Figure 4. DFA construction

Given NFA $M(\Sigma)$, KeyDFA(Σ) is obtained applying standard subset construction algorithm[10]. The resulting automaton for our running example is shown in Figure 4(b). Observe that, although in $M(\Sigma)$ each state contains information of at most one key in Σ, after the conversion, each state q' in KeyDFA(Σ) contains information from all original states represented by q'. As an example, in Figure 4(b), $keyInfo(\{3, 6\}) = \{keyInfo[3], keyInfo[6]\}$.

The automaton construction described in this section is similar to that defined in [9] for XML stream processing, which evaluates the effectiveness of processing a large number of XPath expressions on streams when the DFA is constructed "lazily". Although the number of states in a DFA can grow exponentially on the number of input path expressions, the number of keys for a given document is usually small. Moreover, since changes on keys are not frequent, if versions of the same document are periodically downloaded, KeyDFA(Σ) can be locally stored, instead of being recomputed at each execution of the diff algorithm.

Selection of Candidates. Given KeyDFA(Σ), algorithm XKeyMatch process each of the XML trees given as input using this automaton, gathering information on possible candidates for matchings according to Σ. Let T be one these trees, and KeyDFA(Σ) = (Q, A, δ, q_0, F). Starting with the root of T and start state q_0, T is traversed in preorder, and each step in the tree traversal corresponds to a step in its processing with the automaton. During the traversal, information about key values are collected using data stored at each state q visited. That is, suppose the current state is q when processing a node n in T. If $keyInfo[q]$ contains information on a key path of a key k, then n is a key value for some node n_t, and therefore we can associate n_t with the value of n. Observe that k may contain more than one key path. If this is the case, then n_t is identified as a candidate for matching only if it is associated with values for all key paths of k, and we say that n_t is "keyed" on k.

To keep track of the information needed to determine whether a node is a candidate for matching along the tree traversal, we associate the following with each key $k = (Q, (Q', \{P_1, \ldots, P_n\}))$ in Σ, in a structure called $keyVal[k]$:

- `contextNodes`: a set of nodes in $[\![Q]\!]$;
- `targetNode`: last node visited in $n[\![Q']\!]$ for some n in `contextNodes`;
- `keyNode`: last node visited in `targetNode`$[\![P_i]\!]$.
- `keyPathId`: key path identifier i, $1 \leq i \leq n$;

Recall that a node in the tree may play different roles (as context, target, or key path) for different keys in Σ, and this information is given by $keyInfo[s]$, where s is a state in KeyDFA(Σ). Suppose that the current state of KeyDFA(Σ) is s when processing node n in T. Then for each element v in $keyInfo[s]$ we obtain the value of v.`keyId` $= k$, and values in v are used for filling up fields in $keyVal[k]$. As an example, consider tree T_1 in Figure 1(a). Let the current node in T_1 be 2 (a `university` node), and the current state of KeyDFA(Σ) be $\{3, 6\}$. Then the algorithm sets $keyVal[k_1]$.`targetNode` $= 2$, since $keyInfo[3]$ states that the current node is the target of k_1. Moreover, node 2 is inserted in $keyVal[k_2]$.`contextNodes`, based on the value of $keyInfo[6]$. That is, structure $keyVal[k]$ is a placeholder for information gathered along the tree traversal. Whenever values for all key paths of k are found, values in $keyVal[k]$ contain data on a candidate for matching based on k. The result of function `get_candidates` on T_1 and T_2 is given in Figure 5.

The first line in the table of Figure 5(a) has been collected according to the XML key k_1 : `((university,`

keyId	contextNodes	targetNode	[keyNode, keyPathId]
k_1	1	2	{[3,1]}
k_2	2	4	{[5,1], [7,2]}

(a) `get_candidates`(T_1, `keyDFA`)

keyId	contextNodes	targetNode	[keyNode, keyPathId]
k_1	31	32	{[33,1]}
k_2	32	36	{[37,1], [39,2]}

(b) `get_candidates`(T_2, `keyDFA`)

Figure 5. Selection of candidates on T_1 and T_2

`{@name}))`, while candidate 2 has been collected according to key k_2 : `((university, (//professor, {name, phone})))`. Observe that node 13 (a `professor` node) has not been included in the set, since it does not have values for key path `phone`. Similarly, in Figure 5(b), line 1 has been collected using key k_1, while line 2 has been collected according to k_2.

Matching. Given the set of candidates for matching from both versions of XML trees T_1 and T_2, function `match` looks for candidates in these sets with the same key values. That is, we search for nodes n_1 in candidates of T_1 and n_2 in candidates of T_2 that are keyed on the same key k and coincide on the values of all key paths. When k is a relative key, we can only conclude that n_1 matches n_2 if the contexts in which n_1 and n_2 are defined also match. Consider again the XML keys in our running example. If both candidates contain `professor`-nodes with the same `name` and `phone`, we can only conclude that they are indeed the same professor if the `university` (the context) in which they are defined also match. After finding a pair of target nodes $[n_1, n_2]$ that match, this matching is propagated downwards. That is, if they have been matched based on the values of some key paths, these key path nodes can also be matched.

In function `match`, all pairs of nodes that are candidates for matching are collected in a data structure containing the following information: the key identifier k (`keyId`); the context node in T_1 (`contextNode`$_1$); the target node in T_1 (`targetNode`$_1$); the context node in T_2 (`contextNode`$_2$); the target node in T_2 (`targetNode`$_2$); a set of records [`keyPathId`, `keyNode`$_1$, `keyNode`$_2$], where `keyNode`$_1$ and `keyNode`$_2$ are key nodes in T_1 and T_2, respectively, with the same value.

As an example, consider again XML trees T_1 and T_2 depicted in Figure 1 and the output of function `get_candidates` on T_1 and T_2 given in Figure 5. After comparing the values of candidates for matching, the resulting set contains the values shown in Figure 6. The first candidate is included in the set because nodes 2 and 32 are `university` nodes in T_1 and T_2, respectively, with the same `@name` value. Similarly, the second candidate is inserted because nodes 4 and 36 are `professor` nodes that coincide on the values of both `name` and `phone`. To compute the resulting set of matches, we start by inserting the pair $[1, 31]$, the roots of T_1 and T_2, in the set. When processing the first candidate $[2, 32]$ it is checked whether their context has already been matched. Since this is the case, $[2, 32]$ is inserted in the result. This matching is propagated downwards and pair $[3, 33]$ (`@name` nodes) are also inserted in the result. For the second candidate $[4, 36]$, it is checked whether their context $[2, 32]$ has already been matched. Since this is also the case, we can conclude the it is indeed a match and insert the pair in the resulting set. The propagation of matches includes in the result the following pairs: $[5, 37]$, $[6, 38]$ (`name` nodes), $[7, 39]$ and $[8, 40]$ (`phone` nodes).

keyId	context node$_1$	target node$_1$	context node$_2$	target node$_2$	[keyPathId, keyNode$_1$, keyNode$_2$]
1	1	2	31	32	{[1,3,33]}
2	2	4	32	36	{[1,5,37], [2,7,39]}

Figure 6. Candidates for matching

Implementation. Algorithm XKeyMatch has been implemented in C++, using DOM [3]. XyDiff is the algorithm chosen to take as input the set of matches resulting from XKeyMatch, find additional matchings, and generate the edit script. Some libraries from XyDiff have been used by XKeyMatch to implement the communication between them. We have conducted some experiments to evaluate the effectiveness of our proposal, and to determine if it indeed solves the problems detected in other diff algorithms and reported in Section 1. The results are very encouraging. In particular, for Example 1, the definition of a single key $(//professor, \{name\})$ prevents both problems described in the introduction. A final remark is that, although our proposal can have an impact on the performance of the diff algorithm to which algorithm XKeyMatch is applied to preprocess the input XML documents, many applications favor the quality of the result rather than the efficiency of the algorithm.

4. Conclusion

We have proposed a new approach in the context of diff algorithms for XML. As opposed to previous works, that are based solely on the structural analysis of XML documents, our technique takes into consideration their semantics. Our approach consists of extending the structural analysis with a preprocessing phase which uses XML keys to match elements that refer to the same entity in two versions of the document. Although XKeyMatch requires the user to be familiar with the documents being compared, when the input keys faithfully capture their semantics, our algorithm always generates more meaningful results than others based solely on structure and value similarities. One topic for future work is to perform an experimental study using large amounts of data, especially real ones, in order to determine the impact of the preprocessing phase in practice. Possible test beds are scientific databases, since they present appropriate structure and behavior.

References

[1] S. Abiteboul, G. Cobna, and A. Marian. Detecting changes in XML documents. *ICDE'02*, 2002.

[2] R. Al-Ekram, A. Adma, and O. Baysal. diffx: an algorithm to detect changes in multi-version xml documents. In *CASCON '05: Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, pages 1–11. IBM Press, 2005.

[3] V. Apparao et al. Document Object Model (DOM) Level 1 Specification. W3C Recommendation, Oct. 1998.

[4] T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. W3C Recommendation, Feb. 1998.

[5] P. Buneman, S. Davidson, W. Fan, C. Hara, and W. Tan. Keys for XML. *Computer Networks*, 39(5):473–487, Aug. 2002.

[6] J. Clark and S. DeRose. XML Path Language (XPath). W3C Working Draft, Nov. 1999.

[7] G. Cobena, T. Abdessalem, and Y. Hinnach. A comparative study for xml change detection, 2002.

[8] FSF. Gnu diff. Available at http://www.gnu.org/software/diffutils/diffutils.html.

[9] T. J. Green, A. Gupta, G. Miklau, M. Onizuka, and D. Suciu. Processing xml streams with deterministic automata and stream indexes. *ACM Trans. Database Syst.*, 29(4):752–788, 2004.

[10] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[11] J. Maletic and A. Marcus. Data Cleansing: Beyond Integrity Analysis. *Proceedings of The Conference on Information Quality (IQ2000), Massachusetts Institute of Technology, Boston, MA, USA*, pages 200–209, 2000.

[12] L. Peters. Change detection in xml trees: a survey. Technical report, University of Twente, 2005.

[13] S. M. Selkow. The tree-to-tree editing problem. *Information Processing Letters*, 6:184–186, 1977.

[14] K. Tai. The tree-to-tree correction problem. *Journal of the ACM*, 3(26):422–433, 1979.

[15] Y. Wang, D. J. DeWitt, and J. Cai. X-Diff: an effective change detection algorithm for XML documents. *ICDE*, pages 519–530, 2003.

[16] H. Xu, Q. Wu, H. Wang, G. Yang, and Y. Jia. Kf-diff+: Highly efficient change detection algorithm for xml documents. In *On the Move to Meaningful Internet Systems, 2002 - DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002*, pages 1273–1286, London, UK, 2002. Springer-Verlag.

[17] K. Zhang, R. Stgatman, and D. Shasha. Simple fast algorithm for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18:1245–1262, 1989.

XML Schema Evolution by Context Free Grammar Inference

Júlio C. T. da Silva
Federal University of Paraná
juliocesar@inf.ufpr.br

Martin A. Musicante
Federal Univ. of Rio Grande do Norte
mam@dimap.ufrn.br

Aurora T. R. Pozo
Federal University of Paraná
aurora@inf.ufpr.br

Silvia R. Vergilio
Federal University of Paraná
silvia@inf.ufpr.br

Abstract - **XML has become one of the most usual technologies for data storage, manipulation and transference. Schema languages for XML allow defining application-specific formats (or schemas) for XML documents. Schemas are easily verifiable. However, as the application life-cycle goes on, schemas need to be changed in accordance to any new requirement of the data they define. For this reason it is useful to create mechanisms to automatically produce these changes. In this work, we explore the correspondence between schemas and context-free grammars. We focus our attention on the automatic induction of context-free grammars which, in turn, will be translated to schemas for XML documents. An algorithm of grammatical inference is proposed. Our work extends the well-known LL Parsing algorithm to perform grammar inference.**

1. Introduction

XML (eXtensible Markup Language) has become one of the most important technologies to the storage, manipulation, and transference of data. Very simple and strict formation rules allow XML data to be easily defined and verified with respect to a given schema [15]. Schemas for XML are languages for the definition of specific formats of XML documents. They define a set of rules according to which it is legal to create valid XML documents. Some examples of schema languages and validators are DTD (Document Type Definition) [5] and XML Schema [14].

Often, schemas need to be modified in order to match new requirements of the applications (which manipulates the data they define). These modifications are usually application-domain dependent and they are promoted by data administrators whose domain of expertise is not computer science. Moreover, schema extensions are usually conservative; the new version of the schema must be able to validate previous versions of the XML data.

In [4], Bouchou et al propose a method to dynamically evolve schemas for XML databases by means of inducing the regular expressions contained in DTD documents. Only one modification can be made in the schema per run of the algorithm. It is interesting to study methods that allow more modifications per run, for example, methods based on grammar inference. These methods are particularly suitable, because an XML schema validator defines a language (a set of strings or trees); in this case, a set of XML documents and there is a correspondence between XML schema validators and other formal models to specify languages, like regular expressions and context-free grammars.

Some works on grammar inference are found in the literature. A strategy to induce context-free grammars using Genetic Programming (GP) [8] is proposed in [10]. ICYK, an inductive algorithm based in CYK [7], is presented in [11] and [12]. However, neither the GP approach nor the ICYK algorithm was designed to work with schemas for XML. They do not worry about keeping the structure of the induced grammars similar to a schema. The GP algorithm generates and changes the grammars randomly. ICYK works only with grammars in a specific normal form, where the production rules must have a fixed, short size. To keep the proximity between grammars and schemas, it is not interesting to manipulate grammars randomly, but using an auxiliary structure, like in ICYK. In addition to, it is useful to use a parsing algorithm which works with grammars in a more flexible form.

LL Parsing algorithm [1] is an alternative to CYK and presents some desired characteristics. It uses some structures (a table, and a stack) to control the parsing and it works with production rules very similar to schemas, with different sizes, and of any length. Further, the complexity of LL is O(n), while CYK is O(n^3).

Considering these aspects, in this paper, we introduce an algorithm, named *ILLA* (Inductive LL Algorithm) and based on the ideas of ICYK. Given an initial grammar G, new productions to be added in G are generated by using a parsing table. G becomes capable to recognize the new desired word and keeps recognizing the same words as before. It can be used in the context of XML schemas and to evolve context-free grammars that can be manipulated by LL. If no grammar is available, we extended *ILLA* to infer grammars only from set of samples, similarly to works based on GP. This extension, named *SILLA*, is also described and evaluated in an experiment.

The paper is organized as follows. Section 2 describes *ILLA* and illustrates how it works for evolving a XML schema. Section 3 presents the algorithm *SILLA* and contains results of its evaluation. Section 4 shows related work. Section 5 discusses conclusions and future works.

2. Inductive LL Algorithm

In this section, we introduce *ILLA*, an inductive algorithm based on LL(1) (ILLA). To do this, we first present how LL works.

2.1 LL Parsing algorithm

LL is a predictive, non recursive parsing algorithm described in [1]. It uses a parsing table and a stack to control the grammatical derivation (Fig. 1). The parsing table is a bi-dimensional table, of non-terminal by terminal symbols. Each cell [*X*, *a*] keeps which production rule must be consumed when *X* is on the top of the control stack and *a* is the next input symbol. The parsing table is constructed with the aid of two functions: *First* and *Follow*.

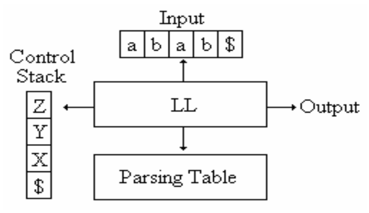


Figure 1 –LL Parser

Given a context-free grammar, the function *First* receives a sequence of grammar symbols as its argument. It returns the set of terminal symbols that can start strings derived from that sequence. The function *Follow* receives a non-terminal symbol and returns the set of terminal symbols that can appear immediately to the right of that non-terminal symbol in any derivation of the grammar.

2.2 Inductive LL Algorithm (*ILLA*)

ILLA is an iterative-deepening search algorithm. The search space is created by the generation of new production rules that can be added to the grammar. When the LL Parser detects that the example being parsed is not derived by the grammar, *ILLA* creates new production rules based on the parsing table used in the derivation. Fig. 2 shows the main function of *ILLA*:

```
ILLA(grammar G, string w, integer n, nMax): returns grammar;
  Try to derive w using LL Parser;
  If it is well-succeeded, then return G;
  Else,
    If n <= nMax, then
      TS := Create_TS_Set(G, P, a),
            where P is the stack used in LL, and
                  a is the terminal symbol of w
                  that was being read when the parsing failed;
      Return Test_TS(G, w, TS, n, nMax);
```

Fig. 2 – Main function of *ILLA*

ILLA function verifies if *G* derives *w*. If not, it calls the function *Test_TS* (Fig. 3), which adds to *G* production rules of a set of options *TS,* created by the function *Create_TS_Set* (Fig. 4), and verifies if the resulting grammar derives *w*.

```
Test_TS(grammar G, string w, set TS, integer n, nMax): returns grammar;
  repeat
    Choose an element TS[i];
    NG := G added with the production rules in TS[i];
    Remove element i from TS;
    Return ILLA(NG, w, n + 1, nMax);
  until NG derives w, or TS is empty;
```

Fig. 3 – Function *Test_TS*

The function *Test_TS* adds an element of *TS* to *G*, and calls recursively the main function *ILLA*, with the adjusted parameters. The function *Create_TS_Set* estimates, according to the error occurred in the parsing, how the production rules must be created, in sense of filling correctly the parsing table of LL. Given a combination of parameters, the function *Create_TS_Set* can call the functions *Create_Optional_Terminal* and *Create_Options_From* (Figs. 5 and 6).

The function *Create_Optional_Terminal* is called when the symbol on the top of *P* is a terminal symbol. This function makes this terminal symbol optional in the production rules that it appears.

```
Create_TS_Set(grammar G, stack P, terminal symbol a):
                    returns a list of sets of production rules;
  If a = $ (that is, if w was derived until the last symbol, but there was still
non-terminal symbols on P), then
    If the symbol on top of P is a terminal symbol, then
      Returns Create_Optional_Terminal(P[top], G);
    Else
      Returns the set { Y ::= ε | Y is a non-terminal in P };
  Else
    Let TS be a set of sets of production rules;
    For each X in P, do
      If X is a terminal symbol, then
        Add Create_Optional_Terminal(X, G) to TS;
      Else,
        Add Create_Options_From(X, a, G) to TS;
        From the second element X on, if the production rule 'X ::= ε'
doesn't belong to G, it is added to all sets generated from X;
    Return TS;
```

Fig. 4 – Function *Create_TS_Set*

```
Create_Optional_Terminal(terminal symbol a, grammar G):
                    returns a set of sets of production rules;
  Let TS := ∅;              // TS is a set of sets of production rules
  For each 'X ::= αaβ' in G, do
      TS = TS - { X ::= αaβ } ∪ { X ::= αAβ,   A ::= a | ε }
  Return TS;
```

Fig. 5 – Function *Create_Optional_Terminal*

```
Create_Options_From(non-terminal X, terminal a, grammar G):
returns a set of sets of production rules;
  Let TS := ∅;              // TS is a set of sets of production rules
  //a) Add a simple production rule TS:
   TS = TS ∪ { X ::= a };
  //b) optional
   For each rule 'Y ::= αXβ' of G, do
        TS = TS - { Y ::= αXβ } ∪ { Y ::= αXZβ,   Z ::= a | ε }
  //c) zero or more
  For each 'X ::= aβ' of G, do
        TS = TS - { X ::= aβ } ∪ { X ::= aβX | ε }
  //d) one or more
  For each 'X ::= aβ' of G, do
       TS = TS - { X ::= aβ } ∪ { X ::= aZ,  Z ::= βX | ε }
  Let F := ∅;               // F is a set of sets of terminal symbols
  If ε ∈ First(X), then F := First(X) U Follow(X);
  Else, F := First(X);
  For each x ∈ F, do
    For each 'Y ::= αxβ' of G, do
       //e) concatenation
       TS = TS - { Y ::= αxβ } ∪ { Y ::= αZ,   Z ::= xβ | αxβ }
      //f) or        TS = TS  ∪ { X ::= αZ,   Z ::= xβ | aβ }
    Return TS;
```

Fig. 6 – Function *Create_Options_From*

The new test set (*TS*) is formed by substituting the production rule 'X *::= αaβ'* by a pair of rules, to make the symbol 'a' optional. This rule replacement is conservative, in the sense that all strings that can be generated using the old rule will be generated using the new ones. The function *Create_Options_From* creates new production rules for the grammar when the error on the parsing process occurs because the cell [*X, a*] in parsing table is empty. The options are generated according to the properties of context-free grammars, aiming to fill correctly the Parsing Table. *ILLA* only generates new non-terminal symbols when necessary, to avoid left recursion or ambiguity.

2.3 Evolving schemas

Now we illustrate how *ILLA* works for evolving schemas. Consider the following DTD document:

```
<?xml version="1.0"?>
<!DOCTYPE contact [
  <!ELEMENT contact (name,email*,telephone)>
  <!ELEMENT name    (#PCDATA)>
  <!ELEMENT email   (#PCDATA)>
  <!ELEMENT telephone   (homephone,celphone)>
  <!ELEMENT homephone  (#PCDATA)>
  <!ELEMENT cellphone  (#PCDATA)>]>
```

That schema corresponds to the following grammar G.

```
1) contact   ::= name.email.telephone
2) name      ::= #PCDATA
3) email     ::= #PCDATA.email
4) email     ::= ε
5) telephone ::= homephone.celphone
6) homephone ::= #PCDATA
7) cellphone ::= #PCDATA
```

A number is associated to each rule of G to ease reference on Tables 1 and 2. Now, consider the following XML document as the positive sample *w*.

```
<contact>
  <name>John Winston</name>
  <email>johnwinston@wch.com</email>
  <telephone>
    <homephone>555-5555</homephone>
    <celphone>999-9999</celphone>
    <officephone>123-4567<officephone>
  </telephone>
</contact>
```

The first call to *ILLA* generates the parsing table shown in Table 1. When the algorithm tries to read the cell [officephone, #PCDATA] in the parsing table, LL fails because this cell does not exist. That means grammar *G* does not derive *w*.

Table 1 – Parsing table to the first calling of ILLA

	#PCDATA	**#PCDATA**	**#PCDATA**	**#PCDATA**	**$**
contact	1				
name	2				
Email		3	4		
telephone			5		
homephone			6		
celphone				7	

Then, *ILLA* has the task of adding production rules to *G* until *w* be derived. When the parsing fails, *ILLA* calls the function *Create_TS_Set*, which creates the set *TS* bellow.

```
TS[1] = [cellphone ::= #PCDATA6]
TS[2] = [telephone ::= homephone.celphone.officephone,
          officephone ::= #PCDATA6, officephone ::= ε]
```

ILLA chooses randomly an element of TS and adds to *G* the production rules contained in it. For demonstration, it is chosen the element TS [2]. The resulting grammar *G'* is shown below:

```
1) contact   ::= name.email.telephone
2) name      ::= #PCDATA
3) email     ::= #PCDATA.email
4) email     ::= ε
5) telephone ::= homephone.celphone.officephone
6) homephone ::= #PCDATA
7) cellphone ::= #PCDATA
8) officephone ::= #PCDATA
9) officephone ::= ε
```

ILLA is called recursively with *G'*, *w*, and the increased iterator as input parameters. The parsing table of the second calling to *ILLA* is shown in Table 2.

Table 2 – Parsing table to the second calling of *ILLA*

	#PCDATA	**#PCDATA**	**#PCDATA**	**#PCDATA**	**#PCDATA**	**$**
contact	1					
name	2					
Email		3	4			
telephone			5			
homephone			6			
celphone				7		
officephone					8	9

When *ILLA* tries to parser *w* in grammar *G'*, the parsing is successful. Then, *G'* is returned as the inferred grammar.

A DTD document that can be generated from the inferred grammar is shown below.

```
<?xml version="1.0"?>
<!DOCTYPE contact [
  <!ELEMENT contact (name,email*,telephone)>
  <!ELEMENT name       (#PCDATA)>
  <!ELEMENT email      (#PCDATA)>
  <!ELEMENT telephone
    (homephone,celphone,officephone?)>
  <!ELEMENT homephone  (#PCDATA)>
  <!ELEMENT cellphone  (#PCDATA)>
  <!ELEMENT officephone  (#PCDATA)>
]>
```

2.4 Inferred grammars

The language recognized by the inferred grammar contains the set of strings that was recognized by the old grammar. However, adding new production rules can give to the grammar a power of derivation much bigger than it is necessary. For this reason, *ILLA* accepts as parameter a set of negative samples, and during the inference process, the algorithm discards any grammar that derives at least one of these negative samples.

The procedure for generation of production rules attends the requirements of LL grammars. An ambiguous grammar can not be manipulated by LL. Because of this, if such grammar is generated it will be discarded.

2.5 Limitations of *ILLA*

ILLA generates new production rules based on the existent rules in the grammar. So, grammars with large number of rules result in a large number of new production rules, and it makes the algorithm performance worse. The search of *ILLA*, in an average case, has the exponential complexity $O(a^n)$, where a is the average number of production rules generated per iteration, and n is the depth. If either a or n is large, the search will be intractable. However, the number of production rules generated is quite reduced if the grammar has not empty production rules, which is usual for grammars representing schemas for XML.

3. Synthesis with *ILLA*

For the purpose of inferring a grammar only from sets of samples, we propose an extension to *ILLA*, an algorithm called *SILLA*. It generates an initial population of grammars based on the structure of the positive samples. Then, it uses *ILLA* to infer grammars for each positive sample.

SILLA uses concepts of the Evolutionary Computation (EC) [2], like population, fitness, and selection of individuals. However, it does not use gene combination and mutation, because this would result in losing the LL properties of the synthesized grammars. *SILLA* gets as input parameters sets of positive and negative samples, and returns the grammar of the final generation with best fitness. The used fitness function is:

$$\frac{\dfrac{\textit{correct positive samples}}{\textit{all positive samples}} + \dfrac{\textit{correct negative samples}}{\textit{all negative samples}}}{2}$$

SILLA creates the initial population, evaluates the fitness of every individual, and verifies if there is a complete solution (a grammar which classifies correctly all the samples). If not, it starts the evolutionary process:

- For each positive sample, and for each grammar of the population, *SILLA* calls *ILLA*, and keeps the resulted grammars in a list of candidates to the next generation;
- The candidates in the list are evaluated and, if there is no complete solution, the selection of individuals is performed, resulting in the new generation. So, the process restarts;
- *SILLA* ends when a complete solution is found, or when the maximum number of generations is reached.

The main function of *SILLA* is presented in Fig. 7. It controls the evolution of the grammars, by means of calling the functions *Create_Initial_Population*, *Evaluate_Fitness*, *Select_Next_Population* (Figs. 8 and 9), and *ILLA*.

The function *Create_Initial_Population* creates grammars based on positive samples in two ways: with large number of production rules, in Chomsky Normal Form and grammars that, seen as trees, have the leaf nodes in the deepest level.

```
SILLA(set of positive samples P, set of negative samples N, integer nMax):    returns a grammar;
  Let Population and Candidates be sets of grammars;
  Population := Create_Initial_Population(P);
  Let i := 1;
  Loop
    If Population includes is a complete solution, then
      Return the complete solution;
    If i > nMax, then
      Return the grammar in Population with best fitness;
    Else,
      Population = Evaluate_Fitness(Population, P, N);
      For each sample w ∈ P, do
        For each grammar G ∈ Population, do
          Candidates := Candidates ∪ ILLA(G, w, n, nMax);
        Evaluate_Fitness(Candidates, P, N);
        Population:= Select_Next_Population (Population, Candidates);
    i++;
  End;
```

Fig. 7 – Main function of *SILLA*

```
Create_Initial_Population(set of positive samples P):
                                        returns a set of grammars;
  result := ∅;                  // a set of sets of production rules
  For each sample w ∈ P, such that | w | = n, do
    result := result ∪{ { S ::= AB, A ::= w[1], B ::= CD,
              C ::= w[2], ... X ::= YZ, Y ::= w[n-1], Z ::= w[n] } }
    Let Avg, Deep and i be integers;
    Add to result a grammar with the following form:
      For Avg := 1 to average size of production rules, do
        Deep := ⌊ log_Avg ( |w| ) ⌋;
        For i = 1 to Deep, generate the production rules:
          - 'S ::= $A_1A_2...A_{Avg}$'
          - '$A_1$ ::= $B_1B_2...B_{Avg}$'
          - '$A_2$ ::= $C_1C_2...C_{Avg}$'
          - (...)
        For every non-terminal symbol X in right side of the production
rules generated in last iteration of the previous loop, generate 'X ::= w[i]'
until consuming all the symbols of w. For the remaining non-terminal
symbols, generate 'X ::= ε';
```

Fig. 8 – Function *Create_Initial_Population*

Notice that when *SILLA* is used in the context of XML schema evolution, the initial population is given only by the original schema. The function *Create_Initial_Population* is not called.

The function *Evaluate_Fitness* updates the set of grammars received, calculating the parameters *fitness*, *correct_positive*, and *correct negative* of every grammar.

The function *Select_Next_Population* fills the new population according to the parameters *fitness*, *correct_posivite,* and *correct_negative*, and their respective weights (previously configured) *F_Weight, P_Weight, and N_Weight.*

3.1 Experiment using *SILLA*

In this section, we evaluate the use of *SILLA* and present some results of an experiment accomplished to compare *SILLA* with the GP approach. To do this, we repeated the experiment described in [9] and summarized in Table 3.

```
Select_Next_Population(set of grammars S1, set of grammars S2): returns a set of grammars;
    U := S1 ∪ S2;
    Fill F_Weight% of result with the grammars of U that have the best fitness, and remove them from U;
    Fill P_Weight% of result with the grammars of U that have the best correct_positive, and remove them from U;
    Fill N_Weight% of result with the grammars of U that have the best correct_negative, and remove them from U;
    Fill the rest of result with grammars chosen randomly in U;
```

Fig. 9 – Function *Select_Next_Population*

We used the same training set, 7 languages extracted from the benchmark called "Tomita Language Set" (TLS). Column *TL* indicates which set of TLS was used. *Avg. Evals* is the average number of evaluations per run before a solution to the training set was found. *Gen. Accuracy* is the percentage of strings correctly classified, calculated as shown below:

$$\frac{\textit{Correct positive examples + correct negative examples}}{\textit{All positive examples + all negative examples}}$$

In our experiment, 50 runs of *SILLA* were performed for each set of strings from TLS. Each run has a population of 50 individuals. The deepness limit is 50 generations. Our results are presented in Table 4:

The general average of accuracy in the experiment using *SILLA* (89,6%) was better than in the experiment using GP (76,6%). The worst average in experiment with GP is 65,25% and with *SILLA* is 77%. While the experiment with GP has no 100% of accuracy for any set, 2 sets of TLS were 100% correctly classified using *SILLA*. For more complex samples, that is, grammars with a large number or rules, the metrics using *SILLA* were worse.

Table 3 – Results of the experiment

TL	Avg. Evals.	Gen. Accuracy		
		Avg. (%)	Variation	Best (%)
1	30	88,39	0,0391	100
2	1010	84,00	0,0232	100
3	12450	66,28	0,0174	100
4	7870	65,25	0,0324	100
5	13670	68,65	0,0147	82,94
6	2580	95,94	0,0269	100
7	11320	67,69	0,0221	100

Hence, it can be said that GP explores a large search space, where the GP algorithm not always converges to complete solutions, even in simple cases, but gets good solutions in the end. In other hand, for these simple cases, *SILLA* always converged. The space searched by *SILLA* is not as large as the one searched by GP. We observed in the experiment that in more complex cases, its search was limited in local maximums and *SILLA* did not find a good solution to the inference. We are now working on this problem.

4. Related Work

With the goal of evolving XML schemas, Bouchou et al [4] proposes an algorithm, named GREC. Regular expressions of a DTD document are transformed in automata that will be induced by GREC. This algorithm allows only one modification per run and during the maintenance of the schemas, different updates are usually necessary to the schema validate the new data. To do this, we propose in this paper the study of the new field of grammar inference. It can be explored, by exploring the correspondence existent between schemas and grammars

Table 4 – Results of the experiment using *SILLA*

TL	Avg. Evals.	*SILLA* Accuracy		
		Avg (%)	Var.	Best (%)
1	53,91	100	0	100
2	109	100	0	100
3	1425,99	86,10	1,252	92
4	2072,53	89,67	1,701	91,33
5	2539,20	77	2,081	84,08
6	2517,34	83,11	1,390	87
7	3011,33	91,43	1,563	93,94

Grammatical Inference (GI) (or induction) is a process of learning a grammar from a set of training data. The most traditional field of GI is pattern recognition and other areas such as gene analysis, sequence prediction, cryptography and information retrieval [3,6,13]. These works do not address evolution of schemas.

Two works on GI and mentioned before are the basis of our study. The first work is the inductive algorithm based on CYK Parsing algorithm [7], described in [11] and [12]. This algorithm, named Inductive CYK (ICYK) is a component of an inductive grammar inference system called Synapse. ICYK adds new production rules to the grammar when it does not derive a positive sample.

At first, the system has no production rules. For a given positive sample string, it generates minimum production rules to derive this string. Then it checks that the rules do not derive any given negative samples. This process continues until the system finds a rule set which derives all the positive samples and none of the negative samples. For generating production rules, the system uses Inductive CYK algorithm, which generates sets of rules required for parsing positive samples. The inductive inference is based on incremental search, or iterative deepening, in the sense that the rules sets are searched within the limits of the minimum numbers of non-terminal symbols and rules. When the search fails, the system iterates the search with larger limits.

The second work uses Evolutionary Computation techniques [2], particularly Genetic Programming [8], to induce context-free grammars [9,10]. The grammars are represented as programs constructed by sets of terminal and non-terminal symbols. The positive and negative samples are used to calculate the fitness function of the algorithm. Besides the basic operators of GP, heuristic operators are proposed in [10]. Also a better initial population is created instead of random generation. As it is noticed in [10], this algorithm infers only grammars that have a small number of productions rules.

In [9], another Genetic Programming-based method is presented. It induces a classification mechanism for positive and negatives samples of a language. The individuals are represented by finite-state automata, and the fitness function is defined according to the classification of a training set.

Neither the GP approach nor the ICYK algorithm was designed to work with schemas for XML. They do not worry about keeping the structure of the induced grammars similar to a schema. The advantage of ICYK is to implement a control and structures to manipulate the grammars during the search. Its advantage is to work with only grammars in a specific normal form that is not similar to schemas. Beside this, the parsing algorithm CYK is $O(n^3)$. The advantage of the GP algorithm is to allow induction from positive and negative samples.

The algorithms described in this paper combine the main ideas and advantages of abovementioned works and present characteristics that make them suitable to the context of XML schemas evolution. *ILLA* uses LL to control the parsing and it works with production rules very similar to schemas, with different sizes, and of any length. Further, the complexity of LL is O(n). *SILLA* allows evolution from samples if no initial grammar is available.

5. Concluding Remarks

This paper proposes a method of grammatical inference based on LL Parser. Grammars can be inferred from a grammar and a positive sample (*ILLA*), or from sets of samples (*SILLA*).

One contribution of this method to researches in grammatical inference area is the use of a Parser with linear complexity. The Parser used in ICYK, for example, has cubic complexity. Furthermore, ICYK works with context-free grammars in a rigid normal form, while *ILLA* is more flexible with the grammars form. It is a contribution to evolution of schemas for XML area, for allowing that schemas be represented by grammars more similar with them.

Another contribution in this area is that *ILLA* can perform various modifications in a grammar per run, while the method proposed in [4] performs only one. This paper presented the results of an experiment that used *SILLA* to infer the grammars that describes the languages of the samples in a benchmark of samples. The results were compared with a similar experiment, using GP. From the comparison, it can be observed that the search space examined by *SILLA* is not as large as in GP. However, *SILLA* was more efficient to infer grammars from simpler samples.

Some improvements can be done in the future on the algorithm proposed in this paper. One of the most significant improvements is the proposition of a useless production rules identifier, to be used during the process of new production rules generation. This can reduce the search space without lose the convergence to solution. The proposition of a heuristic method to evaluate the production rules fitness is a useful improvement. This allows the elaboration of a strategy that mixes learning machine and evolutionary computation. Still in this context, *SILLA* can be extended to work with all the concepts of evolutionary computation, mainly gene combination. A better generation of the initial individuals is also recommended.

The use of a LL(n) parsing algorithm, with $n > 1$, is another interesting improvement, in the sense of helping to predict what production rules must be created to infer a new grammar.

In the context of evolution of schemas for XML, an interesting future work is the integration of *ILLA* with an algorithm that generates grammars from schemas. It will enable evaluation experiments with actual XML schemas.

References

[1] Aho, Alfred V., Sethi, Ravi, and Ullman, Jeffrey D. Compilers, principles, techniques, and tools. 1942, reprinted with corrections in March, 1998. Addison-Wesley.

[2] Back, T., Urich, H., and Schwefel, H. P. Evolutionary Computation: Comments on the History and Current State. IEEE Trans. on Soft. Engin., Vol 17, pp. 3-17, 1991.

[3] Basu, M. Introduction to Biological Sequence Analysis, Tutorial Presentation, World Congress on Computational Intelligence, Hawaii, May, 12-17, 2002.

[4] Bouchou, B., Duarte, D., Alves, Mírian H. F., Laurent, D., Musicante, Martin A., Schema Evolution for XML: A Consistency-Preserving Approach, Lecture Notes in Computer Science, Volume 3153, Jan 2004, pp 876 - 888.

[5] DTD Tutorial. W3Schools Online Web Tutorials: http://www.w3schools.com/dtd/, visited in 26/02/2006.

[6] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. Biological Sequence Analysis, Cambridge University Press, New York, 2000.

[7] Hopcroft, John E., and Ullman, Jeffrey D., Introduction to Automata Theory, Languages, and Computation. 1979. Addison-Wesley.

[8] Koza, R. John. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, 1992.

[9] Luke, S., Hamahashi, S., and Kitano, H. "Genetic" Programming. In GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conference, Banzhaf, W. et al, eds. San Fransisco: Morgan Kaufmann, 1999.

[10] Mernik, M., Crepinsekj, M., Gerlic, G., Zumer, V., Bryant, B., and Sprague, A. "Learning Context-Free Grammars Using an Evolutionary Approach", Technical Report, University of Maribor and University of Alabama at Birmingham, 2003.

[11] Namakura, K., and Ishiwata, Y., Synthesizing context free grammars from sample strings based on inductive CYK algorithm, Fifth International Colloquium of Gramatical Inference, LNAI 1981 Springer-Verlarg, 2000.

[12] Namakura, K., and Matsumoto, M., Incremental Learning Of Context Free Grammar. School of Science and Engineering, Tokyo, Japan.

[13] Paun, G. (Ed.), Mathematical aspects of natural and formal languages, World Scientific Series In Computer Science, vol. 43, World Scientific, Singapore, 1994.

[14] XML Schema Tutorial. W3Schools Online Web Tutorials: http://www.w3schools.com/schema/, visited in 26/02/2006.

[15] XML Tutorial. W3Schools Online Web Tutorials: http://www.w3schools.com/xml/, visited in 26/02/2006.

Software Tradeoff Assistant: An Integrated Framework for Analytical Decision Making and Tradeoffs in Software Development

Rattikorn Hewett and Vikram Patankar
Department of Computer Science, Texas Tech University
rattikorn.hewett@ttu.edu, vikram.patankar@ttu.edu

Abstract

In current software practices, tradeoffs are typically performed in isolation using specific techniques at various stages of software development. Software practitioners lack the ability to share common knowledge about software development factors, select different tradeoff mechanisms, and integrate results from various stages. Furthermore, most existing tradeoff techniques in software engineering do not explicitly resolve conflicts from multiple stakeholders.

This paper presents Software Tradeoff Assistant (STA), an integrated framework that provides decision aids for enhancing understanding and resolving complex tradeoffs at various stages of software development. By integrating various analytical decision making techniques and ontology of software characteristics into tradeoff analysis, STA provides a structured process for reasoning about tradeoffs systematically. STA also includes a tradeoff methodology that facilitates a sound quantitative evaluation of alternatives along with integration of preferences from multiple stakeholders. We describe a preliminary design of STA with an illustrated case study.

1. Introduction

A tradeoff refers to losing one quality or aspect of something in return of gaining another quality or aspect. It implies a decision to be made with full comprehension of both the upside and downside of a particular choice. Most real world problems involve multiple objectives, or a wide range of factors for a single objective. Because of limited resources, it may not be possible to achieve all of the objectives or satisfy all constraints without sacrificing some others.

Tradeoffs are pervasive throughout the software development life cycle irrespective of the size, nature, and complexity of the software project. These tradeoffs made by different roles at different phases of software development change with the progress of the project. During the planning phase of the project, a tradeoff between resource allocation, project cost, and delivery time has to be performed. During the requirement specification phase, there is a tradeoff between the choice of technology, quality requirements, and the development time. Design phase has its own tradeoffs like selecting the suitable architecture satisfying security, maintainability, and performance attributes. After implementation is complete, the testing phase brings its tradeoffs in selecting testing mechanisms, testing depth, and time to stop testing. Software release management again involves tradeoff decisions regarding different features to be included in the product release satisfying the precedence constraints and in the expense of time-to-market. Thus, tradeoffs are ubiquitous in software development and therefore must be tackled explicitly in a judicious manner.

Analyzing tradeoffs is essential and extremely difficult. It is also a knowledge intensive and time-consuming process. Tradeoffs in software development are complex and difficult due to following issues:

- *Uncertainty:* Software development is inherently uncertain because of incomplete information causing unavoidable changes (e.g., staff leaves, uncertain resources or customers do not know exactly what they want software to do leading to changes in requirements and design).
- *Complexity of the decision problems:* Software development often deals with decision problems of multiple objectives, multi-leveled strategies and criteria that involve a large number of decision factors. Making such decisions require a thorough understanding of all aspects of the problem. For example, selecting appropriate technology for application development requires a good balance between technical and management impacts including schedule, cost, resource utility, and quality. There is a need for a principled method for making informed decisions for complex constrained problems.
- *Multiple Stakeholders:* Different decision processes usually involve multiple stakeholders, each of which may contain a different set of needs, preferences, and constraints, which may be in conflict with one or another. It is impossible to satisfy all stakeholders especially when resources are limited. In software development, a number of stakeholders can be large making a simple negotiation inadequate.

To overcome the above difficulties, a systematic tradeoff analysis that offers means to allow rational and informed decisions would be useful. Although many organizations have independent tradeoff analysis methods, these analyses are typically performed in isolation at different stages of software development life cycle [1, 2, 8, 9]. In addition, they mostly rely on manual processes that tend to be ad-hoc and extremely time-consuming. Current practices also suffer from the lack of the ability to provide sharing and reuse of common knowledge about software development factors, selection of different tradeoff mechanisms, and integration of results from various stages of a software development life cycle. Furthermore, most existing tradeoff techniques in software engineering do not explicitly resolve conflicts from multiple stakeholders (e.g., [9, 11, 13]). Our research aims to alleviate the above issues to assist software developers and managers in analyzing tradeoffs in complex decision problems during various software development activities.

This paper presents *Software Tradeoff Assistant* (STA), an integrated framework for tradeoff analysis that provides semi-automatic decision aids for enhancing understanding and resolving complicated tradeoffs throughout the software development life cycle. STA includes a tradeoff methodology that facilitates a sound quantitative evaluation of alternatives along with integration of preferences from multiple stakeholders. The paper is organized as follows. Section 2 describes the components of the STA framework and Section 3 provides more details of tradeoff analysis followed by its illustrated case study in Section 4. Section 5 presents related work of tradeoff analysis applied to software development. Section 6 concludes with summary and future work.

2. STA Framework

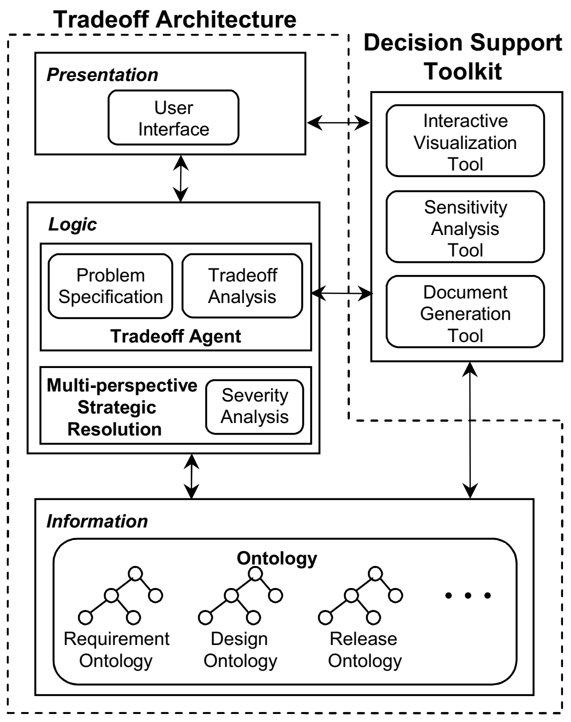


Fig 1. The STA overall framework.

Figure 1 shows an overall framework of STA, which can be divided into two main modules: the tradeoff architecture and the decision support toolkit. The former provides basic core for tradeoff functionalities, whereas the latter gives supplementary analysis and utilities to support the tradeoff reasoning. We describe each of them in more details below.

2.1. Tradeoff Architecture

The tradeoff architecture module can be viewed in three tiers: *Presentation, Logic,* and *Information*. The presentation tier includes a user interface component that interacts with entities external to the tradeoff module. The logic tier includes two components: tradeoff agent and multi-perspective strategic resolution. Finally, the information tier includes ontology of attributes and concepts relevant to various stages of software development. These components work in an integrated manner with the Decision Support Toolkit. They are described in more details below.

User Interface: This component provides a means to access and manipulate the data resulting from the logic components or relevant ontology in the presentation tier. By exploiting components in the decision support toolkit (e.g., the Interactive Visualization Tool), STA can present the display and depict relevant features of different tradeoff methodologies to the users. User Interface can be implemented in various forms (e.g., web-based, station-based).

Tradeoff Agent: This logic component provides the core mechanisms for tradeoff analysis. The tradeoff agent systematically specifies the decision problem and then performs the tradeoffs. Tradeoff agent has two constituent units:

(i) *Problem Specification*
This unit is responsible for acquiring and specifying problems for the Tradeoff Analysis. The user provides information about the problem in form of goals (with respect to the software project), objectives (with respect to the stakeholders), decision criteria, alternatives and constraints along with preferences for the criteria.

(ii) *Tradeoff Analysis*
Once the problem is specified and likely alternatives are identified, this unit provides the tradeoff strategy required to objectively evaluate the options and ultimately to assist decision makers in selecting the best alternative. It includes a variety of Multiple Attribute Decision Making (MADM) methods that are well established and have been successfully used for various tradeoff applications [7]. The Tradeoff Analysis recommends the most suitable technique for analysis by using knowledge about application constraints including the software development phase in which the tradeoff is made, number of criteria and alternatives, and the objectives of the decision makers. Some generic tradeoff analysis methods incorporated in STA include the following MADM methods:

- *AHP* (Analytical Hierarchical Process) uses a hierarchical structure to define and organize criteria. AHP determines scores of each criterion by making pairwise comparisons, and the result of their aggregation gives a ranking that facilitates comparison of alternatives. See [12].
- *ELECTRE* (ELimination and (Et) Choice Translating REality) is an outranking type technique used between every pair of alternatives to arrive at preferred solutions based on two sets of comparisons called 'concordance and discordance' tests [10].
- *SMART* (Simple Multi-Attribute Rating Technique) is based on a linear additive model in which the overall value of an alternative is calculated as the total sum of the performance value of each criterion multiplied with the weight of that criterion. See more details in [4].
- *TOPSIS* (Technique for Order Preference by Similarity to Ideal Solution) relies on the principle that the proximity of the alternative to the ideal solution and the negative-ideal solution is measured as basic for tradeoff. The ideal solution is composite of the best performance values exhibited by any alternative for each criterion. The negative-ideal solution is the composite of the worst performance values [7].
- *NCIC* (Nontraditional Capital Investment Criteria) performs pair-wise comparisons of the performance gains among the criteria, for a given alternative. One of the attributes must be measured in monetary units. These comparisons are combined to estimate the monetary value attributed to each performance gain, and these values are summed to yield the overall implied value of each alternative. See more details in [3].

The details of each of the above analytical methods are beyond the scope of this paper. However, in this paper, we employ the popular AHP method to illustrate its use in the context of the STA framework for supporting tradeoff analysis in software development. Of course, there may be more than one method in the STA that is appropriate for the same decision problem.

Multi-perspective Strategic Resolution: Often stakeholder perceptions conflict during the decision making process because of the differences in the priorities of the stakeholders. Such conflicting stakeholder interests are a significant impediment to the realization and success of tradeoff decisions. The Multi-perspective Strategic Resolution provides means to correlate and integrate the rankings from all stakeholders. Depending on the tradeoff analysis methodology used, the user is presented with a set of effective techniques that incorporate inter-perspective relationships. Examples of resolution techniques include aggregation methods described in Section 3.1. *Severity Analysis* [6] can also be applied to combine the rankings by measuring the consequences and impacts of each of the decision criteria particularly to compensate with those that have not been satisfied. Which multi-perspective resolution method to use depends upon the general strategy that the decision maker perceives best fits the tradeoff at hand.

Ontology: Ontology is a specification of conceptualization [5]. It describes concepts and relationships that exist for knowledge sharing. In the context of tradeoff analysis, ontology can be a repository containing a precisely defined set of hierarchies of decision criteria and alternatives. In the context of software development, ontology can represent knowledge about the factors contributing to various stages of software development. In STA, ontology includes generic concepts about software development activities and tradeoff processes. Ontology can be used as a primary source for acquisitions of inputs required by the Problem Specification. Examples of ontology include taxonomy of designs (e.g., object-oriented, real-time, and user interface), hierarchical structures of non-functional and functional requirements, and taxonomy of criteria (e.g., technical, management). Decision makers can create customized specifications of ontology instances for specific needs through the User Interface. Ontology serves as a generic guide for reasoning about tradeoffs. This is especially useful for novice decision makers.

2.2. Decision Support Toolkit

The decision support toolkit module is designed to provide supplemental support for the basic tradeoff analysis functionality provided by the Tradeoff Architecture. There are at least three basic tools provided in the toolkit as described below.

Interactive Visualization Tool: This component includes graphical user interface for an interactive visual analysis. It allows decision makers to navigate through the solution space in order to gain insight about the decision and increase understanding of complicated tradeoff interactions and dependencies.

Sensitivity Analysis Tool: Sensitivity Analysis is crucial for tradeoff decisions. It evaluates the extent to which varying of the criteria weights and alternative scores affect the position of the alternative in the preference ranking. If the sensitivity check reveals that small changes could reverse a decision, the decision maker must reconsider his/her priorities to make the decision insensitive.

Document Generation Tool: This component provides automated documentation of the entire tradeoff activity including rationales, alternatives, decision criteria used, weight factors, option scores, and the results of sensitivity and severity analysis.

3. Tradeoff Analysis

Because tradeoff analysis is at the heart of the proposed framework, this section describes steps for tradeoff analysis provided in the STA framework.

3.1. Ranking and Aggregation

Ranking of importance and aggregation of ratings from various sources are basic necessities in decision-making techniques including negotiation and tradeoff analysis. There is a wide range of these techniques from very simple to rather sophisticated ones that are commonly used in decision science [7, 8]. Techniques for simple aggregation include arithmetic mean, geometric mean and weighted average mean. STA employs these aggregation techniques and several ranking techniques, some of which are summarized below.

- *Direct Subjective Evaluation*: directly assign weights to items to be ranked subject to expert opinions.
- *Geometric Progression:* assign the most important item to a weight of one, and the rest of 0.5, 0.25, and so on.
- *The SMART method:* assign the least important item a weight of one, and assign relative weights representing multiples of importance for each of the other items.
- *Ratio Pairwise Comparison:* items are compared in pairs using a range of weighting scale.

More details of the above techniques can be found in [8]. For an illustration in this paper, we use the weighted average mean for aggregation and the ratio pairwise comparison, a popular ranking method in AHP [12], which will be described in more details later.

3.2. Tradeoff Mechanisms

The following describes basic steps provided in the STA framework for analyzing tradeoffs in general. Each step on the tradeoff process is driven by a corresponding relevant component of STA as described in Section 2.

Step 1: Problem Specification
Acquire and specify a decision problem. In particular, identify A, a set of *alternatives*, C, a set of *evaluation criteria* for evaluating each alternative, and S, a set of *stakeholders*.

Step 2: Rating Criteria
Obtain a rating R_i^j, for each criterion $i \in C$ and stakeholder $j \in S$. This can be done by any of the ranking techniques described in Section 3.1.

For example, if the user selects the ratio pairwise comparison for the AHP analysis, STA will guide a stakeholder to create a table (matrix) $T = (t_{ij})$ for $i, j \in S$, where t_{ij} is assigned to a number k in a range of scales (e.g., [1,…,9] for the AHP's scale [12]) to signify that criterion i is k times more important than criterion j. (Note that T is reversed symmetric in the sense that $t_{ij} = 1/t_{ji}$.) Thus, we can obtain r_i, an average rating of criterion i over all criteria below. For each criterion $i \in C$,

$$r_i = \sum_{j \in C} t_{ij} \Big/ |C| \quad \text{normalized to} \quad R_i = r_i \Big/ \sum_{j \in C} r_j .$$

This is actually a rating based on one stakeholder. Thus, for each stakeholder $j \in S$, we have obtained a normalized rating R_i^j of criterion $i \in C$.

Step 3: Multi-perspective Resolution
Resolve conflicting criteria ratings (obtained in Step 2) based on different stakeholders by applying any of the aggregation methods described in Section 3.1. For example, if the decision maker selects to use the weighted average mean method by supplying p_j, a priority (influential degree) of stake holder j, then the aggregated rating of criterion i, can be specified by a weight of w_i, where

$$w_i = \sum_{j \in S} p_j R_i^j \text{ for each } i \in C.$$

In addition, *severity analysis* [6] can be performed to assess impacts of each criterion. The resulting severity rating of each criterion can be incorporated into the aggregated rating of the each criterion similarly.

Step 4: Rating Alternatives against Criteria
Obtain a score S_k^i, for each criterion $i \in C$ and alternative $k \in A$. This can be done by any of the ranking techniques described in Section 3.1. For example, if the ratio pairwise comparison for the AHP analysis is applied on a set of alternatives then, for each alternative $k \in A$, a normalized rating score S_k^i for criterion $i \in C$ can be obtained, similarly to Step 2. Score reflects the degree to which an alternative satisfies the criterion.

Step 5: Ranking Alternatives
Perform a final ranking score $RANK_k$, for alternative $k \in A$, based on the weight of each criterion (Step 3) and its corresponding rating score for the alternative (Step 4). Thus, we can compute $RANK_k = \sum_{i \in C} w_i S_k^i$.

4. Case Study

This section illustrates STA's tradeoff mechanisms by applying the AHP method to a case study of a web application development that has several stakeholders with diverse goals. The scenario has been extrapolated from [14], which focuses on tools to facilitate requirements negotiation as opposed to making tradeoff decisions.

4.1 Tradeoff Scenario

The web application development involves three stakeholders: (1) a *marketing manager* (MM) who wants the application to have an attractive graphical user interface, fast response, and be developed in a short time, (2) a *systems developer* (SD) who wants the application to be user friendly, secure, able run on any browser, and easy to maintain, and finally (3) a *programmer* (P) who wants to use the latest technology and needs sufficient time for development. Based on these agendas, six decision criteria are identified: short time to market (TTM), fancy effects (FE), high performance (HP), easy to maintain (EM), ease of use (EU), and browser independence (BI). The technology alternatives available for the development include *static*

HTML (SH), *Flash* (FL), and *JavaScript* (JS). SH offers fast development and a large degree of browser independence. FL is a new technology for this company and offers attractive graphic effects but it is hard to maintain, whereas JS, a dynamic HTML that can have problems with some browsers.

This presents a tradeoff situation where the decision criteria are in conflict. For example, implementation of fancy effects and high performance will decrease the chance of having short time to market. So while selecting the alternative, the criteria have to be balanced carefully so as to select the best possible option.

4.2 Web Application Tradeoff Analysis

Applying Step 1 of the proposed tradeoff mechanisms, we can identify a set of *alternatives* A = {SH, FL, JS}, a set of *criteria* C = {TTM, FE, HP, EM, EU, BI}, and a set of *stakeholders* S = {MM, SD, P}.

In Step 2, by using the ratio pairwise comparison method, Table 1 shows the criteria ratings for the marketing manager (MM). For example, the table entry on the TTM row and EM column of five represents the fact that the rating of TTM is five times of the EM rating. In other words, the marketing manager places importance of short time to market (TTM) as five times of the ease of software maintenance (EM). The last column of Table 1 (or first row of Table 2) gives normalized average rating of each criterion over all criteria. As expected, here the marketing manager (MM) rates time to market (TTM) the highest criterion and high performance (HP) the lowest.

C / $i \in C$	TTM	FE	HP	EM	EU	BI	R_i^{MM}
TTM	1	2	5	5	3	4	0.40
FE	1/2	1	3	3	2	2	0.20
HP	1/5	1/3	1	1	1/2	1/2	0.06
EM	1/5	1/3	1	1	1/2	1/2	0.12
EU	1/3	1/2	2	2	1	1/2	0.12
BI	1/4	1/2	2	2	2	1	0.16

Table 1. Criteria ratings of Marketing Manager (MM).

Similarly, criteria ratings from the other two stakeholders (SD and P) can be obtained (as shown in 2nd and 3rd rows of Table 2). For example, the entry in the 3rd row and 6thcolumn of Table 2 shows that for the programmer (P), browser independent (BI) and fancy effect (FE) are top two equally important criteria with a (normalized average pairwise) rating of 0.26.

C / S	TTM	FE	HP	EM	EU	BI
MM	0.40	0.20	0.06	0.12	0.12	0.16
SD	0.07	0.07	0.12	0.41	0.22	0.11
P	0.07	0.26	0.15	0.10	0.16	0.26

Table 2. Criteria ratings of each stakeholder (R_i^j's).

TTM	FE	HP	EM	EU	BI
0.235	0.192	0.099	0.142	0.152	0.180

Table 3. Criteria Weights (w_i's).

By applying Step 3, Table 3 shows aggregated mean of ratings of each criterion over all stakeholders, given the fact that the stakeholder priority p_{MM}, p_{SD}, and p_{P} is 0.5, 0.2, and 0.3, respectively. For example, w_{TTM} can be obtained by $0.4 \times 0.5 + 0.07 \times 0.2 + 0.07 \times 0.3 = 0.235$.

A / $k \in A$	SH	FL	JS	S_k^{FE}
SH	1	1/5	1/2	0.12
FL	5	1	4	0.67
JS	2	1/4	1	0.21

Table 4. Alternatives' score for Fancy Effect (FE).

By applying the ratio pairwise comparison method in Step 4, Table 4 shows alternatives' score for Fancy Effect (FE) criterion. Similarly, alternatives scores for all the criteria are calculated as shown in Table 5.

	TTM	FE	HP	EM	EU	BI
SH	0.65	0.12	0.65	0.23	0.12	0.65
FL	0.12	0.67	0.12	0.14	0.23	0.12
JS	0.23	0.21	0.23	0.63	0.65	0.23

Table 5. Alternatives' score against all criteria (S_k^i's)

By applying Step 5, the final ranking for each alternative can be computed from results in Table 3 and Table 5. For example, $RANK_{\text{SH}}$ is obtained from $0.65 \times 0.235 + 0.12 \times 0.192 + 0.65 \times 0.099 + 0.23 \times 0.142 + 0.12 \times 0.152 + 0.65 \times 0.180 = 0.40804$. Similarly, $RANK_{\text{FL}}$ and $RANK_{\text{JS}}$ is 0.24516 and 0.34680, respectively. Thus, we can infer that Static HTML (SH) is the best alternative, satisfying the project criteria and preferences from different stakeholders.

During the analysis, STA is designed to facilitate "what-if" analysis by allowing users to iterate through different inputs including ratings, or scorings and also to modify the analysis with different alternatives. The intent is for STA to support interactive tradeoff mechanisms.

5. Related Work

Tradeoffs in various stages of software development have been studied including negotiation in requirements, development, planning, and management [2, 8, 9, 11, 13, 14]. Most have focused on the need to resolve conflicts during requirements [8, 11, 13, 14]. Several techniques have been proposed. For example, the Multi-criteria Preference Analysis Requirements Negotiation (MPARN) uses preference function analysis to assess requirement conflicts amongst the stakeholders [8]. The Quantitative WinWin [11] is a similar requirement prioritization techniques based on Analytical Hierarchical Process (AHP) [12]. A tradeoff technique based on Qualitative Function Deployment (QFD) [14] is applied to resolve tradeoffs in web application development. However, it is restricted to requirement prioritization. These studies do not address tradeoff process in a structured systematic manner and mostly rely on meetings among involved parties to use tradeoffs to help negotiate and resolve conflicts as opposed. Unlike the above work, our work aims to facilitate tradeoff analysis systematically to support semi-automation.

At architectural design level, a notable Architecture Tradeoff Analysis Method (ATAM) [9] employs a risk methodology to evaluate software architecture. ATAM largely leaves tradeoff decisions to requirements negotiation. It does not aim to advance a tradeoff methodology or assist tradeoff analysis but rather to promote use of tradeoffs to select an appropriate architecture early in the software development. Thus ATAM does not share the same objective as our work. Finally, most existing techniques handle tradeoffs specific to software development decisions in an isolated manner. Each tradeoff analysis in different stages is performed separately. This makes it hard to share, reuse or integrate analysis results for a comprehensive picture. To the best of our knowledge, no integrated framework like STA has been developed to support decision making in software development.

6. Conclusion and Future Work

This paper presents a preliminary design of *Software Tradeoff Assistant* (STA), an integrated framework for tradeoff analysis that provides semi-automatic decision aids for enhancing understanding and resolving various complicated tradeoffs throughout the software development life cycle. Tradeoff analysis can be viewed as an optimization problem that involves multiple objectives or single objective with various factors and constraints. To support tradeoff analysis, STA offers a variety of analytical multi-attribute decision making methods that facilitate a sound quantitative evaluation of alternatives along with integration of preferences from multiple stakeholders. Future work includes exploring the application of artificial intelligence techniques for qualitative tradeoff analysis. By integrating various analytical decision making techniques and ontology of software characteristics into tradeoff analysis, STA offers several advantages.

First, it provides a structured process for reasoning about tradeoffs and makes tradeoff process more systematic and explicit. This helps decision makers understand complicated tradeoff interactions and dependencies to arrive at their decisions. Second, by encoding software ontology and employing generic tradeoff methodologies, STA provides tradeoff mechanisms that can be applied and reused in multiple contexts (e.g., different stages of software development) and multiple domains (e.g., medical or engineering software applications). Finally, having an integrated environment for analyzing tradeoffs at various stages of software development, STA provides flexibility in choosing appropriate techniques and facilitates integration of results from different tradeoff methods. Currently, additional issues on control mechanisms in STA and the development of a prototype based on the STA framework are our ongoing study.

References

[1] Amandeep A., Ruhe G., and Stanford M. Intelligent support for software release planning. In *Product Focused Software Process Improvement, PROFES 2004*, volume 3009, pp. 248–262, 2004.

[2] Boehm B. *Value-Based Software Engineering: Overview and Agenda.* USC-CSE-2005-504, University of Southern California, Dept. of Computer Science, 2005.

[3] Boucher T. and MacStravic E. L. Multi-attribute evaluation within a present worth framework and its relation to analytic hierarchy process, *The Engineering Economist*, 37(1): 1–32, 1991.

[4] Edwards W. How to use multiattribute utility measurement for social decision making, *IEEE Transactions on Systems, Man and Cybernetics*, volume SMC-7, pp. 326–340, 1977.

[5] Gruber T. A translation approach to portable ontology specifications, *Knowledge Acquisition*, 5(2):199–220, 1993.

[6] Haimes Y. *Risk Modeling, Assessment, and Management.* John Wiley & Son, Inc., Hoboken, New Jersey, 2004.

[7] Hwang C. and Yoon K. *Multiple Attribute* Decision *Making*. Springer-Verlag, Berlin, 1981.

[8] In H., Olson D., and Rodgers T. Multi-criteria preference analysis for systematic requirements negotiation, *Computer Software and Applications Conference, COMPSAC2002*, pp. 887– 892, 2002.

[9] Kazman R., Klein M., Barbacci M., Longstaff T., Lipson H., and Carriere J. The architecture tradeoff analysis method. In *Engineering of Complex Computer Systems. ICECCS '98*, pp. 68–78, 1998.

[10] Roy B. The outranking approach and the foundations of ELECTRE methods, *Theory and Decision*, 31(1):49–73, 1991.

[11] Ruhe G., Eberlein A., and Pfahl D. Quantitative win-win: a new method for decision support in requirements negotiation. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pp. 159–166, 2002.

[12] Saaty T. *The Analytic Hierarchy Process*. McGraw Hill, New York, 1980.

[13] Yen J. and Tiao W. A. A systematic tradeoff analysis for conflicting imprecise requirements. In *RE '97: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering*, pp. 87–96, 1997.

[14] Ziemer S. and Stålhane T. The use of trade-offs in the development of web applications. In *ICWE Workshops*, pp. 269–281, 2004.

Improving Separation of Concerns in the Development of Scientific Applications

S. M. Sadjadi*, J. Martinez, T. Soldo, L. Atencio
School of Computing and Information Sciences
Florida International University, Miami, FL, U.S.A
{sadjadi,ftrig001}@cs.fiu.edu

R. M. Badia, J. Ejarque
Barcelona Supercomputing Center
Barcelona, Spain
{rosa.m.badia, jorge.ejarque}@bsc.es

Abstract

High performance computing (HPC) is gaining popularity in solving scientific applications. Using the current programming standards, however, it takes an HPC expert to efficiently take advantage of HPC facilities; a skill that a scientist does not necessarily have. This lack of separation of concerns has resulted in scientific applications with rigid code, which entangles non-functional concerns (i.e., the parallel code) into functional concerns (i.e., the core business logic). Effectively, this tangled code hinders the maintenance and evolution of these applications. In this paper, we introduce Transparent Grid Enabler (TGE) that separates the task of developing the business logic of a scientific application from the task of improving its performance. TGE achieves this goal by integrating two existing software tools, namely, TRAP/J and GRID superscalar. A simple matrix multiplication program is used as a case study to demonstrate the current use and capabilities of TGE.

Keywords
Grid Enablement, Transparent Shaping, GRID superscalar

1. Introduction

The advent of cluster and grid computing has created a remarkable interest in high performance computing (HPC) both in academia and industry, especially as a solution to complex scientific problems (e.g., hurricane path prediction). To efficiently utilize the underlying HPC facilities using the current programming models and tools, however, a scientist is expected to develop complex parallel programs; a skill that she might not necessarily have and is better done by an HPC expert.

Current standards for cluster and grid programming such as MPI [8], OGSA [9], and WSRF [10] (and their implementations such as MPICH2 [11], Globus Toolkit [12], Unicore [13], and Condor [14]; to name just a few) have provided scientists with higher levels of abstraction. Noteworthy, these approaches have been successful in hiding the heterogeneity of the underlying hardware devices, networking protocols, and middleware layers from the scientist developers. However, the scientists are still expected to develop complex parallel algorithms and programs. Moreover, as the code for parallel algorithms would typically crosscut the code for business logic of the application, the resulting code will be an entangled code that is difficult to maintain and evolve.

In this paper, we introduce Transparent Grid Enabler (TGE) that addresses these problems by enabling a *separation of concerns* in the development and maintenance of the non-functional concerns (i.e., the parallel code) and the functional concerns (i.e., the business logic) of scientific applications. TGE achieves this goal by integrating two existing programming tools, namely, a Grid framework, called *GRID superscalar* [2], and an adaptation-enabling tool, called *TRAP/J* [6].On one hand, GRID superscalar enables the development of applications for a computational Grid by hiding details of job deployment, scheduling, and dependencies and enables the exploitation of the concurrency of these applications at runtime. On the other hand, TRAP/J supports automatic weaving of alternative parallel code (including the corresponding calls to GRID superscalar runtime) into the sequential code developed by the scientist.

TGE increases the level of modularity of code by separating crosscutting grid related code from the business logic of the application. This allows scientists to continue focusing only on the core logic of the scientific applications, leaving the parallel code and its complexity to the HPC experts. In TGE, the grid enablement or weaving of parallel code into the original application is called to be transparent, because all the grid enablement process occurs automatically with no manual modifications to the business logic of the application and hence "transparent" to the scientist and her sequential code. This way, TGE supports transparent grid enablement of existing scientific applications also.

The rest of this paper is organized as follows. In Section 2, we provide a short background on GRID superscalar and TRAP/J. In Section 3, we introduce a simple case study, called "Matmul", which is a matrix multiplication program. In Section 4, we show how TGE works by adapting Matmul to run on a computational grid. In Section 5, we provide some experimental results and demonstrate the speedup gained because of grid enablement. In Section 6, we discuss some related works and in Section 7, we provide some future research directions. Currently, TGE enables only static grid enablement of Java programs by means of configuration files at startup time. We are planning to provide dynamic grid enablement as well as self-management behavior to scientific applications at run time. Finally, we finish the paper in Section 7 after providing some concluding remarks.

2. Background

Transparent grid enablement is achieved by the combination of TRAP/J and GRID superscalar. Therefore, as a first step, we present a brief background information about both technologies. For more detail, please refer to the references.

2.1 GRID superscalar

Inspired by the *superscalar* processors, GRID superscalar provides an easy programming paradigm for developing parallel programs [2]. Similar to superscalar processors that provide out-of-order and parallel execution of machine instructions by bookkeeping their dependencies, GRID superscalar provides parallelism to the functions of a program written in a high-level programming language such as Java. Using GRID superscalar, a sequential scientific application developed by a scientist is dynamically parallelized in a computational Grid. GRID superscalar hides the details such as resource mapping, staging input data files, cleaning temporary data files, task deployment, task scheduling, exploiting instruction-level parallelism, and exploiting data locality. We note that for many of its responsibilities, GRID superscalar depends on other grid computing toolkits such as GT4 [12], Condor [14], and others.

2.2 TRAP/J

TRAP/J is a tool that enables static and dynamic adaptation in Java programs at startup and runtime, respectively [6]. It consists of two GUI-based interactive tools as follows: (1) *Generator*, which generates an adapt-ready version of an existing application by inserting generic hooks into a previously selected subset of classes in the application; and (2) *Composer,* which allows insertion of new code at the generic hooks both at startup or runtime. We note that only the pre-selected classes are capable of being adapted and they are called *adaptable* classes. Adaptable behavior is provided through alternative implementation of adaptable classes, which are called *delegate* classes. To replace alternative parallel algorithms developed using the GRID superscalar codes, we use the Generator to make the classes with sequential code adaptable, and then we use the Composer to weave in the parallel code.

3. Case Study: Matmul

Our case study is a simple application, called Matmul, which is a matrix multiplication program written in Java. It uses a sequential matrix multiplication algorithm, which computes C = A.B, where A, B, and C are matrices of size NxN. It uses the classic algorithm of "rows by columns" multiplication. This algorithm involves O(N^3) operations.

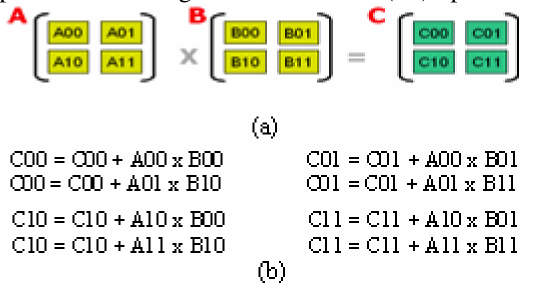


Figure 1: Hyper-Matrix Multiplication.

We will use TGE to make this application grid enabled. First, we use GRID superscalar to develop alternative hyper-matrix multiplication algorithms by splitting the original matrices into a number of sub-matrices or blocks as shown in Figure 1 (a). Then we multiply these sub-matrices accumulatively as shown in Figure 1 (b). GRID superscalar will exploit the task-level parallelism by resolving the dependencies of the tasks as shown in Figure 1 (b). Therefore, instead of just one task as in the original approach, using hyper-matrix multiplication and GRID superscalar, up to 4 tasks can be active at the same time. Similarly, if we split the matrix into 9 blocks, then up to 9 tasks can be executed at the same time and so on and so forth.

4. Grid Enablement of Matmul

We begin with the snippet code of the simple sequential matrix multiplication program that is shown in Figure 2. The bold method method in Figure 2 performs a conventional row by column matrix multiplication. The statement underlined saves the result of the multiplication in the specified file.

```
public static void main(String[] args)
{       . . .
Multiply_Matrices(size, args[1], args[2],
args[3]);
//args[] contains the names of the files
//containing the input matrices
}
public  static  void  Multiply_Matrices(int
size, fileC, fileA, fileB)
{ Block A = new Block(fileA, size);
  Block B = new Block(fileB, size);
  Block C = new Block(size);
  C.Multiply(A,B);
  C.blockToDisk(fileC);
}
```

Figure 2: Original Matrix Multiplication Code

In order to run this application on the grid we will use TRAP/J to weave in the parallel code developed at startup time into this application and use GRID superscalar to run the grid enabled adapted program. We select the Multiply_Matrices method to become adaptable since this method has been identified as the computationally intensive part of the original application. Therefore, a delegate class is developed that re-implements this method using the hyper matrix multiplication algorithm and GRID superscalar.

As shown in Figure 1 (b), since the calculation of each block in the resulting matrix (*C*) is independent of the other ones, they can be executed in parallel, potentially on different processors of a grid computing environment. Figure 3 displays the code for a delegate class, Matmul_Del.java, that was implemented for this case study and includes the Multiply_Matrices method. The beginning of the method (not shown here for simplicity) takes care of splitting the original matrix operands *A* and *B* into blocks, creating files where each block is saved, and creating the files that will store the result of matrix multiplication for each block.

```
public class Matmul_Del implements Delegate
    Interface
{  public static void Multiply_Matrices(int
    size, fileC, fileA, fileB)
  { . . .
   GSMaster.On();
   for(int i=0;i<num_of_pieces;i++)
   {  //Split in 4 pieces-
     for(int j=0; j<num_of_pieces;j++)
     {
      for(int k=0; k<num_of_pieces;k++)
      {  //Sending to Grid
         Matmul.multiply_acc(C[i][j],
               A[i][k],B[k][j],size/
               num_of_pieces);
      }
    GSMaster.Off();
     . . .
    MergeFiles();
}
```

Figure 3: Delegate Class for Multiply_Matrices method

The underlined section of the code calls the matrix multiplication method, Matmul.multiply _acc() for each pair of corresponding blocks. This is the method that allows for parallelism by being executed in separate tasks (possibly running on different nodes). In order for GRID superscalar to know that Matmul.multiply_acc() is the method to be deployed on worker nodes, several steps must be taken. First, an IDL file must be created as shown in Figure 4 to specify the signature of this method.

```
interface MATMUL
{    void  multiply_acc(inout  File  f3,
in File f1, in File f2, in int size);
};
```

Figure 4: Matmul IDL file

This part is much like a CORBA IDL file that is used to generate stubs and skeletons to be used for a remote procedure call. The IDL uses the special keywords “in”, “out”, and “inout” to specify the type of the parameters to be read, written or both, respectively. Using this IDL as input to GRID superscalar, we generate the “worker” versions of Matmul; and using the selected adaptable method as input to TRAP/J, we generate the “master” version of Matmul.

When we execute the master program, it calls the Multiply_ Matrices method, which will be intercepted by the TRAP/J runtime and will forward the control to the code in Figure 3; effectively the parallel code will be executed instead of the original sequential code. As a first step, GRID superscalar will be started with a call to GSMaster.On(), basically to initialize resources in the grid, like for example Globus services. Later, each multiplication of the sub-matrices will be sent to the nodes in the grid using the Matmul_multiply_acc(…). After all the calls to this method for all the multiplications are done, GRID superscalar is disabled by the call to GSMaster.Off(). Finally GRID superscalar runtime will collect the results using the mergeFiles() method, which is in charge of merging the individual output files obtained from the different block multiplications into one matrix file representing the result of the matrix multiplication.

The class Matmul on which the static method Multiply_Accumulative() is invoked is actually provided by GRID superscalar when deploying the application. Basically what it does is to call the GRID superscalar runtime in order to execute the method, Multiply_Accumulative() already defined in the IDL we created at startup. As mentioned before, all the issues related to file handling, concurrency problems, and interactions with the grid (middleware like Globus in this case) are handled by GRID superscalar.

Finally, the obtained grid-enabled application offers the choice for the user to choose among different alternative parallel-computing algorithms; for example, choosing between an algorithm which uses 4 blocks or 9 blocks. The decision of choosing one algorithm or another can be made based on the number of resources available and therefore, taking advantage of the grid infrastructure properly. Moreover, this new grid-enabled application is transparent to the user in the sense the way it was originally executed remains the same.

5. Experimental Results

The case study we discussed on the sections above left us with some interesting results that we present in this section.

First, we should point out that even though our approach takes advantage of parallel programming when using the computational grid, we also face the problem of delays caused by the network traffic, coordination of tasks, and the middleware software services used (Amdahl's law). Therefore, it is reasonable to predict that when the matrices to multiply present a relatively small dimension, the original sequential application will perform faster than the grid-enabled one. As the matrix size increases we will be able to see that this difference in time shortens progressively.

Matrix Size (*N*)	Sequential (ms)	Parallel with 4 blocks (ms)	Speedup (S/P)
144	674	61512	0.010957212
288	2031	66096	0.030728032
576	9527	69365	0.137345924
1152	62269	172787	0.360380121

Table 1: Initial time results

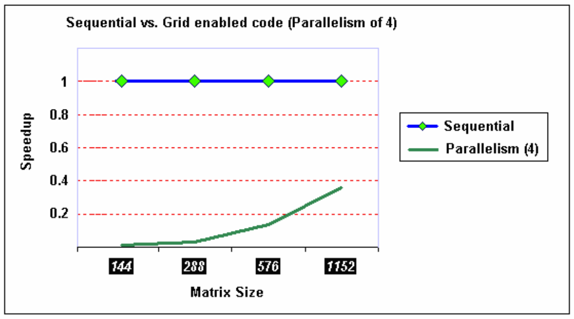


Figure 5: Speedups of the experiments

Table 1 shows the initial experiments we ran, and there you can compare the results obtained by both the original sequential program versus our approach using four nodes of the grid and therefore having a level of parallelism of 4. In this first set of experiments, we noticed that the sequential code always ran faster than the grid-enabled one; however, as the matrix size increased, we could notice that the difference in time between the two approaches became smaller and smaller.

One of the main problems we had by then with the performance was due to the way GRID superscalar works. The method GS_Off(), mentioned in section 4 and in figure 3, is in charge of freeing resources and deleting temporary files after finishing the calls to the grid method and since all the data is distributed along the nodes, then a cleanup was needed everywhere causing our application to take extra time.

Since we wanted to get a more optimized grid-enabled application, we took the GRID superscalar source code and optimized it, removing the cleanup but keeping the main functionality so that we can still get consistent results. Applications that benefit from HPC are usually scientific applications like, for example, those related to hurricane prediction and monitoring. In such cases, temporary left over data is irrelevant if the application provides us with the correct results quickly. Therefore, the approach we took at this stage seems proper.

With this optimization in hand, we also decided to implement a new algorithm in which we handled a parallelism of 9. Due to infrastructure reasons at the time, the number of nodes available for this experiment was at most 6. The results in terms of times consumed for this set of new experiments are shown in Figure 6 and 7.

Matrix Size (*N*)	Seq. (ms)	Par. w/ 4 blocks and 2 workers (ms)	Par. w/ 4 blocks and 4 workers (ms)	Par. w/ 9 blocks and 6 workers (ms)
144	5576	79221	57656	145331
288	14934	86259	62013	146744
576	44755	108107	78096	148240
1152	19318	176464	133058	176464
2304	79837	643925	441891	474215

Table 2: Final time results

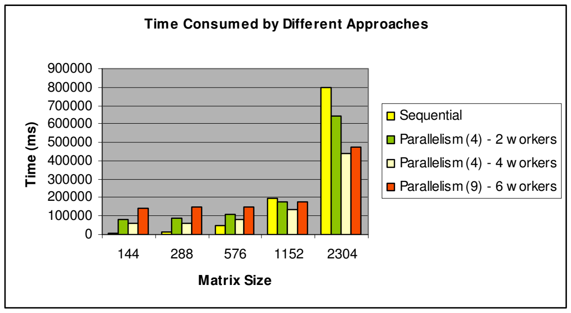


Figure 6: Chart of the times for each approach

With the results (time in ms) obtained in Table 2 and Figure 6 we were able to build a table and a graph showing the Speedups of each algorithm as shown in table 3 and figure 7.

In Figure 7 we see that as the matrix size increases, the speedup improves and finally when the size of the matrix is 1152 (number of rows = number of columns = 1152), all of the algorithms for the grid-enabled application perform better than the original sequential application. Furthermore, when the size of the matrix is 2304, all of the algorithms perform even much better than the sequential one. As a result, we notice that the algorithm with best performance is the one that uses parallelism of 4 and 4 nodes which for a matrix of size 2304 performs almost twice faster than the sequential one. This is because we have more CPU power and we are using all of it because of the parallelism of 4. Besides that, having only 4 nodes, reduces the number of file transfers among the nodes, which in turn reduces time of execution.

Matrix Size	Seq	Parallelism (4) 2 workers	Parallelism (4) 4 workers	Parallelism (9) 6 workers
144	1	0.0703858	0.09671153	0.03836759
288	1	0.1731298	0.24082048	0.10176907
576	1	0.4139879	0.57307673	0.30190907
1152	1	1.0947502	1.45187813	1.09475020
2304	1	1.2398462	1.80670799	1.68355704

Table3: Speedups of each approach

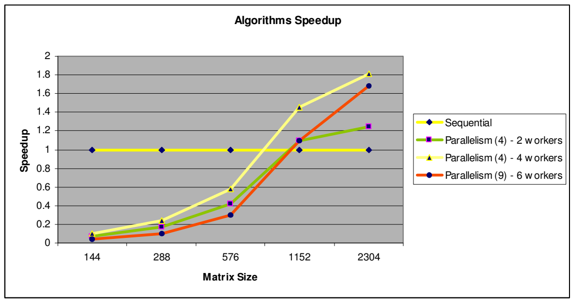


Figure 7: Speedups of each approach

We emphasize that the experiments are part of our ongoing research activities and by no means they are meant to be representative and conclusive with respect to providing a quantitative metric for speedup of sequential applications. The main purpose of these experiments is to show that we were able to use the current prototype of TGE to transparently adapt an application to run on a grid computing environment.

6. Related Work

Other approaches that enable programming parallel applications for computational Grids are Satin, HOCS, ProActive or ASSIST.

Satin [7] is a Java based programming model for the Grid which allows to explicitly expressing divide-and-conquer parallelism. Satin uses marker interfaces to indicate that certain invocation methods need to be considered for potentially parallel (spawned) execution. Moreover, synchronization is also explicitly marked whenever it is required to wait for the results of parallel method invocations.

HOCS [5] is a component oriented approach based on a master-worker schema. Higher-Order Components (HOCs) express recurring patterns of parallelism that are provided to the user as program building blocks, pre-packaged with distributed implementations.

ASSIST [1] is a programming environment aimed at providing parallel programmers with user-friendly, efficient, portable, fast ways of implementing parallel applications. It includes a skeleton based parallel programming language (ASSISTcl, cl stands for coordination language) and a set of compiling tools and run time libraries. The ensemble allows parallel programs written using ASSISTcl to be seamlessly run on top of workstation networks supporting POSIX and ACE (the Adaptive Communication Environment, which is an extern, open source library used within the ASSISTcl run time support).

ProActive [3] is a Java GRID middleware library for parallel, distributed and multi-threaded computing. With a reduced set of simple primitives, ProActive provides a comprehensive API to simplify the programming of Grid Computing applications: distributed on Local Area Network (LAN), on clusters of workstations, or on Internet GRIDs. ProActive is only made of standard Java classes, and requires no changes to the Java Virtual Machine, no preprocessing or compiler modification, leaving programmers to write standard Java code. Architected with interception and reflection, the library is itself extensible, making the system open for adaptations and optimizations. Current implementation is focusing of the CoreGRID NoE specification of the Grid Component Model (GCM) [4].

None of the above mentioned approaches provide an explicit separation of concerns identifying separate tasks for scientist developers and HPC expert developers. TGE can be extended to use these works instead or in complement to GRID superscalar and can be used as an enabler for supporting interoperation among the above mentioned approaches.

7. Future Work

As we mentioned before, we have been able to achieve *static* adaptation. Our next task will be to extend TGE in support of more autonomic behavior and include adaptation at runtime (dynamic) in response to high level system policies such as the addition of more nodes to the grid, process scheduling, etc, or application level policies such as different blocking algorithms, faster algorithms, etc.

At present, dynamic adaptation of Java programs with TRAP/J has been achieved and tested. However, running an application in a grid environment inherently introduces a lot more challenges than just running the program in one node or virtual machine. If we take Matmul as an example, we can clearly see that switching the blocking algorithm dynamically requires us to save the present state of calculations, adapt it to the new algorithm, and continue executing.

Furthermore, moving towards building a more autonomic self adapting and self configuring system, we can take TGE to provide context-aware adaptation. In other words, by invoking the Globus Toolkit monitoring service we can keep track of the state of the runtime environment and retrieve information about resource allocation, scheduling, etc.

8. Conclusion

In this paper we have presented an innovative approach to transparent grid-enablement of scientific applications. We achieved this goal by combining TRAP/J and GRID superscalar. Each tool provided us with the necessary features for transparent software adaptation from a sequential code to a grid-enabled one as Figure 8 briefly sums up.

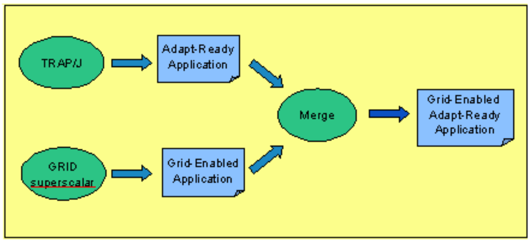


Figure 8: TGE Flow Diagram

The matrix multiplication shown as a case study in this paper is just a simple example to show how our approach works. In fact, this matrix multiplication, for example, could be just one part of a whole application and be the portion of the code that consumes most of the execution time and in that sense, applying our approach would considerably benefit the whole application's performance. In a similar fashion, we could take an existing application and just modify the part of the algorithm that is in charge of the most cpu utilization and just parallelize that logic without having to modify the total original code.

Another important issue to mention is that the optimization discussed in the paper is not targeted only to improve the performance of our case study, in fact, since this optimization has been done to the Grid Superscalar library, any application built from now on will benefit for these improvements. Moreover, even though we took Java as the programming language to describe our case study, other programming languages could be used following the same approach since for example C and Perl are also supported by Grid Superscalar.

Finally, we are aware that we cannot guarantee that in all applications we will be able to separate the parallelism of a portion of the algorithm from the business logic of it; however, there many existing applications that do offer this facility.

9. Acknowledgement

This work was supported in part by IBM (SUR and Student Support awards), the National Science Foundation (grants OCI-0636031, REU-0552555, and HRD-0317692), the Spanish CICYT (contract TIN2004-07739-CO2-01), and the BSC-IBM Master R&D Collaboration agreement. This work is part of the Latin American Grid (LA Grid) project.

References

[1] Marco Aldinucci, Massimo Coppola, Marco Danelutto, Marco Vanneschi, and Corrado Zoccolo. *Assist as a research framework for high-performance grid programming environments*. In Jose C. Cunha and Omer F. Rana, editors, Grid Computing: Software environments and Tools. Springer-Verlag, 2004.

[2] Rosa M. Badia, Raül Sirvent, Jesus Labarta, and Josep M. Perez. *Programming the GRID: An Imperative Language Based Approach*. book chapter in Engineering the Grid, Section 4, Chapter 12 , January 2006.

[3] Laurent Baduel, Françoise Baude, Denis Caromel, Arnaud Contes, Fabrice Huet, Matthieu Morel, and Romain Quilici. *Programming, Composing, Deploying for the Grid* (the reference to be used to cite ProActive), in "GRID COMPUTING: Software Environments and Tools", Jose C. Cunha and Omer F. Rana (Eds), Springer Verlag, January 2006.

[4] CoreGRID Deliverable D.PM.02, 2006, Proposal for a Grid Component Model.

[5] Sergei Gorlatch and Jan Dünnweber. *From Grid Middleware to Grid Applications: Bridging the Gap with HOCs*. In Future Generation Grids, Springer Verlag, 2005.

[6] S. Masoud Sadjadi, Philip K. McKinley, Betty H.C. Cheng, and R.E. Kurt Stirewalt. *TRAP/J: Transparent generation of adaptable Java programs.* In Proceedings of the International Symposium on Distributed Objects and Applications (DOA'04), Agia Napa, Cyprus, October 2004.

[7] Rob van Nieuwpoort, Jason Maassen, Thilo Kielmann, and Henri E. Bal. *Satin: Simple and efficient Java-based grid programming*. Scalable Computing: Practice and Experience, 6(3):19-32, September 2005.

[8] http://www-unix.mcs.anl.gov/mpi/

[9] http://www.globus.org/ogsa/

[10] http://www.globus.org/wsrf/

[11] http://www-unix.mcs.anl.gov/mpi/mpich2/

[12] http://www.globus.org/toolkit/

[13] http://www.unicore.org/

[14] http://www.cs.wisc.edu/condor/

Pattern-based J2EE Application Deployment with Cost Analysis

Nuyun ZHANG, Gang HUANG, Ling LAN, Hong MEI
Institute of Software, School of Electronics Engineering and Computer Science, Peking University，
Key laboratory of High Confidence Software Technologies (Peking University)，Ministry of Education，
Beijing, 100871, China
zhangny04@sei.pku.edu.cn, huanggang@sei.pku.edu.cn, lanling@pku.edu.cn, meih@pku.edu.cn

Abstract

The most challenging problem of J2EE application deployment is to determine which components should be deployed onto which servers in terms of non-functional properties. This paper proposes an empirical approach with some mathematic enhancement. It does two contributions to J2EE application deployment: 1) patterns are adopted to specify why, when, where and how to deploy so that the empirical approach becomes more comprehensible and operational; 2) a mathematic framework to analyze the deployment cost is defined for helping the selection of the best-of-breed pattern. We experiment several patterns on a J2EE benchmark application and evaluate the cost analysis framework.

1 . Introduction

Deployment of J2EE application is the process to deploy application clients, applets, web components and Enterprise JavaBeans components into specific operational environments [6].

The most challenging problem of deployment is component arrangement, namely which components should be deployed onto which servers in terms of performance, reliability, cost and other non-functional properties. Existing approaches for the problem can be summarized as: mathematics, artificial intelligence (AI) and empirical methods.

The mathematics methods build delicate models about the system and solving it to find the optimal deployment plan [4]. They require precise parameters, such as memory consumption of each component and reliability of each node. The AI methods use AI technology to reduce the complexity of the search space that is involved in finding an optimal plan [5]. They need a number of parameters, too. The empirical methods try to conclude some guidelines from experiences or best practices for deployments.

The math and AI methods neglect some details of the running of the application system. In addition, some of the parameters they need are difficult to measure or their precision cannot be guaranteed. Totally depending on them is usually impractical. On the other hand, when turning to the empirical methods, we find their carrying out is seriously depend on the deployers' understanding and manual operations.

This paper proposes a pattern based approach for J2EE application deployment. It is an empirical method but more formal and specific than the traditional ones. It includes clear operational steps and can be carried out automatically. However, there are 2 problems in the approach: how to choose a pattern to execute from several useful patterns and how to map the deployment solutions onto physical machines. We again propose a deployment cost analysis framework to meet the problems.

The contribution of this paper is: it not only brings forward a pattern based approach to J2EE application deployment but also makes it practical. It makes use of patterns to describe deployment guideline and gives a quantitative cost analysis framework to support the execution of patterns. It implements the pattern based deployment process and validates the effectiveness of the approach and the framework by experiments.

2 . Approach Overview

2.1 Description of Deployment Pattern

The description of deployment patterns contains several sections which are shown in Table 1.

2.2 Pattern based Deployment Process

We divide J2EE deployment process into seven stages that are release, installation, update, adaptation, activation, deactivation, and uninstallation. The pattern based J2EE deployment adds some steps between the stages as shown in Figure 1.The execution of pattern starts after the configuration step. Then it goes on as follows:

Firstly, the pattern execution tool identifies the pattern candidates by checking the goal sections and context sections of all patterns.

Secondly, the pattern execution tool picks out one of the pattern candidates to execute. It follows the solution section to make a deployment plan, the implementation section to make a physical deployment plan.

This work has been supported by the National Grand Fundamental Research 973 Program of China under Grant No.2005CB321805, the National Natural Science Foundation of China under Grant No. 90412011, 90612011, 60403030.

Table 1 Description of Deployment Pattern

Section	Purpose	Content	Examples
Name	Unique identification for the pattern	A meaningful name	Components collocating pattern
Goal	For automatic recognition of the pattern	WHY: The goal of deployment	short response time.
Context	For automatic checking of the pattern	WHEN: Preconditions for using the pattern	Workloads, resource constraint of nodes.
Solution	For automatic execution of the pattern	WHERE: Relationships between components and nodes	Component A is on node n. Component B is on node m.
Implement ation	For automatic execution of the pattern	HOW: Relationships between components and physical machines	Component A is on node n. Component B is on node m. Node n is Dell PC 1. Node m is Dell PC 2.

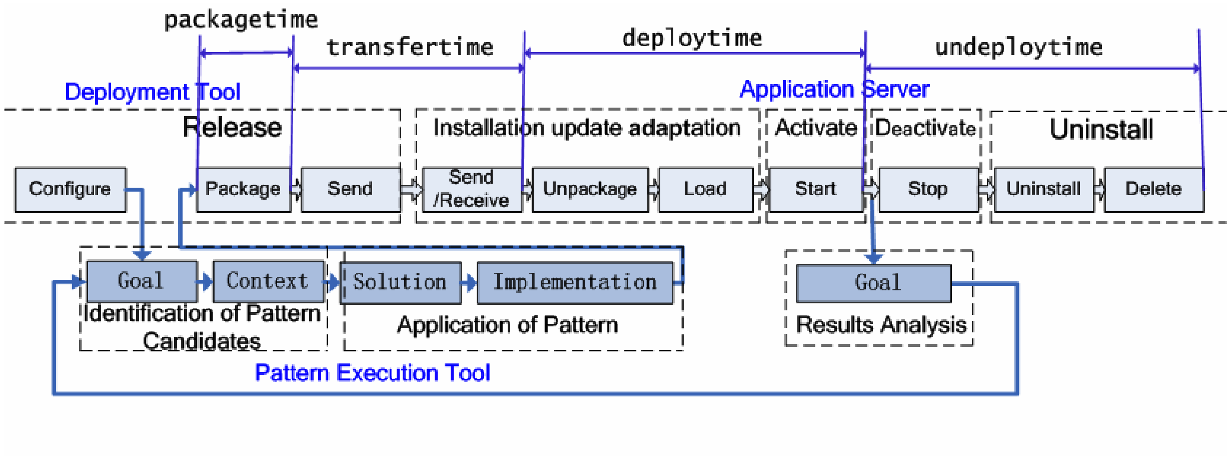


Figure 1. Pattern based deployment process

Thirdly, the deployment tool packages the components and transfer them to their destinations according to the physical deployment plan.

Fourthly, the application servers install the packages and start them. If this is a redeployment, the application servers will stop and uninstall the old components before the installation and starting.

Finally, after a period of running, the tools check whether the deployment has the expected results and adjust the parameters in the patterns. The tools may start a new round of pattern based deployment process to improve the deployment.

3 .Deployment Cost Analysis Framework

The challenges in pattern based deployment process are: 1) in the implementation section of a pattern, we should map the solution to the physical machines. 2) If useful patterns are more than one, we should select one for execution under circumstances that we do not know which pattern will make the system work most well.

There could be many kinds of solutions to the challenges. We consider in an enterprise application like J2EE application that the Cost/Benefit is most important. Since we have no idea of how many benefits deployment will bring, we try to make the cost of deployment minimum. In the deployment process, the service is stopped and that adds to the cost. Therefore, we proposed a framework to evaluate the cost of J2EE application deployment process.

We define the deployment cost as the time deployment takes. In the deployment or redeployment process, loses are brought by the system down time. So we care about it instead of the other expenses such as memory or network occupation.

Our goal is to find the pattern whose deployment time is the least. For this purpose, the cost analysis framework only care about the cost differences in deployment processes, not their absolute values. Some constants will be neglected to simplify the framework.

Packaging and transferring do not interrupt the running of the application system. We do not include them in deployment cost analysis.

The analysis framework is as follows:

First, we define a package as a group of components placed on the same node. The deployment process of an application system is composed of the concurrent deployment processes on all the nodes in the system. The deployment time is determined by the slowest sub-process. We describe it as:

$$CostOfApplication = \max\{costOnEachNode\} \qquad ①$$

The deployment cost on a node is composed of undeploy time of old packages and deployment time of the new ones, as shown in equation ②.

$$costOnEachNode = deploymentTime + undeploymentTime \quad ②$$

1. The deployment time is closely related with the implementation of application server. We have done some experiments on Peking University Application Server (PKUAS) [3] try to find some rules in deployment time.

The deployment time is composed of installation time and starting time. The following hot color map is a contour figure showing the installation time of JAR packages. The 20 colors evenly ranged from 3396 to 17626 ms. The darkest red in the upper right area of the map represents the longest time that is 17626 ms. The darkest blue in the lower left represents the shortest time that is 3396 ms. There is a blank area in the right lower part of the figure, because when package number is small, the package size can hardly be very large.

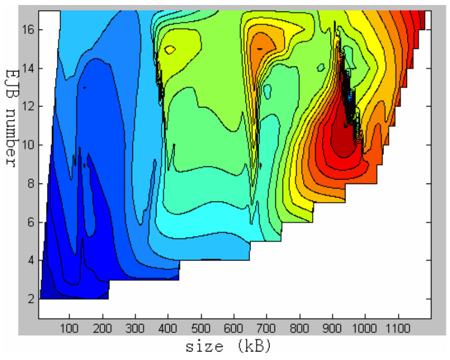


Figure 2. Contour map of installation time of JAR

From figure 4 we can see the installation time has something to do with package size and number of JARs. The more JARs are, the longer the time is. The larger the size is, the longer the time is. Because the contour lines are almost vertical, we know the size of package plays a more important role in the installation time and their relationship can be approximately described by linear equation.

In our cost analysis problem, what we want is only a comparable result, not the absolute number. To simplify the comparison, we give each installation time a reasonable value instead of its real value. The given value can reflect its real value in proportion. The simplified expression of installation time of EJB package should be in a linear form, and we omit the constant to get the following expression:

$$installationTimeEJB = \left\lfloor \frac{ejbpackageSize(kB)}{100} \right\rfloor \quad ③$$

If necessary, by curve fitting and further analysis of the sub deployment time，we can get the exact relationship of installation time, the JAR number and the package size.

We can give a set of test packages. In another specific environment, by deploy the test packages, we can figure out the relationship of installation time, the JAR number and the package size automatically.

2. We do the same kind of experiments to figure out the installation time of WAR packages. In our experiments, the installation time of WAR differs from 282 to 33829 ms as the package size ranges from 41 to 64634 kB. The installation time of WAR is about ten times less than that of JAR of the same size.

We find that no matter how many html, JSP, and images are in the package, the installation time is the same if the whole package size is the same and the Java class number is the same. Figure 3 shows the installation time of WAR when there are a fix number of Java classes. We can see that the installation time has a linear relationship with the size of WAR package.

$$installationTimeWAR = c * WARsize$$

where c is a constant that has something to do with the implementation of the application server and the Java classes in the package.

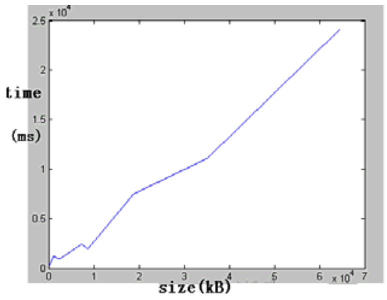


Figure 3. Installation time of WAR

For the purpose of comparison, consulting the experiment results of the installation time of EJB package, we give a simplified expression for WAR installation time:

$$installationTimeWAR = 0.1 * \frac{WARsize}{\sum JARsize} \quad ④$$

where $\sum JARsize$ represents the size of all the JARs in the system.

Integrating the expression ③ and ④ we get the installation time of a package as:

$$installationTime = \left\lfloor \frac{ejbpackageSize}{100} \right\rfloor + 0.1 * \frac{WARsize}{\sum JARsize} \quad ⑤$$

3. By experiments we know the starting time is really small and can be neglect.

4. The undeployment time is composed of the uninstallation time and the stopping time. By experiments we know the stopping time is very small and can be neglected.

5. In our experiments, the uninstallation time of WAR varies from 188 to 781 ms as the package size ranges from 41 to 64634 kB. It plays a so trifling part in the whole deployment time that we neglect it.

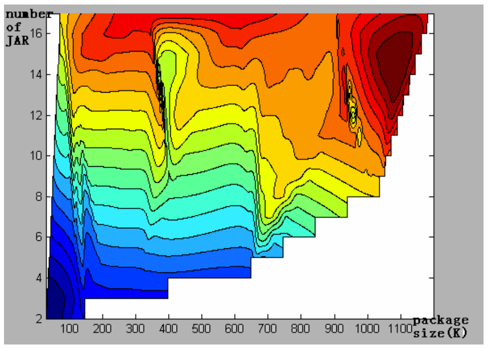


Figure 4. Uninstallation time of JAR

6. Figure 4 is the contour map of uninstallation time of JAR packages by experiments. Because the contour line is almost horizontal, we know the number of JAR is the most important factor in uninstallation time. Using the same analysis method of the installation time, we get the following expression:

$$uninstallationTimeJAR = 0.2 * \ JARnumber \qquad ⑥$$

In conclusion, applying the expression ⑤ and ⑥ to ②, we obtain the deployment cost on each node. Applying the expression ② to ①, we obtain the cost of the whole deployment process.

In different implementations of J2EE deployment process, the cost analysis methods may need adjustment in detail, but the framework is still applicable and it is our future direction of research.

4 .Case study

In this section, we use cost analysis based deployment patterns to redeploy RUBiS [7], which is an auction site prototype modeled after eBay.com that is used to evaluate application design patterns and application server performance scalability. First, we give an entire redeployment process of RUBiS. Then, we verify our cost analysis framework by comparing experimental deployment time and its estimated values. The process is aided by CADTool, which is a J2EE deployment tool [2].

The environment of experiments is shown in Table 2:

Table 2. Deployment environment

Node	Software	Database	CPU	Memory
Aster0	Windows XP + PKUAS	none	2.79G	504M
Aster2	Windows XP + PKUAS	MySQL 4.0.15	2.99G	504M
Aster3	Windows XP + PKUAS	none	2.79G	512M
Client	Windows XP	none	2.79G	512M

4.1 Redeployment of RUBiS

4.1.1 Initial Deployment of RUBiS

In the given environment, there are $18^3 = 2592$ deployment plan of RUBiS, where 18 is the component number and 3 is the node number. We make an initial deployment randomly, as shown in Table 3. We assume the number of concurrent clients is 500.

Table 3. An initial deployment plan

Node	Deploys
Aster0	WAR(241kB), 4 JARs(262kB)
Aster2	MySQL DB, 7 JARs(569kB)
Aster3	6 JARs(369kB)

4.1.2 Identification of Pattern Candidates

1. Deployment goal recognition. The deployment goal is to achieve a good performance in average client session time, as mentioned in section 2.We find out 5 patterns have the goal. They are in Table 4.

Table 4. Pattern Candidates

ID	Key elements in context	Main idea of Solution
1	Client Number<1000	Collocating WAR and JARs
2	Client Number<1000	Centralized deployment of EJB
3	Client Number>1000	Separate WAR and JARs
4	Client Number>1000	Distributed deployment of EJB
5		Separate Resource consuming servers

2. Initial context verification. In the experiment environment, the logic expressions in the contexts of Pattern 1 and 2 are true. Pattern 1 and 2 are useful.

4.1.3 Cost Analysis Aided Pattern Execution

1. Selecting an implementation for each pattern candidate.

We use the cost analysis framework to estimate the every possible implementation of the candidate pattern1 and 2. We use the following formulae mentioned in section 3 to make the estimation:

$$CostOfApplication = \max\{costOnEachNode\}$$
$$installationTime = \left\lceil \frac{ejbpackageSize}{100} \right\rceil + 0.1 * \frac{WARsize}{\sum JARsize}$$
$$undeployTimeJAR = 0.2 * \ JARnumber$$

Table 5. Costs of the implementations of candidate

Pattern	Implementation	Cost on Aster0		Cost on Aster2		Cost on Aster3		Maximum Cost	Time by experiment (ms)
		Deploy cost	Undeploy cost	Deploy cost	Undeploy cost	Deploy cost	Undeploy cost		
1	Put all on Aster0	12+0.02	0.2*4	0	0.2*7	0	0.2*6	**12.82**	**14496**
	Put all on Aster2	0	0.2*4	12+0.02	0.2*7	0	0.2*6	13.42	15695
	Put all on Aster3	0	0.2*4	0	0.2*7	12+0.02	0.2*6	13.22	15476
2	Put all EJBs on Aster0	12+0.02	0.2*4	0	0.2*7	0	0.2*6	**12.82**	**14496**
	Put all EJBs on Aster2	0	0.2*4	12	0.2*7	0	0.2*6	13.4	14663
	Put all EJBs on Aster3	0	0.2*4	0	0.2*7	12	0.2*6	13.2	15278

The estimation processes and results are in Table 5. Taking pattern 1 for example, it has 3 implementations. They are to put all the components on Aster0, Aster2 and Aster3. By cost analysis we know the deployment cost on Aster0 is the least, therefore we take it as the implementation of Pattern 1. By the same way we take deployment on Aster0 again as the implementation of Pattern 2. The last column in

5 is the experiment result for the 6 implementations. We can see our cost analysis framework do find the implementation with least deployment cost for each pattern.

2. Selecting a pattern to execute.

From Table 5 we know the costs of the two implementations are the same, so we are free to choose any one of them. As a matter of fact, they happen to be the same deployment, which is shown in Table 6.

Table 6. Implementation of the two Patterns

Node	Deploys
Aster0	WAR, 17 JARs
Aster2	MySQL DB
Aster3	none

3. Executing the redeployment.

We carry out the redeployment by CADTool and monitor the system running to know that the average session time of the redeployed system is reduced by 627 ms.

4.2 More Experiments on Cost of Deployment

We apply more patterns and their different implementations to make redeployments.

Table 7. Experiment results of redeployments

No	initial deployment ID	Applied pattern ID	Redeployment ID	Cost	Time by experiment (ms)
1	1	1	2	13.42	15695
2	1	2	3	13.4	14640
3	3	1	2	15.42	15297
4	2	5	4	7.4	7693
5	2	5	5	8.42	8532

Table 7 depicts the results of our experiments. The last two columns show the estimated deployment cost is approximately in proportion to the real deployment time. The relationships of the deployment cost can be reflected by that of the estimated costs.

5 .Conclusions

This paper proposes a pattern based approach of J2EE application deployment and make it practical by a quantitative method to evaluate deployment cost. It gives experiments to validate them.

The idea of deployment pattern is not novel, but there is little attention paid in using of them, especially from the deployment cost point of view. To our best knowledge, there is no other work that does the same thing as we do, i.e., giving a deployment cost analysis framework of J2EE application by time.

The current weaknesses of the approach are: lacks of tool support and more analysis experiments on other kinds of application servers. The future work can be on studying more implementation of J2EE deployment process to find more common rules about the deployment cost. In our study of PKUAS deployment, an enhancement of the cost analysis precision is also needed.

References

[1] Gamma et al. Design patterns: elements of reusable object-oriented software, Addison Wesley Longman, 1995.
[2] Ling LAN, Gang HUANG et al. Architecture based Deployment of Large-Scale Component based Systems: the Tool and Principles, CBSE, Lisbon, Portugal, 2004..
[3] Mei, H. and G. Huang. PKUAS: An Architecture-based Reflective Component Operating Platform, invited paper, 10th IEEE International Workshop on Future Trends of Distributed Computing Systems, Suzhou, China, 2004.
[4] Marija Mikic-Rakic, Sam Malek, and Nenad Medvidovic. Improving Availability in Large, Distributed Component-Based Systems Via Redeployment, Component Deployment, Grenoble, France, 2005.
[5] T. Kichkaylo et al. Constrained Component Deployment in Wide-Area Networks Using AI Planning Techniques, International Parallel and Distributed Processing Symposium, Nice, France, 2003.
[6] SUN Microsystems, Java 2 Platform Enterprise Edition Specification, Version 5.0, SUN Microsystems, 2005.
[7] SUN Microsystems, Java 2 Enterprise Edition Deployment API Specification, Version 1.1, SUN Microsystems, 2002.
[8]http://msdn2.microsoft.com/en-us/library/ms998478.aspx
[9] http://rubis.objectweb.org

Exploratory Design of Derivation Business Rules Using Query Rewriting

Roman Krenický, David Willmor and Suzanne M. Embury
School of Computer Science, University of Manchester,
Oxford Road, Manchester, M13 9PL, United Kingdom
sembury@cs.manchester.ac.uk

Abstract

In the context of the WiA$_{\rm BR}$*project, we have been exploring a new form of hypothetical reasoning which allows users to explore the consequences of making changes to their business rules before they expend cost and effort in applying them to their organisation. We focus in this paper on techniques to support querying over hypothetical collections of one kind of business rule, namely, derivation business rules. Rules of this kind would ordinarily be managed through a materialised view mechanism. However, in our context, where updates to both rules and data are expected to occur as frequently as queries, this approach is not suitable. We explore an alternative approach based on query rewriting, and show how the semantics of data access in the context of hypothetical changes to derivation business rules are more complex than might first be thought.*

1. Introduction

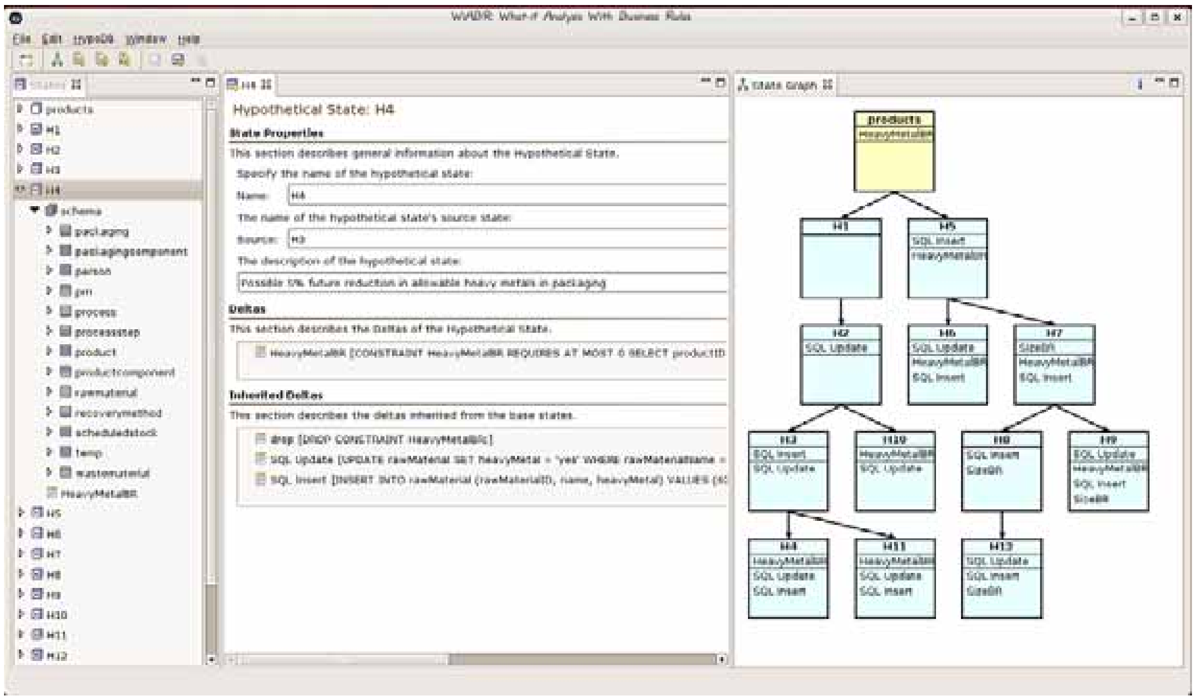
Designing new sets of business rules is a challenging task for modern organisations. In addition to being correct and coherent within themselves, any new business rules must also fit well with the organisation's existing rules. Even a small business typically has hundreds, if not thousands, of business rules that describe their policies and practices, as well as statutory behaviours imposed by governments and other public bodies. Assessing the effects of modifications to even small subsets of the overall body of rules can be challenging, and obscure anomalous interactions can easily be overlooked.

In the WiA$_{\rm BR}$ project[1], we set out to investigate supporting technologies that would assist business managers in assessing the impact of changes to business rules, before money and effort has been put into implementing them (at which point they become even more difficult to change). We have been constructing a tool that sits on top of the organisation's existing databases, and allows the user to explore the effects of making changes to rules without risking any damage to these important operational data sets. Through WiA$_{\rm BR}$, the user makes hypothetical changes to both data and business rules, in order to construct a variety of hypothetical states, each representing a different possible way forward for the organisation. The states can be queried, in order to explore their properties, and comparisons can be made between them. This is illustrated by the screen shot in Figure 1, which shows the main query interface to WiA$_{\rm BR}$, as well as an overview of the hypothetical states the user has created so far, and the relationships between them.

Hypothetical database access is not a new idea. The notion was proposed by Stonebraker *et al.* [1] in the early 80s, and has since been explored and developed by a small number of researchers. Notable among these are: Kulkarni *et al.*, who generalised Stonebraker's original concept into the *independently updated view* (IUV) and proposed both eager and lazy approaches to the materialisation of the hypothetical views [2]; the members of the Heraclitus project team, who promoted data deltas to first class citizens to support more flexible forms of hypothetical querying [3, 4]; and the members of the SESAME project team, who employed the hypothetical queries of Heraclitus in an OLAP context to support decision making [5].

In our context, we need support for hypothetical additions and deletions of business rules, as well as standard hypothetical querying in the presence of changes to rules and data [6]. Existing hypothetical database mechanisms did not support this. In this paper, we describe one part of our exploration of the techniques that are best suited to implementing hypothetical changes to business rules. Business rules come in a variety of different forms, each of which presents different implementation challenges for hypothetical reasoning. Here, we focus on one type of rule, derivation business rules (DBRs). We show how existing approaches to implementing DBRs are not appropriate for our hypothetical context (Section 2), and present instead an alternative approach based on query rewriting (Section 3). The

[1]WiA$_{\rm BR}$ stands for "'What-If Analysis based on Business Rules". It's development was supported by a grant from the UK Engineering and Physical Sciences Research Council.



Hypothetical State: H4

State Properties

This section describes general information about the Hypothetical State.

Specify the name of the hypothetical state:

Name: H4

The name of the hypothetical state's source state:

Source: H3

The description of the hypothetical state:

Possible 5% future reduction in allowable heavy metals in packaging

Deltas

This section describes the Deltas of the Hypothetical State.

HeavyMetalBR [CONSTRAINT HeavyMetalBR REQUIRES AT MOST 0 SELECT productID

Inherited Deltas

This section describes the deltas inherited from the base states.

drop [DROP CONSTRAINT HeavyMetalBR]

SQL Update [UPDATE rawMaterial SET heavyMetal = 'yes' WHERE rawMaterialName =

SQL Insert [INSERT INTO rawMaterial (rawMaterialID, name, heavyMetal) VALUES (6

Figure 1. Main Screen of WiA$_{\mathrm{BR}}$ Showing Hypothetical States

algorithm for applying the rewriting rules in the presence of multiple DBR changes is presented (Section 4). Finally, we draw conclusions and outline the options for future work (Section 5).

2. Hypothetical Changes to DBRs

Business rules are commonly classified into: constraint-based rules, which describe valid or legal states of an organisation; event-based rules, which describe the actions that must be taken in certain circumstances; and derivation business rules, our focus in this paper, which describe how to compute or infer some quantity or value from other organisational data [7]. Classic examples of DBRs are rules that describes how to compute the sales tax due on orders, and the discount a customer may receive on a specific product.

We consider a DBR to define a new attribute on a relation, based on a query over existing attributes, and use the following simple extension to SQL to express them:

```
<defBR> ::= "DEFINE" <rel> "." <attr>
            "AS" <unary-query>
```

Here, `<rel>` denotes the name of a relation, `<attr>` an (unqualified) attribute name and `<unary-query>` a SELECT expression which has exactly one attribute (the derived attribute) in its select-list. This command requests the addition of a new attribute to the given relation. The attribute cannot be updated directly by the user, but receives its value from the embedded query (i.e. `<unary-query>`). Thus, the query must specify at most one result for each tuple in `<rel>`; where no result is specified, the derived attribute is assumed to be NULL. DBR queries may make use of other derived attributes, provided that there are no cycles in the dependencies.

To allow us to link each derived value to the tuple to which it belongs, we require each tuple in `<rel>` to be accessed in the DBR query. This allows us to use tuple identifiers (as supported by all the major DBMSs — for example, the *rowid* column in Oracle, *OID* in PostgreSQL) to link derived values to their associated tuple. Thus, the `<unary-query>` that defines the DBR attribute is transformed by the query processor so that it returns two values: the derived attribute value and the (stored) `<rel>.rowid` attribute. This relation can then be equi-joined with `<rel>` to produce the original relation extended with the new derived attribute[2].

It is possible for the DBR relation to appear more than once in the `<unary-query>` (with aliases), provided that it appears in the FROM clause of the query without an alias exactly once. The query processor will treat this occurrence

[2]The need to preserve tuple identifies in the DBR query means that we cannot use operators that do not have a deterministic semantics for tuple identifiers (such as duplicate elimination or grouping operators) in producing the query result.

of the relation as the significant one for attribute derivation, and will make use of its tuple identifier in joining the derived values to the main relation.

DBRs can be removed with the command:

```
<dropDBR> ::= "DROP" "DERIVATIONBR"
              <rel> "." <attr>
```

which has the effect of deleting `<attr>` from the schema of `<rel>`.

In WiA$_{\mathrm{BR}}$, these commands are executed against individual states. Two kinds of state are distinguished: *historical states*, which correspond to the contents of one of the organisation's existing databases, and *hypothetical states*, which describe hypothetical scenarios based on making changes to other states. A hypothetical state is therefore defined in terms of two properties, the *origin state* from which it is created and the *delta* which describes the set of hypothetical changes that were made to the origin state in order to produce this hypothetical state. As the state overview panel in Figure 1 shows, hypothetical states (such as state `H1`) can be created from historical states (such as `product`) or from other hypothetical states (such as state `H3`, which is an extension of state `H2`). A delta is a hypothetical change (or a sequence of such changes) to either data or business rules. So, each hypothetical state represents a possible scenario for change in the organisation.

Users can explore the network of hypothetical states they have created, to see the effects of the hypothetical changes. For example, a user might create a hypothetical state based on changes to the business rules controlling the discounts offered on profits. Several further hypothetical states could be created from this, one of which assumes the new discounts increase sales by 10%, another assuming an increase of 20%, a third assuming a decrease of 10%, and so on. A query can be issued to each of these states to calculate the profits expected in each scenario, and thus to highlight risks and potential benefits of changing the business rules in this manner in the real organisation.

To support this, we need a mechanism for query answering against hypothetical states that takes into account the changes to data and business rules that define the state. DBRs are commonly implemented using materialised views [8], in which the current values for the derived attributes are stored in a relation, ready for access on querying. Whenever an update occurs, the materialised view is updated to bring it back into consistency with the updated data.

Materialised views have been studied extensively, and have been shown to be an excellent and efficient solution when query rates are high and update rates are low. Because the data is stored, there is no overhead in accessing derived attributes at query time. But, when a new view is added, then there is a significant overhead while the materialised tables are initially populated, and a further (though hopefully reduced) cost when changes occur to the base data and the effects must be duplicated on the materialised view. In ordinary database applications, this is a good trade off, since we expect queries to be occurring frequently, updates slightly less frequently and the creation of new views very rarely. Unfortunately, in the context of exploratory rule design, the situation is reversed. The user typically needs to create many hypothetical states, and the view will need to be rematerialised from scratch for each one that modifies the business rule in question or data stored in its relation. We also expect there to be roughly equal numbers of updates and queries for each state, but that no single state will need to answer a great many queries (certainly not as many as a typical OLTP query).

For all these reasons, we need a different strategy for handling DBRs in hypothetical states. Since queries occur relatively infrequently in our context, this suggests that it may be better to impose the major DBR cost at query time, by folding the DBR definition into the query and computing the attribute values only for such tuples as are directly involved in the query. In the following section, we discuss the ramifications of this approach for exploratory business rule design in hypothetical databases.

3. DBR Deltas through Query Rewriting

Given a DBR with query uq that adds an attribute a to a relation r, whenever a query q is posed against a state in which the DBR is active, we must rewrite q so that all references to $r.a$ are replaced with an equivalent expression which describes how to compute the attribute value. The necessary rewritings are summarised in Figure 2. For consistency within this paper, we will also adopt this rewriting approach to handle data deltas, so there are four cases to consider, though in our implementation we make use of stored hypothetical relations for this purpose [2].

More precisely, we apply the following rewrite rules, to translate a query q applied to a hypothetical state s. Our implementation is based on Oracle, and the peculiarities of SQL for this DBMS have an occasional impact on the detailed form of the rules. We also assume the existence of a relation called `singleton`, which contains exactly one tuple.

Data Insertion Deltas: for a delta `INSERT INTO <R> (<cols>) VALUES (<ivals>)` replace each `<R>` in q with:

```
(SELECT <cols> FROM <R>)
UNION ALL
(SELECT <ivals> FROM singleton)
```

Here, we use the singleton relation to produce a single-tuple relation with the values to insert into `<R>`. Any

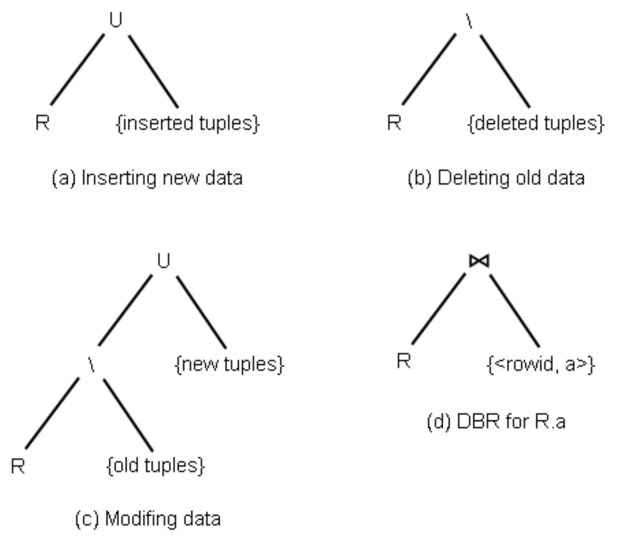


Figure 2. Overview of Delta Rewritings

attributes of `<R>` not mentioned in `<cols>` are also selected in the first SELECT statement and NULL-values are inserted for them by adding “NULL” to the selection list of the second SELECT statement

For a delta of the form `INSERT INTO <R> (<cols>) <iquery>` replace each `<R>` with:

```
(SELECT <cols> FROM <R>)
   UNION ALL <iquery>
```

with the same treatment of unselected attributes.

Data Deletion Deltas: for a delta of the form `DELETE FROM <R>`, replace each `<R>` in *q* with:

```
SELECT * FROM <R> WHERE 1=0
```

to create an empty relation with the correct schema[3]. An alternative is to rewrite the query so that it no longer contains `<R>`, but this is a much more complex rewriting and unconditional deletions are comparatively rare in practice.

For a delta of the form `DELETE FROM <R> WHERE <dcond>`, all occurrences of `<R>` should become:

```
SELECT * FROM <R> WHERE rowid
  NOT IN (SELECT rowid FROM <R>
          WHERE <dcond>)
```

[3]The SQL standard allows the explicitly unsatisfiable selection condition “WHERE FALSE”, but this is not supported by Oracle.

Various other rewritings look superficially plausible for this case, but they are either not supported by Oracle, behave incorrectly in the case of NULL values or else introduce undesirable duplicate elimination effects. The rewrite given here has the advantage that tuple identities (so important for later application of DBRs) are preserved.

Data Modification Deltas: for a delta of the form `UPDATE <R> SET <ucol1>=<uexpr1>, ...` we replace each `<R>` in *q* with:

```
SELECT <uexpr1> AS <ucol1>, ...,
       <Rcols - ucols> FROM <R>
```

where `<Rcols - ucols>` are the attributes of `<R>` which are not updated by the delta.

For a delta of the form `UPDATE <R> SET <ucol1>=<uexpr1>,...WHERE <ucond>`, we need to select the updated versions of the tuples that satisfy the given condition and union them with the (unchanged) tuples which don’t. In Oracle SQL, this can be achieved by the expression:

```
(SELECT <uexpr1> AS <ucol1>, ...,
        <Rcols - ucols> FROM <R>
 WHERE <ucond>)
UNION ALL
(SELECT <ucol1>, ...,
        <Rcols - ucols> FROM <R>
 WHERE rowid NOT IN
       (SELECT rowid FROM <R>
        WHERE <ucond>))
```

New DBR Deltas: as might be expected, the rewriting rules for DBR deltas is more complicated than those for data deltas. DBRs take the form:

```
DEFINE <R>.<a> AS
SELECT <expr> <from-where-etc>
```

where `<from-where-etc>` is the remainder of the defining query for the new attribute, i.e., the FROM clause and the optional WHERE and ORDER BY clauses). In the context of such a delta, we replace every occurrence of `<R>` in *q* with:

```
<R>
LEFT OUTER JOIN
(SELECT <expr> AS <a>, <R>.rowid
 <from-where-etc>) Tmp
ON (<R>.rowid=Tmp.rowid)
```

This rewriting takes advantage of the assumption mentioned earlier that the DBR query will always include a single non-aliased occurrence of `<R>`. As discussed, the `rowid` attribute is added to the SELECT clause of the DBR query, and the resulting binary relation is joined with `<R>`. We use a left outer join so that tuples for which a derived attribute value is not computed by the DBR query remain in the result with `<R>.<a>` equal to NULL.

Delete DBR Deltas: if we have a delta which deletes a derived attribute `<R>.<a>` from a state, we must replace each `<R>` in q with a projection on all of the attributes of `<R>` excluding `<a>`, i.e.

```
SELECT <Rcols - a> FROM <R>
```

A further complicating factor which applies to all the rewriting rules mentioned here is that any replacement expression for a relation must preserve its original name. This is so that hypothetical queries can be expressed in terms of the basic schema, rather than having to be rewritten to use different relation names for each hypothetical state, which would be unintuitive and awkward for the user. We can achieve this using aliasing, but only at the expense of wrapping the rewritten expression in a further SELECT clause. For example, the replacement expression for `<R>` in the presence of a new DBR delta would be enclosed in the following expression:

```
SELECT <R>.*, Tmp.<a> FROM <ReplExpr>
```

There are also certain cases in which special handling of tuple identifiers is required, which space does not allow us to discuss. The full set of rewrite rules are given elsewhere [9].

4. Application of the Rewrite Rules

The previous section described the rewrite rules to be applied for application of a single delta to a query being evaluated against a hypothetical state. In general, of course, we will have a sequence of deltas describing the difference between this hypothetical state and its origin state. The effects of all of these must be absorbed into the query in the correct order before it can be evaluated against the root (historical) state, as a normal query. Since each delta must be applied in the context of those which precede it, we work through the sequence of deltas in reverse order, applying the rewrite rules as we go. However, the presence of DBR deltas complicates this process significantly, since the application of each delta may also imply a change to one or more of the derived attributes currently in scope. Thus, in between each delta application, we must reapply the deltas for each of the derivation business rules currently in scope.

This point is best explained with an example. Suppose we have a sequence of deltas $<\delta_1, \delta_2>$ which together describe a hypothetical state s_1 derived from another state s_2. Suppose further that the state s_1 "inherits" two DBRs (r_1 and r_2) from s_2 and that δ_1 is a data delta while δ_2 adds a new DBR (r_3) to the state. If we were building up s_1 as a true database state, rather than a hypothetical one, we would need to apply the deltas in the following order:

$$\delta_1 \; ; \; applyDBRs \; ; \; \delta_2 \; ; \; applyDBRs$$

Therefore, if we wish to convert a hypothetical query q expressed against state s_1 into a query that can be executed against the base state s_2, we must apply the rewrite rules for the "inverse" of this set of changes to q. Furthermore, since the DBRs are themselves queries, they are also subject to the effects of subsequent DBR deltas, which may change the schema elements referenced in each. Therefore, they must also be rewritten, before being applied to the hypothetical query, but in a forward order rather than the reverse order described here.

These considerations yield the following algorithm for query rewriting in the presence of a sequence of data and DBR deltas:

1. Apply all (derivation) business rule deltas in forward order to the inherited set of business rules.
2. Apply all derivation business rules to the query (i.e., replace each occurrence of each rule's relation by the appropriate rewriting expression)
3. Apply the last delta of the sequence to the rewritten query, as follows:
    - if it is a DBR delta, apply it in reverse to the set of business rules, or
    - if it is a data delta, apply it to the query by replacing each occurrence of its relation by the appropriate rewriting expression.
4. Apply the current set of derivation business rules to the rewritten query.
5. Repeat from step 3 with the next delta in the (reverse) sequence.
6. When all deltas are processed, apply the initial (inherited) set of DBRs to the query that result.

This algorithm is considerably more complex, and involves far more rewriting steps, than might have been predicted for the rewriting approach. Since some of the rewritings introduce several copies of the affected relation, the repeated application of the rewriting rules can very quickly explode, resulting in a highly nested query with many occurrences of the relations involved in DBRs. Some of these will be handled elegantly by the DBMS's query optimiser, which may be able to spot the repeated expressions and cache them, to avoid continually re-evaluating them. In our trials with even relatively simple examples, however, we have found that the complexity of the rewritten queries quickly becomes too much for the Oracle query processor to handle elegantly. It is therefore vital that we do not apply rewritings that are unnecessary. One optimisation step that we have implemented in our hypothetical query engine is to check that the derived attributes involved in the deltas are actually used within the query before applying them. This is done by pushing a projection on all attributes down through the relational algebra tree for the query, collecting restrictions on the attributes of interest as we go. Any attribute not restricted by this process is unused by the query, and DBRs relating to it can be ignored for the purposes of query rewriting.

5. Conclusions

We have described how we can support hypothetical reasoning over collections of derivation business rules, using a query rewriting approach. This allows the user to experiment with different combinations of new and existing rules, and to apply them to hypothetical scenarios based on the historical data stored in the organisation's databases. The ability to change one's business rules rapidly and reliably is important for any agile organisation, wishing to keep pace with and advance over its competitors. $\mathrm{WiA}_{\mathrm{BR}}$ provides a sandbox environment in which potential new business rules can be explored and examined, before an organisation embarks on the often risky and expensive process of rolling out the changed business rules to staff and information systems.

We chose to explore query rewriting approaches to hypothetical reasoning due to the unsuitability of materialised solutions for this context. And, in simple cases, the rewriting approach does provide the benefits we expected of it. However, when many hypothetical states are chained one on top of another, or when delta sequences are long, the explosion in the complexity of the rewritten query mean that performance becomes unacceptable for an exploratory style application. One possible solution to this would be to explore a combination of materialised and query rewriting approaches, based on the stability of the individual hypothetical states and the patterns of querying experienced.

A further aspect that is still to be explored in our work is the issue of explanations for the results of hypothetical queries. For example, if a query predicting the profit to be obtained in a given hypothetical scenario gives a much lower result than expected, the user might reasonably ask the system to provide some help in explaining the root cause of this surprising data. This is a form of data lineage problem [10], and further work is required to understand how it would operate in the $\mathrm{WiA}_{\mathrm{BR}}$ context.

References

[1] Stonebraker, M.: Hypothetical Data Bases as Views. In: Proceedings of the ACM-SIGMOD Conference on Management of Data, Ann Arbor, Mich, US (1981)

[2] Kulkarni, U.R., Ramirez, R.G.: Independently Updated Views. IEEE Transactions on Knowledge and Data Engineering **9** (1997) 798–812

[3] Ghandeharizadeh, S., Hull, R., Jacobs, D.: Heraclitus: Elevating Deltas to be First-Class Citizens in a Database Programming Language. ACM Transactions on Database Systems **21** (1996) 370–426

[4] Griffin, T., Hull, R.: A Framework for Implementing Hypothetical Queries. SIGMOD Record **26** (1997) 231–242

[5] Balmin, A., Papadimitriou, T., Papakonstantinou, Y.: Hypothetical Queries in an OLAP Environment. In El Abbadi, A., Brodie, M., Chakravarthy, S., Dayal, U., Kamel, N., Schlageter, G., Whang, K.Y., eds.: Proceedings of 26th International Conference on Very Large Data Bases, Cairo, Egypt, Morgan Kaufmann (2000) 220–231

[6] Embury, S., Willmor, D., Dang, L.: Assessing Impacts of Changes to Business Rules through Data Exploration. In: Proceedings of the International Conference on Software Engineering Advances, IEEE Computer Society Press (2006)

[7] Shao, J., Pound, C.: Reverse Engineering Business Rules from Legacy Systems. BT Technology Journal **17** (1999) 179–186

[8] Chaudhuri, S., Krishnamurthy, R., Potamianos, S., Shim, K.: Optimizing queries with materialized views. In Yu, P., Chen, A., eds.: Proceedings of the Eleventh International Conference on Data Engineering, Taipei, Taiwan, IEEE Computer Society (1995) 190–200

[9] Krenický, R.: A Hypothetical Querying Approach for What-If Analysis Based on Business Rules. Student Research Project Report, University of Manchester/Universität Karlsruhe (2006)

[10] Cui, Y., Widom, J.: Lineage Tracing for General Data Warehouse Transformations. VLDB Journal **12** (2003) 41–58

Classification of Design Pattern Traits

Jing Dong, Yajing Zhao
Department of Computer Science
University of Texas at Dallas, Richardson, TX 75083, USA
{jdong, yxz045100}@utdallas.edu

Abstract

Design patterns describe good solutions to common and recurring problems. The applications of design patterns may vary in different layouts, which pose challenges for recovering and changing these design pattern instances since essential characteristics of each design pattern are described implicitly. In this paper, we categorize different characteristics of each design pattern as its traits in form of predicates. We classify different predicates into groups and levels. In this way, the significant characteristics of each design pattern are explicitly specified in predicates that can be used for design pattern recovery and evolution analysis.

1. Introduction

Design patterns [9] which describe good solutions to recurring problems have been widely accepted and applied in industry. The descriptions of each design pattern typically are high-level design guidelines, such as intent, structure, behavior, participants, and collaborations. Each design pattern describes flexible designs that may be applied in various ways. This poses critical challenges on design pattern recovery and evolution.

Existing approaches [1][2][6][12] for design pattern recovery generally match essential characteristics of each design pattern with the system design. However, different approaches render different result when matching the same design pattern with the same software system. We found several possible reasons. First, each design pattern typically includes a group of classes, each of which plays some role. Some approaches choose partial (rather than full) matches of the roles. Second, the relationships, such as association, generalization, and delegation, between classes are normally important parts of a design pattern, which have been missed by some approaches. Third, there are several ways to implement the delegation relationship in object-oriented programming languages. Some approaches may not consider all these variations. Fourth, existing OO programming languages provide library classes, such as LinkedList, ArrayList, HashMap, and Hashtable, which facilitate the implementation of some design patterns including aggregation relationship, such as the Composite pattern. On the other hand, it complicates the pattern recovery processes [13]. All these possible reasons for result discrepancies from different approaches can be boiled down to one critical issue, which is the lack of explicit specification of essential characteristics of each design pattern.

Design patterns encapsulate future evolutions and changes which will not affect other part of the design. When multiple design patterns are applied and composed, nevertheless, the interactions among them may cause design patterns loose essential characteristics so as to incur an error. Thus, it is important to check whether all essential characteristics of a design pattern still hold when an instance evolves. Such checking also requires explicit specification of the essential characteristics of each design pattern.

In this paper, we categorize the characteristics of each design pattern as its traits, which are specified as predicates. Each predicate is also classified into different groups and levels. In this way, we can use a list of predicates to specify the essential characteristics of a design pattern.

The remainder of this paper is organized as follows. The next section presents a classification of different kinds of predicates to specify essential characteristics of each design pattern. Section 3 discusses the applications of our specification of design pattern traits in pattern discovery and evolution. The last two sections are related work and conclusions.

2. Design Pattern Traits

In this section, we present predicates to specify essential characteristics of each design pattern. In particular, we define two groups of predicates, entity and relationship. Each entity predicate is further categorized as "HAS" or "IS" predicates. Each predicate category includes three levels (class, element, and implementation) of predicates and negation predicates. In each level, we define root and derived traits. Negation predicates are also necessary, as explained in Section 2.3.4.

2.1 Entity Predicates (HAS)

Predicates in this section define whether a design pattern has a specific class, whether a class has a specific operation, whether an operation has a particular parameter, etc.

2.1.1 Class Level

Predicates in this level define whether a piece of design has a class playing a particular role. For example, Figure 1 shows a class diagram of the Composite pattern, which has Component, an abstract class, Composite and Leaf, two concrete classes. To specify these pattern characteristics, we define the following root and derived class level HAS predicates, and their usage is explained by Example 2.1.

Trait 2.1 (Root class level HAS predicate)

- hasClass (D, C): Design D includes Class C.

Trait 2.2 (Derived class level HAS predicates)

- hasAbstractClass (D, C) = hasClass (D, C) Λ isAbstract (C): Design D includes abstract class C.
- hasConcreteClass (D, C) = hasClass (D, C) Λ isConcrete (C): Design D includes concrete class C.

Example 2.1 (Composite pattern HAS traits)

hasAbstractClass (CompositePattern, Component) Λ
hasConcreteClass (CompositePattern, Composite) Λ
hasConcreteClass (CompositePattern, Leaf)

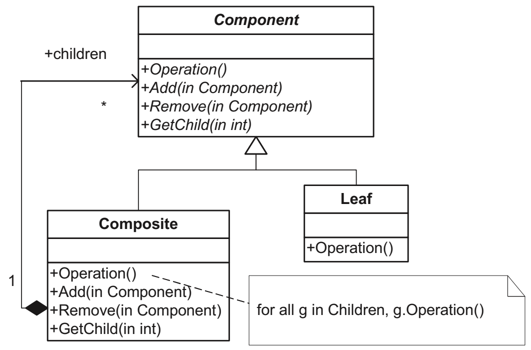


Figure 1 Class Diagram of Composite Pattern

2.1.2 Element Level

Predicates in this level define the required attributes and operations in a class. Their usage is illustrated in Example 2.2.

Trait 2.3 (Root element level HAS predicates)

- hasOperation (C, O): Class C has operation O.
- hasAttribute (C, A): Class C has attribute A.

Trait 2.4 (Derived element level HAS predicates)

- hasAbstractOperation (C, O) = hasOperation (C, O) Λ isAbstract (O): Class C has abstract operation O.
- hasConcreteOperation (C, O) = hasOperation (C, O) Λ isConcrete (O): Class C has concrete operation O.
- hasPublicOperation (C, O) = hasOperation (C, O) Λ isPublic (O): Class C has public operation O.
- hasPrivateOperation (C, O) = hasOperation (C, O) Λ isPrivate (O): Class C has private operation O.
- hasProtectedOperation (C, O) = hasOperation (C, O) Λ isProtected (O): Class C has protected operation O.
- hasPublicAttribute (C, A) = hasAttribute (C, A) Λ isPublic (A): Class C has public attribute A.
- hasPrivateAttribute (C, A) = hasAttribute (C, A) Λ isPrivate (A): Class C has private attribute A.
- hasProtectedAttribute (C, A) = hasAttribute (C, A) Λ isProtected (A): Class C has protected attribute A.
- operationSet (OS, C) = $\forall Opr_i \in OS$, hasOperation (C, Opr_i): each operation Opr_i in OS is a method of Class C.

Example 2.2 (Composite pattern HAS traits)

hasAbstractOperation (Component, Operation) Λ
hasAbstractOperation (Component, Add) Λ
hasAbstractOperation (Component, Delete) Λ
hasAbstractOperation (Component, GetChild) Λ
hasConcreteOperation (Composite, Operation) Λ
hasConcreteOperation (Composite, Add) Λ
hasConcreteOperation (Composite, Delete) Λ
hasConcreteOperation (Composite, GetChild) Λ
hasConcreteOperation (Leaf, Operation)

2.1.3 Implementation Level

Predicates in this level define the behavior of operations. For example, the Add() and Delete() operations shall have one parameter of type Component. To specify these characteristics, we define the following predicates:

Trait 2.5 (Root implementation level HAS predicates)

- hasStmt (O, S): Operation O has a statement S.
- hasReturnValue (O, Ob): Operation O has Ob as return value.
- hasReturnType (O, T): Operation O has T as return type.
- hasParameter (O, T, k): T is the type of the k^{th} parameter of Operation O. If k=1, then k can be omitted.

Example 2.3 (Composite pattern HAS traits)

hasParameter (Add, Component) Λ
hasParameter (Delete, Component) Λ
hasParameter (GetChild, int)

2.2 Entity Traits (IS)

Predicates in this section define the constraints for classes, attributes, and operations.

2.2.1 Class Level

Predicates in this level define the types of classes, for example, abstract or concrete.

Trait 2.6 (Root class level IS predicates)

- isAbstract (C): Class C is an abstract class.
- isConcrete (C): Class C is a concrete class.

2.2.2 Element Level

Predicates in this level define the characters of attributes and operations, for example, whether an operation is abstract or not.

Trait 2.7 (Root element level IS predicates)

- isAbstract (O): Operation O is an abstract operation.
- isConcrete (O): Operation O is a concrete operation.
- isPublic (O): Operation O or attribute O is public.
- isPrivate (O): Operation O or attribute O is private.
- isProtected (O): Operation O or attribute O is protected.
- equal (A, B): A and B is equal to each other. For example, equal (name (O1), name (O2)).
- isType (Ob, T): Object Ob is of type T.

2.3 Relation Traits

Predicates in this section define the relation between classes, attributes, and operations.

2.3.1 Class Level

Predicates in this level define the constraints for the relations between classes.

Trait 2.8 (Root class level relation predicates)

- generalize (C1, C2): Class C1 is a subclass of class C2.
- associate (C1, C2): Class C1 keeps a reference to class C2.
- aggregate (C1, C2): Class C1 maintains a reference to C2, and class C2 is a part of C1 semantically.
- oneToMore (C1, C2): The multiplicity of association or aggregation relationship from class C1 to class C2 is 1:m.
- oneToOne (C1, C2): The multiplicity of association or aggregation relationship from class C1 to class C2 is 1:1.

- moreToMore (C1, C2): The multiplicity of association or aggregation relationship from class C1 to class C2 is m:m.
- create(C1, C2): Class C1 is responsible for creating Class C2.
- correspondingRelated (P (SET1, SET2)): $\forall$ E1 $\in$ SET1, $\exists$ E2 $\in$ SET2, s.t., P (E1, E2). This predicate is used specially in the Abstract Factory pattern where there shall be at least one create method for each family of products.

Trait 2.9 (Derived class level relation predicates)

- childrenSet (CS, C) = $\forall \text{Class}_i \in CS$, generalize ($\text{Class}_i$, C): each class Class_i in CS is a subclass of C.

Example 2.4 (Composite pattern relation traits)
generalize (Composite, Component) Λ
generalize (Leaf, Component) Λ
aggregate (Composite, Component) Λ
oneToMore (Composite, Component)

The usage of create(C1, C2) and correspondingRelated (P (SET1, SET2)) can be illustrated with the Abstract Factory pattern in the following partial specifications.

Example 2.5 (Partial Abstract Factory pattern relation traits)
correspondingRelated (create (S, $\cup S_i$)) Λ
childrenSet(CS, AbstractFactory) Λ
childrenSet(S_i, AbstractProduct_i)

2.3.2 Element Level

Some of the relationships can be detailed in element level, such as create.

Trait 2.10 (Root element level relation predicates)

- create (C1, M, C2): method M in class C1 creates class C2.

2.3.3 Implementation Level

Predicate in this section describes relations between classes at the implementation level.

Trait 2.11 (Root implementation level relation predicate)

- delegate (C1, O1, C2, O2): Operation O1 in class C1 forwards request to Operation O2 in class C2.

Example 2.6 (Composite pattern relation traits)
delegate (Composite, Operation, Component, Operation)

2.3.4 Negation Relationships

It is mandatory that some relations shall not exist between classes, attributes, or operations. In the Composite pattern, for example, there is a generalization relationship from Composite to Component. Hence, there shall not be a generalization relationship from Component to Composite, to avoid a circular generalization relationship. Negation relationship is used to guarantee this kind of characteristics.

Trait 2.12 (Root negation relation predicates)

- noDirectAccess (C1, C2): Class C1 does not have direct access to class C2, the attributes or the operations of C2

Trait 2.13 (Derived negation relation predicates)

- differentInterface (C1, O1, C2, O2) = ¬ ($\forall$i hasParameter(O1, Ti, i) Λ hasParameter(O2, Ti, i) Λ hasReturnType(O1, Tr) Λ hasReturnType(O2, Tr)): Operation O1 in class C1 has different interface from Operation O2 in class C2.
- noGeneralize (C1, C2) = ¬ (generalize (C1, C2)): There shall not be a generalization relationship from class C1 to C2.
- noAggregate (C1, C2 = ¬ (aggregate (C1, C2))): There shall not be an aggregation relationship from class C1 to C2.

Example 2.7 (Composite pattern relation traits)
noGeneralize (Component, Composite) Λ
noGeneralize (Component, Leaf) Λ
noAggregate (Component, Composite)

The usage of differentInterface (C1, O1, C2, O2) and noDirectAccess (C1, C2) can be illustrated with the Adapter pattern [9].

Example 2.8 (Partial Adapter pattern relation traits)
differentInterface (Adapter, Request, Adaptee, SpecificRequest) Λ noDirectAccess (Client, Adaptee)

3. Applications

The predicates introduced in the previous section have many practical usages. In this section, we discuss two applications, one on design pattern recovery and the other on design pattern evolution.

3.1 Design Pattern Recovery

Design pattern recovery from source code is a popular research topic [1][2][6][12]. There are many tools developed. As discussed previously, however, different tools may generate different results when discovering the same pattern from the same system. The main reason is that each approach defines its own set of pattern characteristics and embeds the knowledge in their tools. In the previous section, we present an approach to explicitly specify the pattern characteristics as traits. There are several benefits of our approach. First, it makes explicit the pattern characteristics to be discovered. Second, different approaches may share the same set of characteristics of each design pattern. Third, it allows different approaches to be compared based on the same standard. Fourth, the pattern characteristics described by our predicates can be the input of pattern recovery tools such that the algorithms for pattern matching can be separated from the definition of patterns. We introduced our design pattern recovery approach in [6]. We will extend our tool to accept design pattern traits defined in this paper. We will transform the predicates defined in the previous section into XMI format so that they can be the input of our tool in the future.

As an example, the traits of the Composite pattern can be described by combining the predicates listed in Example 2.1, 2.2, 2.3, 2.4, 2.6, and 2.7.

3.2 Design Pattern Evolution

Each design pattern typically documents possible future changes that may only affect limited part of the pattern. The evolution process of design patterns can be achieved by adding or removing design elements in existing design patterns. A classification of possible ways of pattern evolution has been presented in [7]. When a design pattern evolves, a group of modeling elements may be added or removed from the original design. However, such information on pattern evolutions is generally implicit in the description of each design pattern. Missing part of this group of modeling elements may result in inconsistencies in the design. Thus, it is important to specify the essential characteristics to be added into or removed from a design pattern when the pattern evolves. Our approach presented in the previous section can be used to explicitly define such essential characteristics of each design pattern evolution. After a design pattern instance evolves, thus, a group of predicates corresponding to the group of modeling elements can be added or removed. For example, an initial instance of the Composite pattern may have one leaf class and one composite class, as shown in Figure 1. The traits of this instance are already described in the previous sections. When the pattern instance evolves by adding another leaf class, Leaf1, new traits are to be included correspondingly. The following predicates are added to the original one to form the new traits of evolved Composite pattern instance.

hasConcreteClass (CompositePattern, Leaf1) Λ
hasConcreteOperation (Leaf1, Operation) Λ
generalize (Leaf1, Component) Λ
noGeneralize (Component, Leaf1)

4. Related Work

Eden *et al.* [8] describe a precise method to specify how a design pattern can be applied into existing code in a meta-programming language. They presented a prototype to support the specification of design patterns and automatic applications of patterns. Balanced Pattern Specification Language (BPSL) [11] considers incorporates First Order Logic (FOL) to describe the structural aspect and Temporal Logic of Actions (TLA) to depict behavioral aspect of design patterns. In contrast to the above two approach, we consider all kinds of traits of patterns that are important in various applications, such as recovery and evolution. Our specification includes low-level implementation information, rather than only high-level design knowledge.

Design Pattern Markup Language (DPML) is defined in [2]. It provides an easy way for users to modify pattern descriptions to suit their needs, or define their own patterns. Contrast to their approach, our approach defines not only normal traits but also negation predicates that ensure the pattern traits by excluding certain characteristics.

Dietrich [3] introduces an approach using the web ontology language (OWL) to document design patterns found. They build uniform resource identifiers (URIs) for pattern artifacts. In contrast, we use simple predicates to define each pattern characteristics. We also classify all the predicates into different categories and levels.

Montero *et al.* [10] propose a semantic ontology-based representation for domain specific patterns based on the domain knowledge for which they were written. Instead of focusing on the domain specific patterns, we deal with general design patterns that help software design and improve software adaptability and extensibility.

Our previous work [4][5] on formalizing design patterns and reasoning about their compositions also focuses on high-level design structure and behavior of design patterns. Different from our previous goals of formal specification and verification of design patterns and their compositions at design level, our approach in this paper aims at pattern recovery and evolution based on a light-weighted formalism.

5. Conclusions

In this paper, we present our approach of explicitly specifying essential characteristics of a design pattern using predicates. We classify predicates into several groups, e.g., entity and relation, and levels, e.g., class, element, and implement. Each level includes root and derived predicates. We plan to incorporate this work into design pattern recovery and evolution. In particular, our tools will separate the pattern specifications from its recovery and evolution so that our tools can accept any pattern descriptions in the format defined in this paper. Besides, the theory can also help automatic application of design patterns.

References

[1] G. Antoniol, R. Fiutem, and L. Cristoforetti, “Design pattern recovery in object-oriented software.” *Proceedings of the 6th IEEE International Workshop on Program Understanding (IWPC)*, pp 153-160, 1998.

[2] Z. Balanyi and R. Ferenc, “Mining design patterns from C++ source code.” *Proceedings of the 19th IEEE International Conference on Software Maintenance (ICSM)*, pp. 305-314, September 2003.

[3] J. Dietrich and C. Elgar, “A formal description of design patterns using OWL.” In Proceedings of the Australian Software Engineering Conference (ASWEC), 2005.

[4] J. Dong, P. Alencar, and D. Cowan, A Behavioral Analysis and Verification Approach to Pattern-Based Design Composition, the International Journal of Software and Systems Modeling, Springer-Verlag, Volume 3, Number 4, December 2004, Pages 262-272.

[5] J. Dong, P. Alencar, and D. Cowan, Automating the Analysis of Design Component Contracts, International Journal of Software - Practice and Experience (SPE), Wiley, Volume 36, Issue 1, pages 27-71, January 2006

[6] J. Dong, D. S. Lad and Y. Zhao, “DP-Miner: Design Pattern Discovery Using Matrix.” The Proceedings of the Fourteenth Annual IEEE International Conference on Engineering of Computer Based Systems (ECBS), Arizona, USA, March 2007.

[7] J. Dong, S. Yang, and K. Zhang, “A model transformation approach for design pattern evolutions.” Proceedings of the Thirteenth Annual IEEE International Conference on Engineering of Computer Based Systems (ECBS), p.p. 80-89, Germany, March 2006.

[8] A. H. Eden, A. Yehudai, and J. Gil, “Precise specification and automatic application of design patterns.” In International Conference on Automated Software Engineering, IEEE Press, pp 143–152, 1997.

[9] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[10] S. Montero, P. Diaz, and I. Aedo, “A semantic representation for domain-specific patterns.” In International Symposium on Metainformatics, U. K. Wiil, Ed., Springer-Verlage, LNCS 3511, pp. 129-140, 2005.

[11] T. Taibi and D. Ngo, “Formal Specification of Design Patterns – A Balanced Approach.” In Journal of Object Technology, 2(4), pp 127-140, July-August, 2003.

[12] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. Halkidis, “Design Pattern Detection Using Similarity Scoring.” *IEEE transaction on software engineering*, Vol. 32, No. 11, November 2006.

[13] R. J. Wirfs-Brock, "Refreshing Patterns," *IEEE Software*, vol.23, no.3, pp. 45-47, May/Jun, 2006.

A Proposal for a Conceptual Data Warehouse Quality Model

Manuel Serrano, Rafael Romero, José-Norberto Mazón, Juan Trujillo, and Mario Piattini

***Abstract*— Data warehouses are considered a cornerstone of a decision support system, since they provide the adequate information resources for decision making in an integrated manner. Therefore, the quality of a data warehouse is a key issue in any business organization, and several works have pointed out the importance of measuring the quality in data warehouse domain. However, to the best of our knowledge, the current approaches that deal with the quality aspects of data warehouses only consider isolated quality metrics, without being part of a comprehensive quality model that allows designers to assess the quality of a data warehouse in a systematic and objective way. Having considered this limitation in current data warehouse research, in this paper we present a quality model for the data warehouse domain. This model allows designers to state the meaning of quality in the data warehouse domain, thus measuring the quality of a data warehouse conceptual model in a systematic and objective way. We have used a well-known methodology to build this quality model, and we have related the elements of this quality model to our previously defined and validated quality metrics for conceptual models of data warehouses.**

***Index Terms*— Data Warehouse, Quality, Quality Model**

I. Introduction

Nowadays, data warehouses have become one of the most crucial components of decision support systems, since they efficiently and effectively allow the integrated management of data in order to provide information resources that support effective problem and opportunity identification, critical decision-making, and strategy formulation, implementation, and evaluation. Therefore, data warehouse technology has become an adequate solution, providing such resources in an environment where increasing competition, sophisticated and informed costumers, unpredictable market fluctuations, and changing regulatory environments are putting much pressure on business organizations [1].

As a result of this importance, companies are increasing their investment in the development of data warehouses. In fact, the investment in this kind of systems is around several millions a year and increases by 20% per year [2]. However, despite the potential of data warehouses and the huge amount of money invested, success is not necessarily guaranteed. As stated by several authors, a data warehouse project is a real risk [3] and more than 60% of data warehouses do not meet the user expectations [4]. Therefore, designers have to deal with quality issues to avoid these pitfalls and guarantee the success of the data warehouse project [5].

Dealing with quality issues is not an easy task, since quality is an abstract and subjective aspect, for which there is no universal definition: it is usually said that there is a quality definition for each person. In this way, it is very complex to measure or assess the quality of a software product in an objective way. Several guides have been proposed to address the complexity of data warehouse quality, but most of the time, guides are not enough, since they can help designers in their work, but they imply rather subjective decisions, and this can lead to "not-so-good" products.

In order to overcome this inherent subjectivity of quality in data warehouse projects, in this work we introduce a quality model that helps designers to assess the quality of a data warehouse in an objective way, thus guaranteeing success in designing a good data warehouse. In this paper, we focus on the quality of conceptual data warehouse models and we present the first steps towards building a data warehouse quality model. We focus on the quality of the conceptual models, as a *good* design may (or may not) lead to a *good* system, but a *bad* design will surely render a *bad* product of low quality. Thus, it is important to focus on the quality of the product from the first steps of its development, i.e. the conceptual model.

The remainder of this paper is structured as follows. The next section shows related work regarding data warehouse quality and models. In Section 3, a method for developing quality models based on ISO 9126 [6] is presented. In Section 4, the method is applied to the building of a data warehouse quality model. The last section presents some conclusions and future work arising from this work.

II. Related Work

From the beginnings of data warehousing, quality has been an important issue, from the perspectives of both data and

Manuscript received March 8, 2007. This work was supported in part by the CALIA Project supported by the University of Castilla – La Mancha and the DIMENSIONS Project (PBC-05-012-1) supported by Consejería de Educación y Ciencia – Junta de Comunidades de Castilla – La Mancha. Jose-Norberto Mazón is funded by the Spanish Ministry of Education and Science under a FPU grant (AP2005-1360)

Manuel Serrano (corresponding author, Phone: +34 926295300) and Mario Piattini are with the Alarcos Research Group, University of Castilla – La Mancha, Ciudad Real, Spain (e-mail: Manuel.Serrano@uclm.es and Mario.Piattini@uclm.es).

Rafael Romero, José-Norberto Mazón and Juan Trujillo) are with the Department of Software and Computing Systems, University of Alicante, Alicante, Spain (e-mail: romero@dlsi.ua.es, jnmazon@dlsi.ua.es and jtrujillo@dlsi.ua.es).

schema. This interest in quality is motivated by the importance of the information quality in the decision support systems.

Several techniques have been used in an attempt to improve the quality of data warehouses, such as software process improvement models. Among these techniques we can find the Capability Maturity Model [7] or SPICE [8], which relies on the belief that *"good processes deliver good software"*, but unfortunately good processes do not guarantee good software. Good processes only increase the probability that good software may be obtained.

In this search for quality, several quality standards have been applied to data warehouse development (such as ISO 9126 [6] and IEEE 1061 [9], usually complemented by GQM techniques [10, 11]) to attempt to measure the quality of the data warehouse. These standards are too general and complex and are usually not concrete enough to be useful in assessing the quality of the system.

With regard to data warehouse quality measurement we can find several proposals for measuring some quality dimensions of data warehouses such as the metrics proposed by [12] and [5]. These proposals are good approaches to data warehouse measurement but they are not complete, since they are not part of a quality model that allows designers to use them in a systematic and objective way.

Lastly, we can find the proposal of a data warehouse quality model, named DWQ [13] which attempted to assure the quality of the data stored inside the data warehouse to improve the data warehousing experience; this project had some quality dimensions and focused on data quality. We think that data warehouse quality should not only be assessed in terms of data quality and that schema quality is as important an issue as data quality. In this paper we propose a quality model for conceptual data warehouse models, based on the ISO 9126 standard.

III. Quality Model Development Method

Developing a quality model is not an easy task. Several aspects should be considered and the whole process should be carried out in an objective and methodological way. For this purpose, Franch and Carvallo [14] proposed a method to build quality models based on the ISO 9126-1 standard [6]. This method is now briefly described.

The proposed methodology comprises of six steps and also considers a preliminary activity (step 0). A graphical view of the methodology is shown in Figure 1. Although the steps seem to be sequential, these steps can be intertwined and repeated if needed.

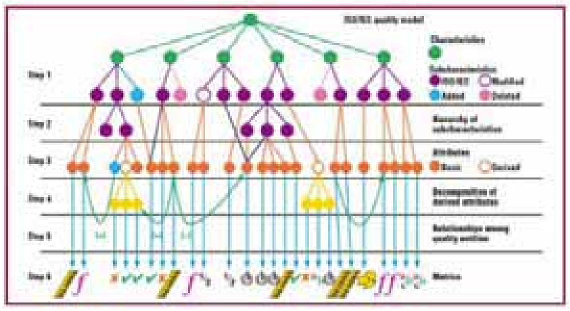


Fig. 1. Franch and Carvallo [14] six steps methodology

Step 0. Defining the domain

Previous to the application of the methodology, it is necessary to carefully examine and describe the domain of interest, with the help of experts. In order to describe the domain the use of conceptual modelling to keep track of relevant concepts is recommended.

Step 1. Determining quality subcharacteristics

The decomposition of characteristics into subcharacteristics that appear in the ISO standard is quite reasonable and should be used unless good reasons for not doing so emerge during domain analysis. In this case, new subcharacteristics are added, some of them are refined and it is even permisable to remove some of the subcharacteristics that do not apply to the domain.

Step 2. Defining a hierarchy of subcharacteristics

Typically, further decomposition of subcharacteristics with respect to some factors is needed. In this way, we obtain a hierarchy of subcharacteristics.

Step 3. Decomposing subcharacteristics into attributes

Quality subcharacteristics provide a comprehensible abstract view of the quality model. But we need to decompose these abstract concepts into more concrete ones - the quality attributes. An attribute keeps track of a particular observable feature of the packages in the domain. Attributes should be precisely defined to clarify the underlying quality concepts that they represent and to link them with the appropriate subcharacteristics.

As the standard itself mentions, it is not possible, from a practical point of view, to measure all the subcharacteristics in the entirety of a software product, but we can create a complete list of those which are most relevant.

Step 4. Decomposing derived attributes into basic ones

Some of the attibutes obtained in Step 3 can be directly measured, but others may be so abstract that they require more decomposition. In this way, attributes are classified as being 'derived' and 'basic'. Derived attributes should be decomposed until they are completely expressed in terms of those which are basic.

Step 5. Stating relationships between quality entities

If we wish to obtain a complete quality model, we must state the relationships between quality entities. The model becomes more exhaustive and the implications of quality requirements become clearer.

We can classify the relationships into three types:

- Collaboration: Growing the first quality entity implies growing the second quality entity.
- Damage: Growing the first quality entity implies decreasing the second quality entity
- Dependency: Some values of the first quality attribute require that the second one fulfils certain conditions.

Step 6. Determining metrics for attributes.

It is not only necessary to identify the attributes but also to select metrics for all the attributes. For this task, we can use the general theory of metrics.

IV. DATA WAREHOUSE QUALITY MODEL

In this section, we explain how to apply the method described in the previous section to develop a quality model for data warehouse domain. We have developed each of the proposed steps, and we have even related our previously defined metrics to this quality model.

Step 0. Defining the domain

In this step we define the domain for which we want to build a quality model, in this case the conceptual data warehouse schemata. Next, we outline an approach to data warehouse conceptual modeling, based on the UML (Unified Modeling Language). This approach has been specified by means of a UML profile[1] which contains the stereotypes necessary to carry out conceptual modeling successfully [15]. This profile contains the stereotypes which are necessary to elegantly represent main multidimensional (MD) properties at the conceptual level (see Table 1). Specifically, the structural properties of MD modeling are represented by means of a UML class diagram in which the information is clearly organized into facts and dimensions. These facts and dimensions are respectively represented by Fact and Dimension classes. Fact classes are defined as composite classes in shared aggregation relationships of n Dimension classes. The minimum cardinality in the role of the Dimension classes is 1 to indicate that every fact must always be related to all the dimensions. A fact is composed of measures or fact attributes. These are represented as attributes with the FactAttribute stereotype By default, all measures in the Fact class are considered to be additive. For non-additive measures, additive rules are defined as constraints and are included in the Fact class. Furthermore, derived measures (indicated by /) and their derivation rules can also be explicitly represented as tagged values of a FactAttribute.

[1] A *profile* is a set of improvements that extends an existing UML type of diagram to a different use. These improvements are specified by means of the extendibility mechanisms provided by UML (stereotypes, properties and restrictions) in order to be able to adapt it to a new method or model.

Our approach also allows the definition of degenerate dimensions, thereby representing other fact features in addition to the measures for analysis. These degenerated dimensions are represented as stereotyped attributes of the Fact class (DegenerateDimension stereotype).

The many-to-many relationships between a fact and a specific dimension are specified by means of the cardinality 1...n in the role of the corresponding Dimension class. In this case, we usually need to describe specific attributes to provide further features for every instance combination in this particular relationship. In doing so, the provided attributes are usually called degenerate facts. These degenerate facts are represented as an association class attached to a many-to-many aggregation relationship between a Fact class and a Dimension class. This DegenerateFact class may contain FactAttributes and DegenerateDimensions.

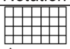
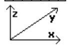
With respect to dimensions, each level of a classification hierarchy is specified by a Base class. Every Base class may contain several dimension attributes (DimensionAttribute stereotype), an identifying attribute (OID stereotype), and must also contain a Descriptor attribute (D stereotype). An association (represented by a stereotype called Rolls-UpTo) between Base classes specifies the relationship between two levels of a classification hierarchy. The only prerequisite is that these classes must define a Directed Acyclic Graph (DAG) rooted in the Dimension class (the DAG constraint is defined in the stereotype Dimension). The DAG structure can represent both multiple and alternative path hierarchies. A Dimension class contains a unique first hierarchy (or dimension) level called a terminal dimension level. A roll-up path is a subsequence of dimension levels, which starts in this terminal level (lower level of detail) and ends in an implicit level (not graphically represented) that represents all the dimension levels.

We use roles to represent the way in which the two Base classes see each other in a Rolls-UpTo association: role R represents the direction in which the hierarchy rolls-up, whereas role D represents the direction in which the hierarchy drills-down. Moreover, we use roles to detect and avoid cycles in a classification hierarchy, and therefore, to help us to achieve the DAG condition.

Due to the flexibility of UML, we can also consider non-strict hierarchies (an object at a hierarchy's lower level belongs to more than one higher-level object) and complete hierarchies (all members belong to one higher-class object and that object consists of those members only). These characteristics are specified, respectively, by means of the cardinality of the roles of the associations and by defining the stereotype Completeness in the association between Base classes. Lastly, the categorization of dimensions is considered by means of the generalization/specialization relationships of UML.

Our profile is formally defined and uses the Object Constraint Language (OCL) to express well-formed rules of the new defined elements (see Table 1), thereby avoiding its arbitrary use. We refer the reader to [15] for a further explanation of this profile and its corresponding OCL constraints.

TABLE 1.
MAIN STEREOTYPES OF OUR UML PROFILE FOR MD MODELING OF DW.

Stereotype	Notation
Fact	
Dimension	
Base	
DegenerateFact	
FactAttribute	FA
DegenerateDimension	DD
DimensionAttribute	DA
OID	OID
Descriptor	D
Rolls-UpTo	<<Rolls-UpTo>>
Completeness	<<Completeness>>

Step 1. Determining quality subcharacteristics

In this step the quality characteristics are decomposed into subcharacteristics following the ISO 9126 standard and the characteristics of the domain we are working with.

The ISO 9126 proposes the decomposition shown in Table 2.

TABLE 2.
ISO/IEC 9126-1 CHARACTERISTICS AND SUBCHARACTERISTICS

Characteristics	Subcharacteristics
Functionality	Suitability Accuracy Interoperability Security
Reliability	Maturity Fault Tolerance Recoverability
Usability	Understandability Learnability Operability Attractiveness
Efficiency	Time behaviour Resource utilization
Maintainability	Analyzability Changeability Stability Testability
Portability	Adaptability Installability Coexistence Replaceability

In our domain, data warehouse conceptual models, we firmly believe that some of the dimensions of the ISO 9126 model are not applicable, since the quality of a conceptual data warehouse schema is related to the correctness, completeness, understandability, security and stability of that schema. Those dimensions that are not related to these characteristics have been removed. In Table 3 we can find the subcharacteristics that are taken into account after removing those that are not applicable to our domain.

TABLE 3. FINAL SET OF QUALITY SUBCHARACTERISTICS

Characteristics	Subcharacteristics
Functionality	Suitability
	Accuracy
	Security
Usability	Understandability
	Learnability
Maintainability	Analyzability
	Changeability
	Stability

Step 2. Defining a hierarchy of subcharacteristics

In our context (quality of conceptual data warehouse models), there is no need to decompose the subcharacteristics stated in the previous step into smaller ones.

Step 3. Decomposing subcharacteristics into attributes

As we have previously stated, quality subcharacteristics provide a comprehensible abstract view of the quality model, but they are too abstract and difficult to measure. In this step, the abstract concepts are descomposed into attributes. For each relevant characteristic and subcharacteristic we provide a set of attributes that are related to that characteristic (see Table 4).

TABLE 4. ATTRIBUTES RELATED TO EACH QUALITY SUBCHARACTERISTIC

Characteristics	Subcharacteristics	Attribute
Functionality	Suitability	Adequacy of data modelled to requirements Completeness Well-Specified Semantic correctness
	Accuracy	Accurate to requirements Semantic correctness
	Security	Presence of security constraints Adequacy of security rules
Usability	Understandability	Ease of understanding Schema Complexity Schema Size Schema Depth
	Learnability	Ease of understanding Schema Complexity Schema Size Schema Depth
Maintainability	Analyzability	Ease of understanding Schema Complexity Schema Size Schema Depth Coupling Cohesion Normalization Modularity
	Changeability	Ease of understanding Schema Complexity Coupling Cohesion Normalization Modularity
	Stability	Schema Complexity Coupling Cohesion Normalization Modularity

Step 4. Decomposing derived attributes into basic ones

After Step 3 we may find that some attributes are too abstract to be easily measured and we should decompose them into more concrete ones. However, the attributes presented in Table 4 are concrete enough, so it is therefore unnecessary to further decompose them.

Step 5. Stating relationships between quality entities

In this step we state the relationships between quality entities, so the model becomes more exhaustive and its implications become clearer. Table 5 shows the relationships between quality attributes. In this table 'COL' means Collaboration, 'DEP' means Dependency and 'DMG' means Damage.

TABLE 5. RELATIONSHIPS BETWEEN QUALITY ATTRIBUTES

	Adequacy of data modelled to requirements	Completeness	Well-Specified	Semantic correctness	Accurate to requirements	Presence of security constraints	Adequacy of security rules	Ease of understanding	Schema Complexity	Schema Size	Schema Depth	Coupling	Cohesion	Normalization
Completeness	COL													
Well-Specified	COL	COL												
Semantic correctness	COL	COL	COL											
Accurate to requirements	COL	COL	COL	COL										
Presence of security constraints	DEP	COL	COL	COL	COL									
Adequacy of security rules		COL		COL		DEP								
Ease of understanding		DMG	DMG	COL	COL	DMG								
Schema Complexity	DMG	DMG	DMG	DMG	DMG	DMG	COL	DMG						
Schema Size	DMG	DMG	DMG	DMG	DMG			DMG						
Schema Depth	DMG	DMG	DMG	DMG	DMG			DMG	COL					
Coupling			DMG	DMG	DMG									
Cohesion			DMG	DMG	DMG							DMG		
Normalization	DMG					DMG		COL				COL	COL	
Modularity				DMG	COL			COL	COL			COL	COL	COL

For example in Table 5, we can see that the Modularity collaborates with the Ease of understanding because if the system is modular it is probably easier to understand. This table is a summary of all the relationships between the quality attributes.

Step 6. Determining metrics for attributes.

After identifying the attributes, we must assign metrics to each attribute. Table 6 shows the proposed metric for each attribute.

TABLE 6.
METRICS ASSOCIATED TO EACH QUALITY CHARACTERISTIC.

Characteristics	*Subcharacteristics*	*Attribute*	*Metric*
Functionality	Suitability	Adequacy of data modelled to requirements	Number of redundant requirements Number of requirements mapped into the model
		Completeness	Number of missing requirements Number of missing attributes Number of missing elements Number of missing relationships Number of requirements not mapped into the model
		Well-Specified	Number of wrong specifications Number of redundant specifications
		Semantic correctness	Number of requirements mapped into the schema Adequacy to requirements
	Accuracy	Accurate to requirements	Number of missing requirements Number of elements that are not represented in the requirements
		Semantic correctness	Number of requirements mapped into the schema Adequacy to requirements
	Security	Presence of security constraints	Number of security constraints Ratio of security constraints
		Adequacy of security rules	Number of security rules Complexity of security rules
Usability	Understandability	Ease of understanding	Number of classes Number of relationships Ratio of relationships Adequacy of names
		Schema Complexity	Number of relationships Ratio of relationships
		Schema Size	Number of classes Number of relationships
		Schema Depth	Length of schema Depth of hierarchy tree
	Learnability	Ease of understanding	Number of classes Number of relationships Ratio of relationships Adequacy of names
		Schema Complexity	Number of relationships Ratio of relationships
		Schema Size	Number of classes Number of relationships
		Schema Depth	Length of schema Depth of hierarchy tree
Maintainability	Analyzability	Ease of understanding	Number of classes Number of relationships Ratio of relationships Adequacy of names
		Schema Complexity	Number of relationships Ratio of relationships
		Schema Size	Number of classes Number of relationships
		Schema Depth	Length of schema Depth of hierarchy tree
		Coupling	Schema coupling

		Cohesion	Schema cohesion
		Normalization	Normalization complaint
		Modularity	Number of modules Ratio of modules
	Changeability	Ease of understanding	Number of classes Number of relationships Ratio of relationships Adequacy of names
		Schema Complexity	Number of relationships Ratio of relationships
		Coupling	Schema coupling
		Cohesion	Schema cohesion
		Normalization	Normalization complaint
		Modularity	Number of modules Ratio of modules
	Stability	Ease of understanding	Number of classes Number of relationships Ratio of relationships Adequacy of names
		Schema Complexity	Number of relationships Ratio of relationships
		Coupling	Schema coupling
		Cohesion	Schema cohesion
		Normalization	Normalization complaint
		Modularity	Number of modules Ratio of modules

V. Conclusions and Future Work

Nowadays, data warehouses have become an important software product used in decision support systems. Due to the environment in which data warehouses are used, assuring the quality of these systems is a crucial issue. When dealing with data warehouse quality, it is important to assess the quality of both data and schemas, as the quality of the data and also the whole system is affected by the quality of the model that supports them.

In order to assess the quality of the data warehouse schema, it is necessary to define a quality model in order to help designers in assuring the quality of the final system as it is being developed in a systematic and objective way.

In this paper we have reviewed a method for creating a quality model based on the ISO 9126 standard and we have applied its six steps to create a first proposal of a conceptual data warehouse quality model.

As a result of this method we have obtained a set of important quality characteristics which are broken down into sub-characteristics and quality attributes. As a final step we have proposed a set of initial metrics for assessing those quality attributes, and we have related them to our previously defined and validated metrics.

The next step in this process of proposing a quality model is to validate it. We are planning to begin with the validation of the proposed metrics in order to assure their usefulness and adequacy. After, we believe that we should attempt to validate the whole quality model in an industrial environment.

References

[1] B. Shin, "An Exploratory Investigaction of System Success Factors in Data Warehousing," *Journal of Association for Information Systems*, vol. 4, pp. 141-170, 2003.

[2] T. Chenoweth, D. Schuff, and R. St. Louis, "A method for developing Dimensional data marts," *Communications of the ACM*, vol. 46, pp. 93-98, 2003.

[3] P. Vassiliadis, "Gulliver in the land of data warehousing: practical experiences and observations of a researcher," presented at International Workshop on Design and Management of Data Warehouses (DMDW'2000), Stockholm (Sweden), 2000.

[4] C. Stedman, "Warehousing Projects Hard to Finish," *Computerworld*, vol. 32, pp. 29, 1998.

[5] N. Prat and S. S.-S. Cherfi, "Multidimensional Schemas Quality Assessment," presented at The 15th Conference on Advanced Information Systems Engineering (CAiSE '03) Workshops, Klagenfurt/Velden (Austria), 2003.

[6] ISO/IEC, "9126-1: Software Engineering - Product quality - Part 1: Quality model.," 2001.

[7] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber, "The Capability Maturity Model for software," Software Engineering Institute, Carnegie Mellon University CMU/SEI-93-TR-024, February 2003 1993.

[8] K. El Emam, J. N. Drouin, and W. Melo, "Spice: The Theory and Practice of Soft-ware Process Improvement and Capability Determination," IEEE Computer Society 1998.

[9] IEEE, "IEEE Std 1061-1998 IEEE Standard for a Software Quality Metrics Methodology," 1998.

[10] V. Basili and D. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering*, vol. 10, pp. 728-738, 1984.

[11] V. Basili and H. Rombach, "The TAME project: towards improvement-oriented software environments," *IEEE Transactions on Software Engineering*, vol. 14(6), pp. 728-738, 1988.

[12] M. Serrano, "Definition of a set of metrics for assuring data warehouse quality," Univeristy of Castilla - La Mancha (Spain), 2004.

[13] M. Jarke, M. Lenzerini, Y. Vassiliou, and P. Vassiliadis, *Fundamentals of Data Warehouses*, second edition ed: Springer-Verlag., 2002.

[14] X. Franch and J. P. Carvallo, "Using Quality Models in Software Package Selection," *IEEE Software*, vol. 20, pp. 34-41, 2003.

[15] S. Luján-Mora, J. Trujillo, and I.-Y. Song, "A UML profile for multidimensional modeling in data warehouses," *Data & Knowledge Engineering (DKE)*, vol. 59, pp. 725–769, 2006.

Integrating Complex Data into a Data Warehouse

F. Ravat[(2)], O. Teste[(1)], R.Tournier[(1)], G. Zurfluh[(2)]

[(1)] IRIT, SIG/ED, Université Toulouse 3,
118 route de Narbonne,
F-31062 Toulouse Cedex 9, FRANCE

[(2)] IRIT, SIG/ED, Université Toulouse 1,
2 rue du doyen G. Marty,
F-31042 Toulouse Cedex 9, FRANCE

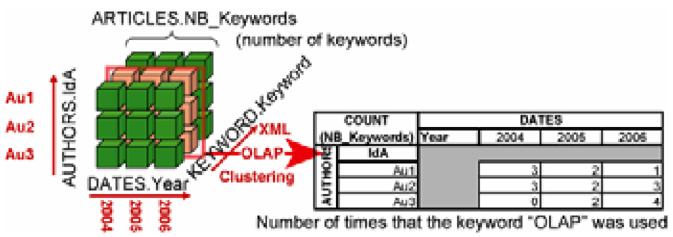
***Abstract*— This paper deals with decision support systems. Nowadays, analysts whish to directly integrate documents into their analyses, in order to improve the reliability of the decision-making process. Thanks to the XML exchange format, unstructured data are now available in a auto descriptive and semi-structured format. In this paper, we provide an approach in order to integrate XML data into a data warehouse. As case study, we propose the multidimensional analysis of XML documents that represent scientific articles. In order to analyse such documents we define a new approach that consists in considering textual data as analysis indicators.**

***Index Terms*—Decision support systems, OLAP, Document Warehouse, XML, Non-additive measure.**

I. Introduction

OLAP (On-Line Analytical Processing) systems allow analysts to improve decision-making process by consulting and analysing aggregated business data. These analyses rest on a centralized data management system: a data warehouse [7].

The use of Multidimensional Databases (MDB) on data warehouse data enables decision-makers to gain insight into enterprise performance. MDB Multidimensional modelling [7] represents data as points in a multidimensional space with the use of the cube metaphor. MDB are modelled through subjects of analysis, named *facts*, and analysis axes, named *dimensions* [7]. These structures are grouped into a star schema [7]. In Fig. 1, the *number of keywords* used in scientific publications is analysed according to three axes: *Authors*, *Dates* and *Keywords* of the publications. A slice of the cube has been extracted and corresponds to the usage of keyword: "*OLAP*".



COUNT	DATES			
(NB_Keywords)	Year	2004	2005	2006
AUTHORS IdA				
Au1		3	2	1
Au2		3	2	3
Au3		0	2	4

Fig. 1 – Cube representation of an MDB.

According to [12] 20% of corporate information system data is transactional and may be processed within OLAP systems. The remaining 80%, namely "paperwork", stays out of reach of OLAP technology due to the lack of tools for non-numeric data or text-rich data. Text is not as structured as a data warehouse would tolerate. But recently XML[1] technology has provided a wide framework for working with documents. XML stores data with an auto-descriptive formalism: a DTD (Document Type Definition). Thus, documents stored as semi-structured data were integrated within repositories and document warehousing [13] emerged, e.g. Xyleme[2]. Structured or semi-structured documents are now becoming a conceivable data source for multidimensional analysis.

We argue that, to provide more exhaustive multidimensional analyses, OLAP decision support systems should provide the use of a 100% of corporate information system data. Analysts should be able to integrate text-rich documents into the analysis process along with more traditional business data. Not taking into account these data sources would inevitably lead to the omission of relevant information during an important decision-making process or the inclusion of irrelevant information and thus producing inaccurate analyses [12].

Amongst the problems due to the integration and processing of complex data within a data mart, this paper focuses on how to feed a multidimensional conceptual schema with complex XML data. In this study we work on the integration of XML documents representing scientific articles in order to assemble a data mart to provide analysis of these documents.

As in [3], we distinguish two types of XML documents: *data-centric documents* which are transactional data stored within documents, such as spreadsheets, logs, transactions (…); and *document-centric documents* which are text-rich documents such as reports, articles, e-books (…).

The integration of data-centric documents in MDB has been introduced in several works such as [4]. See [15] and [16] for examples and a more complete list of works. In [1], the authors focus on the integration of complex data [2], but remain in the domain of data-centric documents. None of these propositions focus on the integration of complex data as factual data.

In [6], the authors describe a document warehouse [13], where documents are grouped by similar structure. In [9], the authors use the xFACT framework [8] for the analysis of document-centric documents. The authors suggest the use of text mining techniques for the aggregation of textual data,

[1] XML, Extended Markup Language from http://www.w3.org/XML
[2] Xyleme server from http://www.xyleme.com

(heart of OLAP analyses). In [11] we defined such a function, providing qualitative analysis. These propositions are limited concerning loading XML document-oriented documents within an MDB.

Complex data warehousing is a recent term [2]. But up to now the most advanced propositions offer filtering and cleaning solutions in order to obtain traditional data that allow quantitative analyses based on numeric measures. In order to provide a more complete OLAP framework that goes further than the analysis of numeric data, our objective is to provide an enriched OLAP framework with increased analysis capabilities. Amongst the numerous problems to be tackled, this paper focuses on loading data in an MDB. To answer to our objectives, we modify the star schema [7]. In [10], the authors present a tool for integrating XML data within a relation database. In the same way, we whish to allow a user integrate XML data within an MDB. We also propose an approach for the integration of complex data as factual data.

The rest of the paper is organised as follows: section 2 defines the conceptual model on which rests our proposition; and section 3 deals with the approach to integrate complex data within multidimensional structures.

II. Conceptual Model: Textual Constellation

For analysis purpose, data are organised multidimensionally in an MDB. We propose to extend the star schema [7] into a textual constellation.

A. Formal Definition

A *textual constellation schema TS* is defined by $TS = (F^{TS}, D^{TS}, Star^{TS})$ where $F^{TS} = \{F_1,\ldots,F_m\}$ is a set of facts; $D^{TS} = \{D_1,\ldots,D_n\}$ is a set of dimensions and $Star^{TS}: F^{TS} \to 2^{D^{TS}}$ is a function that associates each fact to its linked dimensions[3]. If F^{TS} is a singleton, *TS* is called a *textual star schema*.

A *fact F* is defined by $F = (M^F, I^F, IStar^F)$ where $M^F = \{M_1,\ldots, M_w\}$ is a set of *measures*; $I^F = \{i^F_1,\ldots, i^F_q\}$ is a set of *fact instances*; and $IStar^F: I^F \to I^{D1}\times\ldots\times I^{Dn}$ is a function which respectively associates fact instances to their linked dimension instances.

A *measure M* is defined by $M = (m, f_{AGG})$ where *m* is the measure and $f_{AGG} = \{f_1,\ldots,f_x\}$ is a set of aggregation functions compatible with the measure's additivity type. Measures may be additive, semi-additive or non-additive [7], [5]. In contrast, where the standard framework has only one type of measure, we define two measure types: numeric and textual measures. A *numeric measure* is exclusively composed of numeric data and is either additive or semi-additive. With an *additive measure*, all traditional aggregation functions may be used. *Semi-additive* measures represent snapshot measures (e.g. temperatures, stock quantities…) and only certain aggregation functions may be used. A *textual measure* is a measure composed of text and is always non-additive. This text may represent a word, a paragraph or even a whole document. With *non-additive measures*, only generic aggregation functions may be used (such as COUNT and LIST). In [9], the authors suggest the use of specific aggregation functions, such as TOP_KEYWORDS that returns the major keywords of a text and SUMMARY that generates the summary of a text. AVG_KW, defined in [11], combines several keywords into an "average" keyword.

[3] The notation 2^D represents the powerset of *D*.

A *dimension D* is defined by $D = (A^D, H^D, I^D)$ where $A^D = \{a^D_1,\ldots,a^D_u\}$ is a set of *attributes* (parameters and weak attributes); $H^D = \{H^D_1,\ldots,H^D_x\}$ is a set of hierarchies that represent the attribute organisation and $I^D = \{i^D_1,\ldots,i^D_p\}$ is a set of *dimension instances*.

A *hierarchy H* is defined by $H = (Param^H, Weak^H)$ where $Param^H = \langle p^H_1,\ldots, p^H_{np}, \rangle$ is an ordered set of attributes, called *parameters* (with $\forall k\in[1..np]$, $p^H_k\in A^D$, and $\forall H\in H^D$ a common root parameter to all hierarchies of *D*: $p^H_1 = a^D_1$,); and $Weak^{Hi} : Param^{Hi} \to 2^{A^D - Param^{Hi}}$ is a function that possibly specifies associations of parameters to other attributes called *weak attributes*, which complete the parameter semantic.

B. Example

A decision-maker analyses a collection of scientific articles. This analysis integrates two indicators: the acceptance rate and the textual content of the articles. This analysis is done according to the date of publication, the conference or journal where it was published, the authors and the structure of the document. See Fig. 2 for the representation of the constellation. Note that the *Text* measure corresponds to the content of the fact, i.e. a scientific article. Although slightly different from one another, all these articles are composed of sections, possibly sub-sections and paragraphs (described by the *Structure* dimension).

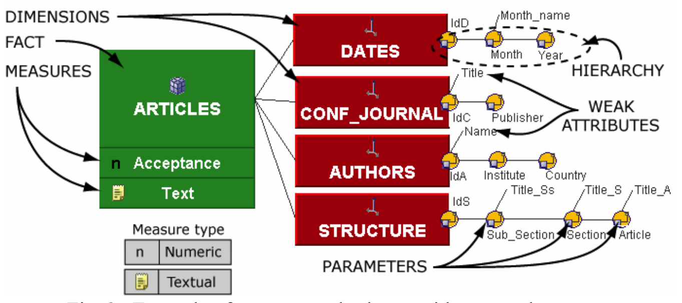


Fig. 2 –Example of a conceptual schema with a textual measure.

III. Integration of Complex Data

To integrate XML data within a MDB, we use a mixed approach. It consists in 1) a top-down approach, e.g. [7], that consists in designing MDB from user requirements; and 2) a bottom-up approach, e.g. [15] that consists in building a MDB according to data sources.

A. Conceptual Design of the MDB

User requirements are expressed through a multidimensional conceptual schema. We use a textual constellation to that end. Fig. 2 shows a textual constellation for scientific publication analysis. The resulting schema is then adapted to be compatible with the XML sources.

Sources are composed of XML documents homogeneously structured. Structures are specified by a DTD (Fig. 3) which defines the different elements that compose XML documents. These elements are hierarchically structured and may have different granularities (e.g. one-to-many relationships represented by "+"). We display the DTD as a tree (see Fig. 4).

```
<?xml version="1.0" encoding="UTF-8"?>
<!--file Article.dtd-->
<!--Root element-->
<!ELEMENT Article (PUBLICATION, AUTHOR+, ARTICLE_CONTENT)>
<!ELEMENT PUBLICATION (Year, Month, Publisher, Publication_Title)>
<!ELEMENT AUTHOR (Name, Institute, Country)>
<!ELEMENT ARTICLE_CONTENT (Title_A, Section+, References)>
<!ELEMENT Section (Title_S, Sub_Section+)>
<!ELEMENT Sub_Section (Title_Sub, Paragraph+)>
<!ELEMENT References (Entry_Ref+)>
<!--Leaves of the XML document tree structure -->
<!ELEMENT Year (#PCDATA)>
<!ELEMENT Month (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
<!ELEMENT Publication_Title (#PCDATA)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Institute (#PCDATA)>
<!ELEMENT Country (#PCDATA)>
<!ELEMENT Title_A (#PCDATA)>
<!ELEMENT Title_S (#PCDATA)>
<!ELEMENT Title_Sub (#PCDATA)>
<!ELEMENT Paragraph (#PCDATA)>
<!--for simplification reasons Entry_Ref is not detailed-->
```

Fig. 3 – Example of a DTD that defines a collection of XML documents.

XML source analysis allows the identification of available factual and dimensional data. Thus, it is easy to verify conformity and viability of the conceptual schema derived from user requirements. The conceptual schema is revised when incompatibilities are detected, adapting elements to be compatible or deleting them (e.g. due to the absence of data).

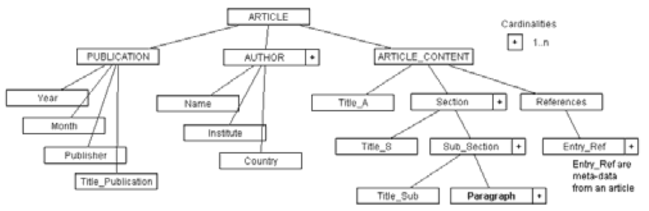


Fig. 4 – Tree-like representation of the previous DTD (leaves content are not presented).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ARTICLE SYSTEM "Article.dtd">
<ARTICLE>
 <PUBLICATION>
     <Year>2006</Year>
     <Month>September</Month>
     <Publisher>Springer</Publisher>
     <Publication_Title>DaWaK Proceedings</Publication_Title>
 </PUBLICATION>
 <AUTHOR>
     <Name>Author1</Name>
     <Institute>Institue1</Institute>
     <Country>France</Country>
 </AUTHOR>
 <AUTHOR> ... </AUTHOR>
 <ARTICLE_CONTENT>
     <Title_A>Title of the article</Title_A>
     <Section>
         <Title_S>Title of section 1</Title_S>
         <Sub_Section>
             <Title_Sub>Title of subsection 1.1</Title_Sub>
             <Paragraph>text of paragraph</Paragraph>
             <Paragraph> text of paragraph </Paragraph>
         </Sub_Section>
         <Sub_Section> ... </Sub_Section>
     </Section>
     <Section>
         <Title_S>Title of section 2</Title_S>
         <Sub_Section> ... </Sub_Section>
     </Section>
     <References>
         <Entry_Ref>First reference</Entry_Ref>
         <Entry_Ref>Second reference</Entry_Ref>
     </References>
 </ARTICLE_CONTENT>
</ARTICLE>
```

Fig. 5 – Example of an XML document respecting the previous DTD.

B. *Integration: Loading the MDB*

To link multidimensional elements of the schema to XML source data, the analyst follows a set of *identification rules*. Contrarily to other works, such as [1], XML sources are not transformed as this would: 1) cost time; and 2) require complex extraction and transformation rules. Our approach is based on the integration of application-oriented XML documents rather than OLAP-oriented documents. In our context factual information is held in leaves of XML structure.

Rule 1. Factual data is one of the elements with the finest granularity within the XML document.

E.g. the leaf nodes *paragraph* represent all the text of an article and are factual data (see **Erreur ! Source du renvoi introuvable.**).

Rule 2. Dimensional data is disseminated within the XML document with a coarser granularity than factual data.

In our example, the node *paragraph* has a finer granularity than any other node (*year*, *section*…). For one article there is one date record but many paragraphs.

Rule 3. Hierarchical parameter data may be found within nodes with one-to-many relationships. All nodes with a coarser granularity than a node defining factual data may correspond to a hierarchy of parameters of a dimension.

In our example, the hierarchy of nodes below *ARTICLE_CONTENT* represent the *Structure* dimension.

Rule 4. Some nodes at the same granularity level may define hierarchical data. This is done with semantic constraints.

E.g. the parameters of the *Dates* dimension (*Year* and *Month*) are at the same level within the DTD tree. Their hierarchical order may only be determined by semantic of the nodes and the knowledge of the analyst. As this represents a date, the user will designate that *years* include *months*.

Following these rules, the analyst establishes links between nodes representing the XML elements of the data sources and the elements of the conceptual schema (measures, parameters and weak attributes). Fig. 7(a) shows the association between nodes of the XML arborescence that correspond to dimensional data and the conceptual elements that correspond to the dimensions (for simplicity reasons only the elements that correspond to the *Dates* and *Structure* dimensions are represented). Fig. 7(b) shows the association for the factual data.

Once conceptual elements and XML nodes have been associated, the system follows a series of *cleaning rules* to validate the conceptual schema.

Rule 1. Any weak attribute not linked to an XML node is removed.

Rule 2. Any parameter not linked to an XML node is removed, except if a weak attribute linked to it is linked to a source node. The parameter will be filled with key data.

For example, in Fig. 7(a)-grey circle, the lowest granularity for the *Dates* dimension is *Month*. No link was established to integrate data in the root parameter: *IdD*. This parameter being

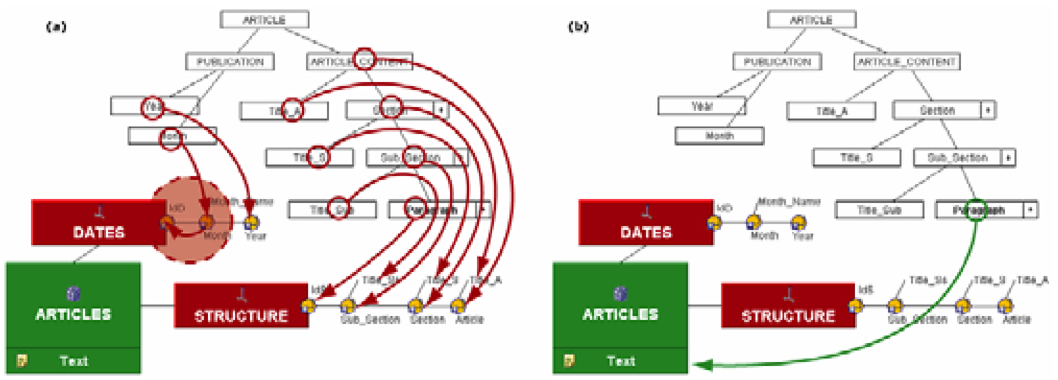


Fig. 6 – Integration of XML source data with a constellation schema (not all elements are represented): (a) dimensional data and (b) factual data.

useless is removed. *Month* becomes the new root parameter, once the user has confirmed the removal of the element.

IV. Conclusion

In this paper we have presented an approach for the integration of complex data within an MDB. We have modified the traditional star schema [7] and introduced within a constellation the concept of textual measure in order to handle the analysis of documents-centric documents. Due to lack of space, we invite the reader to read [11] for more details on textual measures. A set of rules allows a user to load complex data within the MDB and specially to integrate complex factual data. Contrarily to other works, this approach is based on the absence of pre-processing of XML text-rich data sources.

As 80% of corporate information system data is composed of document-centric documents that may not be managed by current decision support systems [12]. We believe that, associated to adapted aggregation functions [11], this will allow the combination of qualitative analysis to quantitative analysis and open a new window on decision support.

We are currently implementing a Java CASE tool (Computer-Aided Software Engineering) based on a ROLAP data warehouse for storing multidimensional data and XML files for storing complex data. An SQL engine associated to an XQuery engine are in charge of linking relational and XML data.

Nowadays, OLAP systems do not provide a complete environment for textual or complex data; we consider as future work to continue the specification of adapted aggregation functions.

References

[1] Boussaid B., Messaoud R.B., Choquet R., Anthoard S., "X-Warehousing: An XML-Based Approach for Warehousing Complex Data", *10th East European Conference Advances in Databases and Information Systems* (*ADBIS*), LNCS 4152, Springer, pp. 39–54, 2006.

[2] Darmont J., Bousaid O., *Complex Data for Decision Support*, Idea Group Publishing , ISBN: 159140655-2, 2006.

[3] Fuhr, N., Großjohann, K., "XIRQL: A Query Language for Information Retrieval in XML Documents", *24th int. ACM conf. on research and development in Information Retrieval* (*SIGIR*), ACM Press, pp. 172–180, 2001.

[4] Golfarelli M., Rizzi S., Vrdoljak B., "Data Warehouse Design from XML Sources", *4th ACM Int. Workshop on Data Warehousing and OLAP* (*DOLAP*), ACM Press, pp. 40–47, 2001.

[5] Horner J., Song I-Y., Chen P.P., "An analysis of additivity in OLAP systems", in *ACM 7th Int. Workshop on Data Warehousing and OLAP* (*DOLAP*), ACM, pp. 83–91, 2004.

[6] Khrouf K., Soulé-Dupuy C., "A Textual Warehouse Approach: A Web Data Repository", *Intelligent Agents for Data Mining and Information Retrieval*, Masoud Mohammadian (Eds.), Idea Publishing Group, ISBN: 1-59140-277-8, pp. 101–124, 2004.

[7] Kimball R., 1996. "*The data warehouse toolkit*", Ed. John Wiley and Sons, 1996, 2nd ed. 2003.

[8] Nassis V., Rajugan R., Dillon T.S., Rahayu J.W., "Conceptual Design of XML Document Warehouses", *6th Int. Conf. on Data Warehousing and Knowledge Discovery* (*DaWaK*), LNCS 3181, Springer, pp. 1–14, 2004.

[9] Park B-K., Han H., Song I-Y., "XML-OLAP: A Multidimensional Analysis Framework for XML Warehouses", *6th Int. Conf. on Data Warehousing and Knowledge Discovery* (*DaWaK*), LNCS 3589, Springer, pp.32–42, 2005.

[10] Popa L., Hernández M.A., Velegrakis Y., Miller R.J., Naumann F., Ho C-T., "Mapping XML and Relational Schemas with Clio", *18th Int. Conf. on Data Engineering* (*ICDE*), IEEE Computer society, pp.498–499, 2002.

[11] Ravat, F., Teste, O., Tournier, R.. "OLAP Aggregation Function for Textual Data Ware-house", *9th Int. Conf. on Enterprise Information Systems* (*ICEIS*), INSTICC Press, June 2007 (to appear).

[12] Ravat, F., Teste, O., Tournier, R., Zurfluh, G., "Algebraic and graphic languages for OLAP manipulations", *Int. j. of Data Warehousing and Mining* (*ijDWM*), IDEA Group Publishing, 2007 (to appear).

[13] Sullivan D., *Document Warehousing and Text Mining*, Wiley John & Sons, ISBN: 0471399590, 2001.

[14] Tseng F.S.C., Chou A.Y.H, "The concept of document warehousing for multi-dimensional modeling of textual-based business intelligence", *J. of Decision Support Systems* (*DSS*), vol.42(2), Elsevier, pp. 727–744, November 2006.

[15] Vrdoljak B., Banek M., Skocir Z., "Integrating XML into a Data Warehouse", *2nd Int. Workshop on Data Engineering Issues in E-Commerce and Services* (DEECS 2006), LNCS 4055, Springer, pp. 133–142, 2006.

[16] Yin X., Pedersen T.B., "Evaluating XML-extended OLAP queries based on a physical algebra", *7th ACM Int. Workshop on Data Warehousing and OLAP* (*DOLAP*), ACM, pp.73–82, 2004.

Learning from Software Quality Data with Class Imbalance and Noise

Andres Folleco
Taghi M. Khoshgoftaar*
Jason Van Hulse
Chris Seiffert

Abstract

The objective of this study is to provide an empirical analysis of the effects of learning from imbalanced and noisy software measurement data. We observe the impact of four levels of imbalance and three levels of class noise on the performance of 11 learning algorithms using real-world software quality data. Analysis of the results demonstrate a significant relationship between learner performance, level of class imbalance, and class noise.

Keywords: class noise, class imbalance, software quality classification.

1 Introduction

The objective of software quality classification initiatives is to categorize instances (program modules) as fault-prone (fp) or not fault-prone (nfp) by inferring a 'model' from previously labeled examples. The timely application of classification models can assist in directing quality improvement efforts to modules that are likely to be fault-prone during operation, thereby utilizing the software quality testing and enhancement resources in a cost-effective manner. One commonly-encountered difficulty in software quality classification is that the difference in the number of instances belonging to each class can be severe. In binary classification (the only type studied in this work), classes fp and nfp are $imbalanced$ if $\pi_{fp} \neq \pi_{nfp}$, where π_{nfp} and π_{fp} are the proportion of class nfp and fp examples. In general, $\pi_{fp} < \pi_{nfp}$, and fp is called the $minority$ class, while nfp is called the $majority$ class. Little emphasis within the software quality classification domain has been placed on studying the relationship between learning (classification) algorithms, class imbalance, and class noise in software measurement data, despite the fact that these are critical issues. In particular, many software measurement datasets heavily favor the nfp class, such as datasets collected from high-assurance software systems [8]. Standard classification algorithms may not perform optimally when learning from skewed data. Therefore, in this work, we address the following research questions:

- Which learners are robust in the presence of class noise? Increasing the level of class noise shows significant performance impacts on most learners. The impact depends heavily on the learner used: some learners, such as Naive Bayes, are relatively unaffected by class noise, while other learners perform worse with noisier data.
- Is learner performance impacted by class imbalance? In other words, as the class imbalance becomes more severe, what happens to the performance of the classifiers?
- How do learners perform when confronted with different levels of class imbalance and class noise? Our experimental results have led us to conclude that some learners that are robust to noise are also good at handling more severe class imbalance.

The contribution of this work is to address the above three research questions, and in particular to empirically study classification performance using imbalanced and noisy software measurement data. To the best of our knowledge, this important topic has not been examined previously.

1.1 Related Work

Most related work addresses class imbalance and class noise separately without considering their simultaneous effects. Weiss and Provost [13] address the optimal class distribution for decision tree construction when the amount of training data must be limited. Although they consider various levels of imbalance, they use a single learner, C4.5, while varying the number of positive and negative examples. Other work has focused on learning from imbalanced datasets [2, 4, 5, 6], comparing two or more procedures and

*Readers may contact the authors through Taghi M. Khoshgoftaar, Empirical Software Engineering Laboratory, Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431 USA. Phone: (561)297-3994, Fax: (561)297-2800, Email: taghi@cse.fau.edu.

proposing new techniques. Zhu and Wu [16] conducted a study using class and attribute noise concluding that both have significant impact on classification, while the former had a more severe impact. A related study [17] investigated the impact of noisy data in cost-sensitive learning. Recent work by our research group has included the use of frequent itemsets to detect noisy examples in software measurement data [12]. While our group has recently performed tests in the domain of class imbalance [10], we now present a systematic analysis of different levels of class imbalance and class noise in combination with a significant number of learners applied to software measurement data.

The paper is organized as follows: Section 2 describes the 11 learners used. The experimental methodology are provided in Section 3. Experimental results are in Sections 4, with the conclusions provided in Section 5.

2 Classifiers

All learners were implemented in the WEKA tool [14]. Default parameter changes were done only when experimentation showed a general improvement in the classifier performance based on preliminary analysis.

Naive Bayes (NB) utilizes Bayes's rule of conditional probability and is termed 'naive' because it assumes independence of the features. *Logistic regression* (LR) is a statistical regression model for categorical prediction. *RIPPER* (Repeated Incremental Pruning to Produce Error Reduction) is a rule-based learner and is named JRip in WEKA. The *random forests* (RF) classifier [3] uses bagging and the 'random subspace method' to build an ensemble of randomized decision trees which are combined to produce the final prediction. The default WEKA parameters for these four learners were not changed.

C4.5 is a benchmark decision tree learning algorithm. Two different versions of the C4.5 classifier were used. C4.5 (D) uses the default parameter settings in WEKA, while C4.5 (N) uses no decision-tree pruning and Laplace smoothing [13]. For a *Multilayer perceptrons* (MLP) learner (a type of neural network), the 'hiddenLayers' parameter was changed to '3' to define a network with one hidden layer containing three nodes, and the 'validationSetSize' parameter was changed to '10' to cause the classifier to leave 10% of the training data aside to be used as a validation set to determine when to stop the iterative training process. *Radial basis function networks* (RBF) are another type of artificial neural network. The only parameter change for RBF was to set the parameter 'numClusters' to 10. Two *K nearest neighbors* [1] (kNN) learners were built with changes to two parameters. The code implementing this was also changed so that it chooses the number of nearest neighbors which maximizes the average of the accuracies on each class, rather than the overall accuracy. The 'distanceWeighting' parameter was set to 'Weight by 1/distance'. Two different 'kNN' learners were built using $k = 2$ and $k = 5$ and were denoted '2NN' and '5NN'. The *support vector machine* (SVM) learner called SMO in WEKA had two changes to the default parameters: the complexity constant 'c' was set to 5.0 and 'buildLogisticModels' was set to 'true'.

Independent Variables
Logical Operators
Total Lines of Code
Executable Lines of Code
Unique Operands
Total Operands
Unique Operators
Total Operators
Cyclomatic Complexity
Dependent Variable
$nfaults$

Table 1. CCCS Dataset Software Metrics

3 Experimental Design

We tested the effects of four levels of class imbalance and three levels of class noise on classifiers created using C4.5(D), C4.5(N), NB, MLP, 2NN, 5NN, RIPPER, LR, RBF, RF, and SVM learners.

3.1 CCCS Dataset

The CCCS dataset is a military command, control and communications system [7]. CCCS has 282 instances (program modules), where each instance is an Ada package consisting of one or more methods. CCCS has eight software metrics which are used as independent variables. An additional attribute $nfaults$ (the dependent variable) indicates the number of faults attributed to a program module. Table 1 lists the software metrics in CCCS. Additional studies may consider different software metrics, and the objective of our experiments is not to analyze the viability of these particular metrics. CCCS is denoted $\mathcal{C}^o$ in this study. The natural distribution of $nfaults$ in $\mathcal{C}^o$ has over 50% of the program modules with no faults (i.e., $nfaults = 0$), and approximately 19% of the program modules had one fault. The median value of $nfaults$ is 0, the largest value is 42 and the mean is 2.369.

3.2 Cleansing CCCS

Since $\mathcal{C}^o$ is a real-world dataset, it has some instances which have a natural occurring noisy value for $nfaults$, so-called *inherent* noise. *Injected* noise is noise that is artificially introduced into the dataset. Generating realistic examples of noise in a domain-sensitive manner is a difficult research issue, and is critically important because measuring the results of any technique using unrealistic noise can be misleading since it may not represent the types of noise found in real-world datasets. Using a hybrid procedure introduced for cleansing noise from a continuous dependent variable [9], a software engineering expert oversaw the cleansing of $\mathcal{C}^o$. The hybrid procedure also determined a clean value for $nfaults$ (denoted $nfaults^c$) for the instances deemed to be noisy. 81 instances were identified as having inherent noise in $nfaults$ (denoted $nfaults^n$). A detailed description of the cleansing process is presented in our previous work [9].

3.3 Injected Noise

Two additional datasets were derived from $\mathcal{C}^o$, denoted $\mathcal{C}^{5p}$ and $\mathcal{C}^{10p}$. Note that all three of these datasets have the same number of instances (282) and the same number of attributes (9). These datasets were created as follows:

$\mathcal{C}^{5p}$: Starting with the original dataset $\mathcal{C}^o$, the software engineering expert inspected the dataset and identified 14 instances (or 5% of the instances in $\mathcal{C}^o$) that were relatively clean. These 14 instances were then corrupted (with respect to $nfaults$) in an expert-supervised manner such that the noise was reasonable for the given dataset and that the noisy value was different than the clean value[1].

$\mathcal{C}^{10p}$: In addition to the 14 clean instances with injected noise in the dependent variable in $\mathcal{C}^{5p}$, the software engineering expert identified 14 more relatively clean instances for corruption. Noise was injected in a manner similar to that of the first 14 instances, resulting in the $\mathcal{C}^{10p}$ dataset. $\mathcal{C}^{10p}$ therefore has 28 instances (or 10% of the dataset) with injected noise in $nfaults$.

$\mathcal{C}^o$ has 81 noisy examples with respect to $nfaults$, and for these 81 instances both the noisy value $nfaults^n$ and clean value $nfaults^c$ are known [9]. $\mathcal{C}^{5p}$ has 81 inherently noisy examples in addition to 14 instances with injected noise. For the injected noise, $nfaults^c$ is the value that was originally in the dataset, and $nfaults^n$ is the corrupted value. Therefore relative to $nfaults$, $\mathcal{C}^{5p}$ has a total of $81 + 14 = 95$ noisy examples. The situation for $\mathcal{C}^{10p}$ is similar, however there are a total of $81+28 = 109$ noisy examples. For the remaining examples in each of these three datasets, we set $nfaults^n = nfaults^c$. Therefore, both a noisy value $nfaults^n$ and a clean value $nfaults^c$ are available.

[1]In other words, the noise injected into the dataset was such that it might realistically occur in real-world measurement data. This is in contrast to randomly corrupting $nfaults$ to a possibly unrealistic value for the given domain.

Dataset	$\#fp$	$\#nfp$	$\%fp$	$\%nfp$
C_4^*	56	226	19.86	80.14
C_6^*	35	247	12.41	87.59
C_8^*	27	255	9.57	90.43
C_{12}^*	19	263	6.74	93.26

Table 2. CCCS Levels of Imbalance in L^c

3.4 Level of Imbalance

From $\mathcal{C}^o$, $\mathcal{C}^{5p}$, and $\mathcal{C}^{10p}$ a total of twelve new datasets with a binary class L are derived. Let $\mathcal{C}^*$ denote any of the three initial datasets, and let $\lambda \in \{4, 6, 8, 12\}$ denote a threshold on the dependent variable $nfaults$ in $\mathcal{C}^*$. $\mathcal{C}^*$ is transformed to the dataset $\mathcal{C}^*_\lambda$ by replacing $nfaults$ with a binary class attribute L according to the following rule:

$$L_\lambda^\blacktriangle(x) = \begin{cases} nfp & \text{If } nfaults^\blacktriangle(x) < \lambda \\ fp & \text{otherwise} \end{cases}$$

where $\blacktriangle \in \{c, n\}$ with $(c = clean, n = noise)$ and $L_\lambda^\blacktriangle(x)$ is the class label of instance x using the clean value $nfaults^c(x)$ or the noisy value $nfaults^n(x)$. From the perspective of software quality modeling, instances are divided into two classes, fp and nfp. Since the purpose of software quality modeling is to identify modules likely to have more faults, classifiers are constructed to correctly categorize as many fp (true positive) and nfp (true negative) instances as possible. Resources can then be directed toward at-risk software modules (true positives) with minimal wasted effort on modules with few or no faults that were incorrectly predicted as fault-prone (false positives).

Since an instance x is labeled as fp only when $nfaults(x) \geq \lambda$, increasing the value of λ reduces the number of instances labeled as fp in the dataset. Therefore, increasing the value of λ also increases the level of imbalance in L. Four values of λ are used in our experiments, resulting in four levels of imbalance in 12 derived datasets. $\mathcal{C}_4^o, \mathcal{C}_6^o, \mathcal{C}_8^o$, and $\mathcal{C}_{12}^o$ are the four datasets with a binary class label derived from $\mathcal{C}^o$ using the threshold $\lambda \in \{4, 6, 8, 12\}$. $\mathcal{C}_4^{5p}, \mathcal{C}_6^{5p}, \mathcal{C}_8^{5p}$, and $\mathcal{C}_{12}^{5p}$ were derived from dataset $\mathcal{C}^{5p}$, while $\mathcal{C}_4^{10p}, \mathcal{C}_6^{10p}, \mathcal{C}_8^{10p}$, and $\mathcal{C}_{12}^{10p}$ were derived from dataset $\mathcal{C}^{10p}$. Table 2 shows the number of fp and nfp instances in the data with respect to L^c (the clean class value). Since this table is based on L^c, the number of fp and nfp instances are the same for the three datasets with a given λ. In other words, $nfaults^c$, and hence L_λ^c, is the same for C^o, C^{5p}, and C^{10p} for a fixed value λ. The number of fp and nfp instances with respect to L^n at a given λ varies in the three datasets because $nfaults^n$ changes for some instances significantly (not shown due to space limits).

Data	$\%n\to p/p$	$\%n\to p/n$	$\%p\to n/n$	$\%p\to n/p$	$\%noise$
C_4^o	25.45	6.19	6.61	26.79	10.28
C_6^o	22.86	3.24	3.24	22.86	5.67
C_8^o	29.63	3.14	3.14	29.63	5.67
C_{12}^o	37.5	2.28	3.38	47.37	5.32
C_4^{5p}	38.1	10.62	7.76	30.36	14.54
C_6^{5p}	41.86	7.29	4.18	28.57	9.93
C_8^{5p}	45.45	5.88	3.61	33.33	8.51
C_{12}^{5p}	50	3.42	3.79	52.63	6.74
C_4^{10p}	49.32	15.93	9.09	33.93	19.5
C_6^{10p}	56.6	12.15	5.24	34.29	14.89
C_8^{10p}	57.89	8.63	4.51	40.74	11.7
C_{12}^{10p}	59.09	4.94	3.85	52.63	8.16

Table 3. Dataset Class Noise Characteristics

3.5 Class Noise

The number of noisy instances in the datasets prior to the transformation of $nfaults$ to the class label L is not necessarily the same as the number of noisy instances in the post-transformation datasets. A class L for an instance x will only be noisy if the value of λ falls between $nfaults^n(x)$ and $nfaults^c(x)$. For example, if an instance x with $nfaults^n(x) = 7$ and $nfaults^c(x) = 10$ will not be identified as noisy for $\lambda = 6$, since $L_6^c(x) = L_6^n(x) = fp$. Since both $nfaults^n(x)$ and $nfaults^c(x)$ are greater than 6, x will be correctly labeled as fp even though $nfaults^n(x) \neq nfaults^c(x)$. However, if $\lambda = 8$ this same instance will be incorrectly labeled as nfp (since $nfaults^n(x) < \lambda$) when it should be labeled as fp (since $nfaults^c(x) \geq \lambda$). Lastly, this instance would be correctly labeled as nfp for $\lambda = 12$, since both $nfaults^n$ and $nfaults^c$ are less than 12.

Table 3 shows the class noise percentage in each of the 12 datasets. Noisy instances are denoted as $x\to y$, where x and y indicate whether L^c and L^n, respectively, belong to the negative (n) or positive (p) class. Column $\%n\to p/p$ has the percent of instances labeled fp that should have been labeled nfp. Column $\%n\to p/n$ has the percent of instances with $L^c = nfp$, but $L^n = fp$. Column $\%p\to n/n$ has the percent of instances labeled nfp that should have been labeled fp. Column $\%p\to n/p$ has the percent of instances with $L^c = fp$, but $L^n = nfp$. Column $\%noise$ has the percent of all instances incorrectly labeled:

$$\%p\to n/p = \frac{\#p\to n}{\#p\to p + \#p\to n} \quad (1)$$

$$\%p\to n/n = \frac{\#p\to n}{\#n\to n + \#p\to n} \quad (2)$$

$$\%n\to p/n = \frac{\#n\to p}{\#n\to n + \#n\to p} \quad (3)$$

$$\%n\to p/p = \frac{\#n\to p}{\#p\to p + \#n\to p} \quad (4)$$

3.6 Classifier Evaluation

We utilize a widely-known performance metric called the area under the ROC curve (AUC) to evaluate learner performance in our experiments. The ROC curve graphs the true positive rate on the y-axis versus the false positive rate on the x-axis, and therefore measures the tradeoff between detection rate and false alarm rate. The AUC ranges from zero to one, with higher values denoting a classifier with generally better performance (in general, a higher AUC implies a higher true positive rate with a lower false positive rate, which is preferred in most applications). Two different classifiers can be evaluated by comparing their AUC values. Provost and Fawcett [11] give an extensive overview of ROC curves and their potential use for optimal classification.

3.7 Summary of the Experimental Design

Performance is measured using 10-fold cross validation (CV) with 30 independent replications for each experiment, making all experiments performed in this work very comprehensive and statistically significant. With 10-fold CV, nine folds are used as a training dataset and one fold is the hold-out (test) dataset. When constructing a classifier using the training dataset, the noisy label L^n was used, as the objective of our study is to examine the impact of noise on learning. In other words, class noise is in the training dataset, but *not* in the test dataset. Therefore for the hold-out partition in 10-fold CV, the classifier's predictions are compared to L^c. So for each dataset C^*, $nfaults^c$ and $nfaults^n$ were transformed using λ to create two class attributes, L_λ^c and L_λ^n. A classifier is constructed using 10-fold CV, with L_λ^n as the class label in the training dataset (nine folds of CV), and evaluated against L_λ^c in the test dataset (the hold-out fold of CV). A total of 39,600 learners (= 12 datasets $\times$ 10-fold CV $\times$ 30 repetitions $\times$ 11 learning algorithms) were constructed in these experiments.

4 Experimental Results

4.1 Effect of Imbalance and Noise on AUC

The mean AUC, calculated from all 12 datasets, is provided in Table 4. The two best learners were NB and SVM (rank of one and two, respectively). MLP also performed very well, and 5NN obtained a higher AUC than 2NN. RBF obtained the worst performance of all learners, and RIPPER and C4.5 (D) also performed very poorly.

Table 5 shows the percent change of the mean AUC calculated between $\mathcal{C}^o$ and $\mathcal{C}^{10p}$ by each level of imbalance $\lambda \in \{4, 6, 8, 12\}$. NB has one of the smallest changes in AUC across all levels of class imbalance. The change in AUC for NB from $\mathcal{C}^o$ to $\mathcal{C}^{5p}$ at each level of class imbalance

Learner	Mean AUC	Rank
5NN	0.9493	4
MLP	0.9744	3
SVM	0.9843	2
2NN	0.9397	7
NB	**0.9925**	**1**
LR	0.9451	5
RF	0.9424	6
C4.5 (N)	0.9159	8
RIPPER	0.8847	9
C4.5 (D)	0.8606	10
RBF	0.7089	11

Table 4. AUC by Learner

	AUC			
Learner	$\lambda = 4$	$\lambda = 6$	$\lambda = 8$	$\lambda = 12$
2NN	10.47	8.32	3.09	1.56
5NN	10.94	9.18	3.01	0.95
C4.5 (D)	0.84	8.52	10.29	0.77
C4.5 (N)	3.99	8.49	11.24	8.56
LR	-1.96	3.02	0.26	1.14
MLP	0.72	1.85	1.74	-0.48
NB	**0.12**	**0.05**	**0.00**	**0.02**
RBF	11.69	28.46	11.69	19.13
RF	12.15	9.44	3.77	1.41
RIPPER	6.93	4.24	4.09	-1.37
SVM	1.63	3.62	0.32	0.89

Table 5. (%) change in AUC from $\mathcal{C}^o$ to $\mathcal{C}^{10p}$

(not shown due to space limits) is also the lowest among all learners. Some learners had negative rates of change in their AUCs because their values increased at higher noise levels. In other words with more noise in the training dataset, the learner performed slighly better. For example, with $\lambda = 4$, LR had a slightly higher AUC when constructed using dataset C^{10p} compared to dataset C^o. However for the few cases (LR with $\lambda = 4$ and MLP and RIPPER with $\lambda = 12$) that do perform better at higher levels of noise, the difference is close to zero and hence most likely a statistical anomaly. It is clear that in general, regardless of the level of imbalance, the learners achieve lower AUCs when the training dataset contains more noise. RBF suffers the most when the level of noise is increased, with the AUC decreasing by 11.69% or more regardless of λ. Both 2NN and 5NN are more impacted by noise with less severe imbalance ($\lambda = 4$ or 6), while C4.5 (D) and C4.5 (N) suffer the largest decrease in AUC with $\lambda = 8$. The SVM and MLP learners are also relatively robust to noise across different levels of imbalance.

Figures 1 and 2 present the rankings of the learners by different levels of imbalance (for a fixed dataset C^{10p}) and noise (for a fixed level of imbalance $\lambda = 12$), respectively. In Figure 1, the mean AUC was calculated for each learner separately for the four datasets C_4^{10p}, C_6^{10p}, C_8^{10p}, and C_{12}^{10p}

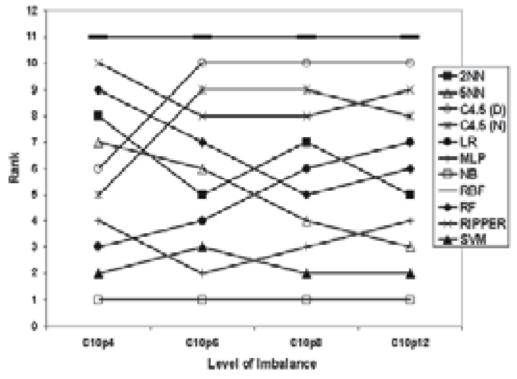


Figure 1. Performance Ranking by Imbalance Level

(for each of these datasets, each learner was run using 10-fold cross validation 30 times, so each learner has 30 AUC values for a given dataset). The learners were then ranked from one to 11 based on the mean AUC for the given dataset, with the learner obtaining the highest AUC given a rank of one, and the learner with the lowest AUC given a rank of 11. The trend of the learner rankings for a fixed level of noise (C^{10p}) but different levels of imbalance is given in Figure 1. Figure 2, on the other hand, fixes the level of imbalance ($\lambda = 12$) but varies the dataset (and hence the level of noise) from C^o to C^{5p} to C^{10p}.

From Figure 1, NB is the top performer using the datasets with the highest level of class noise ($\mathcal{C}^{10p}$) regardless of the level of class imbalance, demonstrating unmatched robustness when dealing with both class imbalance and class noise. SVM is the second most robust learner, only dropping its ranking one place at $\mathcal{C}_6^{10p}$. 5NN has the largest ranking improvement from seventh to third as the level of class imbalance became progressively more severe with dataset $\mathcal{C}^{10p}$. LR, C4.5(D), and C4.5(N) each had a loss of ranking equal to four positions as the level of imbalance increased. RBF performed the worst at all levels of imbalance for dataset $\mathcal{C}^{10p}$.

Figure 2 shows the effect of class noise on the rankings for each learner with a level of class imbalance held constant at $\lambda = 12$. The two best learners across all three class noise levels are SVM and NB. At $\mathcal{C}^{10p}$, NB switched ranking positions with SVM and became the best learner. This indicates that NB is more robust to the highest levels of class imbalance and class noise. MLP has the largest improvement in rank (four rank positions) between $\mathcal{C}_{12}^o$ and $\mathcal{C}_{12}^{5p}$ datasets with a slight decrease at $\mathcal{C}_{12}^{10p}$. RBF performed the worst at all levels of noise, followed by C4.5 (D) and RIPPER.

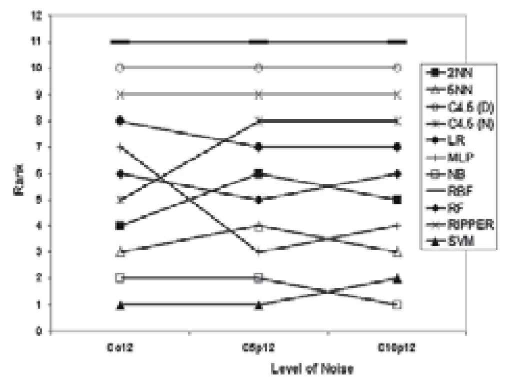


Figure 2. Performance Ranking by Noise Level

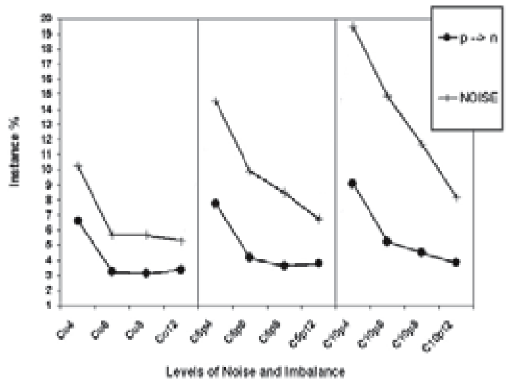


Figure 3. Relationship Between Noise and Imbalance

4.2 Noise Reduction as Imbalance Worsens

Figure 3 shows three separate plots of the percent of overall noisy (NOISE) instances and $p \rightarrow n$-type noisy instances in each of the 12 derived datasets from $\mathcal{C}^o$, $\mathcal{C}^{5p}$, and $\mathcal{C}^{10p}$. $p \rightarrow n$-type noisy examples are important because they indicate instances labeled as negative (from majority class nfp) but are actually positive (belong to minority class fp). The figure shows the change in NOISE and $p \rightarrow n$-type noise on the y-axis, as the level of class imbalance increases, shown on the x-axis.

In all four datasets derived from $\mathcal{C}^o$ located at the left side of Figure 3, the levels of NOISE and $p \rightarrow n$-type noise drop between imbalance levels $\lambda \in \{4, 6\}$ down to about 50% of the highest NOISE level at $\mathcal{C}_4^o$ and about 53% of the highest $p \rightarrow n$-type noise level at $\mathcal{C}_4^o$. For the rest of the class imbalance levels, the levels of NOISE and $p \rightarrow n$-type noise changed little. For the datasets derived from $\mathcal{C}^{5p}$ at the center of Figure 3, the levels of NOISE drop continuously as the level of class imbalance increases. The net reduction in the level of NOISE was about 55% from the highest level at $\mathcal{C}_4^{5p}$. The levels of $p \rightarrow n$-type noise between $\lambda = 4$ to $\lambda = 6$ decreased by about 50% from the highest level at $\mathcal{C}_4^{5p}$, elsewhere the noise distribution remained stable.

For the datasets derived from $\mathcal{C}^{10p}$ at the right side of Figure 3, NOISE and $p \rightarrow n$-type noise both dropped continuously as the class imbalance increased from $\lambda = 4$ to $\lambda = 12$. In summary, all twelve derived datasets experienced a reduction in noise as the class imbalance worsened. In other words, the datasets became progressively cleaner but more imbalanced as the threshold λ increased from four to 12. This effect is attributed to the noise injection process used in this study.

Figure 4 shows the changes in mean AUC values calculated from all the learners at each level of class noise for all levels of class imbalance. The AUC from $\mathcal{C}^{10p}$ has the lowest values, while the AUC from $\mathcal{C}^o$ has the highest values.

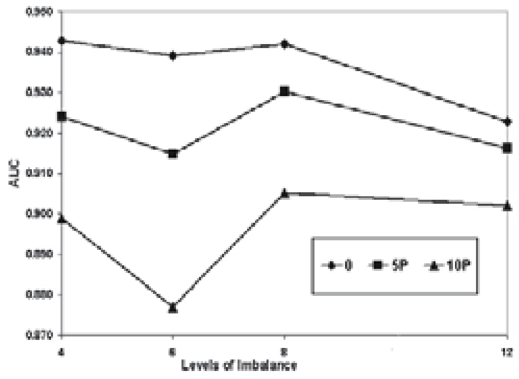


Figure 4. AUC **by Level of Imbalance**

With datasets $\mathcal{C}^{5p}$ and $\mathcal{C}^{10p}$, the learners are affected by the class imbalance at $\lambda = 6$, but at $\lambda = 8$, the performance is improved compared to $\lambda \in \{4, 6\}$. The effect of class imbalance on the AUC of all learners was not as significant as originally expected. As the level of imbalance became more severe, the AUC remained relatively stable. This effect can be attributed to the reduction in the amount of noise in our data that naturally occurs as we increase the value of λ. In other words, for our study, imbalance and class noise were inversely correlated. With more severe class imbalance, the level of noise decreased, and therefore the effects of these factors to some degree offset one another.

4.3 Threats to Validity

Experimental work in the domain of empirical software engineering often includes two common types of threats to validity [15]: threats to *internal* validity and *external* validity. Threats to internal validity are unaccounted influences that can impact the empirical results. Learners were constructed using a publicly available, high quality, and commonly-used data mining tool called WEKA [14]. All results were verified for accuracy by our group and the software engineering domain expert.

External validity considers the generalization of the results outside the experimental setting, and what limits, if any, need to be applied. A significant amount of experimentation was performed in this study, with 39,600 learners constructed from 12 datasets derived from the CCCS dataset. The CCCS dataset has been carefully analyzed and used in many studies by our group and by our software engineering domain expert. The emphasis of this study is centered towards the software engineering domain. Future work could use datasets from other software projects. This study is unique in that we have constructed many carefully controlled experiments under the supervision and guidance of a domain expert with a deep understanding of the data.

5 Conclusions

This study highlighted the significant relationship between classification algorithms, class imbalance, and class noise in software quality data. A summary of the conclusions discussed in Section 4 is presented next:

1. The performance of the learners at different levels of class noise was significantly affected when the noise levels increased. The impact depends heavily on the learner used: some learners, such as NB are relatively unaffected by class noise, while other learners such as RIPPER or RBF perform worse when the data is noisy. When using data of unknown quality for classification purposes, it is critical to consider the base learner.

2. Naive Bayes was the most robust learner with respect to class noise and class imbalance. This is clearly demonstrated in the AUC values for each learner in Table 4. In addition, from Table 5, NB has a much lower change in AUC than any other learner as the noise level increases, demonstrating its unmatched robustness to class noise in our experiments.

3. The effects of class imbalance on the mean AUC at each level of class noise was not as significant as originally anticipated. This observation is illustrated in Figure 4. The largest change in AUC between any two levels of class imbalance occurred at imbalance levels of $\lambda \in \{6, 8\}$ for the $\mathcal{C}^{10p}$ datasets. As mentioned, this is due to the inverse correlation between imbalance and class noise. At higher levels of imbalance, the dataset is cleanest relative to the class, and hence the effect of imbalanced classes is offset by a relatively cleaner class label.

Future work will consider extending the scope of our research by exploring the cross-effects of dependent/independent variables with class imbalance, noise, and missing values. In addition, as with any empirical work, more carefully designed experiments should be performed to verify the conclusions obtained in this work.

References

[1] D. W. Aha. *Lazy learning*. Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[2] R. Barandela, R. M. Valdovinos, J. S. Sanchez, and F. J. Ferri. The imbalanced training sample problem: Under or over sampling? *In Joint IAPR International Workshops on Structural, Syntactic, and Statistical Pattern Recognition (SSPR/SPR'04), Lecture Notes in Computer Science 3138*, (806-814), 2004.

[3] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[4] N. V. Chawla, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer. Smote: Synthetic minority oversampling technique. *Journal of Artificial Intelligence Research*, (16):321–357, 2002.

[5] C. Drummond and R. C. Holte. C4.5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. In *Workshop on Learning from Imbalanced Data Sets II, International Conference on Machine Learning*, 2003.

[6] N. Japkowicz and S. Stephan. The class imbalance problem: a systematic study. *Intelligent Data Analysis*, 6(5):429–450, 2002.

[7] T. M. Khoshgoftaar and E. B. Allen. Classification of fault-prone software modules: Prior probabilities, costs and model evaluation. *Empirical Software Engineering*, 3:275–298, 1998.

[8] T. M. Khoshgoftaar, E. B. Allen, and J. Deng. Using regression trees to classify fault-prone software modules. *IEEE Trans. Reliability*, 51(4):455–462, 2002.

[9] T. M. Khoshgoftaar, J. Van Hulse, and C. Seiffert. A hybrid approach to cleansing software measurement data. In *Proceedings of the* 18^{th} *IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2006)*, pages 713–722, Washington, D.C., November 13-15 2006.

[10] A. Napolitano. Alleviating class imbalance using data sampling: Examining the effects on classification algorithms. *Master's Thesis, Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL USA*, December 2006. Advised by Taghi M. Khoshgoftaar.

[11] F. Provost and T. Fawcett. Robust classification for imprecise environments. *Machine Learning*, 42:203–231, 2001.

[12] J. Van Hulse and T. M. Khoshgoftaar. Class noise detection using frequent itemsets. *Intelligent Data Analysis: An International Journal*, 10(6):487–507, 2006.

[13] G. M. Weiss and F. Provost. Learning when training data are costly: the effect of class distribution on tree induction. *Journal of Artificial Intelligence Research*, 19:315–354, 2003.

[14] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, California, 2nd edition, 2005.

[15] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering: An Introduction*. Kluwer International Series in Software Engineering. Kluwer Academic Publishers, Boston, MA, 2000.

[16] X. Zhu and X. Wu. Class noise vs attribute noise: A quantitative study of their impacts. *Artificial Intelligent Review*, 22(3-4):177–210, November 2004.

[17] X. Zhu and X. Wu. Cost-guided class noise handling for effective cost-sensitive learning. In *4th IEEE International Conference on Data Mining (ICDM 2004)*, pages 297–304, November 2004.

Architectural Elements Recovery and Quality Evaluation to Assist in Reference Architectures Specification

Aline Pires Vieira de Vasconcelos[1,2] Cláudia Maria Lima Werner[1]
[1]COPPE/UFRJ/ Systems Engineering and Computer Science Program
P.O. Box 68511 – ZIP 21945-970 - Rio de Janeiro – RJ – Brazil
[2]CEFET Campos (Federal Center for Technological Education of Campos)
Dr. Siqueira, 273 – ZIP 28030-130 - Campos dos Goytacazes - RJ - Brazil
(aline, werner)@cos.ufrj.br

Abstract

Reference architectures are the basis for application instantiation in both Domain Engineering and Product Line contexts. They are created based on domain requirements, commonalities, and variability. Considering that one of the essential sources of information in this context is the existing systems available in the domain, reverse engineering becomes a key activity to assist in reference architectures specification. Although many approaches consider reengineering existing systems to help in reusable components extraction and variability identification, they do not propose an approach, with well-defined activities and criteria, to support reference architectures specification from existing systems. In this context, this paper describes an approach to help in this task covering three macro-activities, namely: reverse engineering focusing on architectural elements recovery; recovered architecture quality evaluation; and domain commonalities and variability identification. These activities are guided by general criteria, which can be reused across domains, and are supported by a tool set integrated to a reuse based software development environment, i.e. the Odyssey environment. The results of experimental studies conducted to evaluate the first two macro-activities are described in this paper.

1. Introduction

Domain reference architectures are architectures for application families (i.e. domains) that must be in conformance with the functional and non-functional requirements of the domain [1]. Domain reference architectures (or DSSAs – Domain Specific Software Architectures) were first introduced in the context of Domain Engineering (DE) approaches [2], but are also the basis for product instantiation in the context of Product Line (PL) approaches [3], also being called Product Line Architectures (PLA). Reference architectures are built upon domain requirements, commonalities, and variability.

According to Kang [2], existing systems are one of the most meaningful domain information sources. Although many DE and PL approaches do emphasize the importance of analyzing existing systems during domain modeling and design, they do not provide a method, with well-defined activities, criteria and tool support to guide this analysis.

In this context, this paper presents a three-phase process, encompassing three macro-activities, to assist in domain reference architectures specification from existing systems, namely: reverse engineering and architectural elements recovery, performed through the proposed ArchMine approach; recovered architecture quality evaluation, performed through an extended version of an existent approach, i.e. ArqCheck [10]; and domain commonalities and variability identification, performed through the proposed ArchToDSSA approach. It is important to emphasize, however, that for specifying domain reference architectures, the architecture of some existing systems in the same domain must be recovered and compared. According to [4], a common number is 3-4 systems. Besides assisting in reference architectures specification, the proposed approach supports program comprehension by recovering updated documentation for existing systems, which, in general, is outdated. The proposed activities and criteria are general, and can be applied for reference architectures specification in different domains.

In order to represent domain commonalities and variability, we follow the Odyssey environment [5] notation, Odyssey-FEX, in which a domain design element can be: mandatory, i.e. common to all domain applications, or optional; and invariant, i.e. it can not be adapted, or variation point, i.e. it can be adapted through the selection of variants. These classifications are orthogonal, and Odyssey-FEX encompasses their representation in all domain models, e.g. features, use cases, classes. In this work, domain variability is represented in a UML class model, but this representation is mapped to other domain models according to the mapping heuristics of Odyssey.

The paper is organized as follows: Section 2 presents the approach to assist in reference architectures specification; Section 3 describes the results of experimental studies that evaluated the first two macro-activities; Section 4 presents meaningful related work; and Section 5 outlines some conclusions.

2. Proposed approach to assist in reference architectures specification

The following sub-sections describe the three macro-activities composing our approach, together with their supporting techniques, criteria, and tool set. An example concerning a school domain is provided along the explanations. Although the approach is application independent, its supporting tool set was designed to analyze Java applications. As mentioned before, it is integrated to Odyssey [5], a reuse based software development environment.

2.1 Reverse engineering and architectural elements recovery

Architectural elements recovery is performed through our reverse engineering approach, i.e. ArchMine, that is mainly based on dynamic analysis and data mining techniques. Its focus is on clustering source code classes into architectural elements based on the functionality they support. These architectural elements indicate domain concepts or reusable component candidates. Reusable components are self-contained artifacts that perform specific functions and have clear defined interfaces [6]. The recovered reusable component candidates must be further evaluated and reengineered to generate components.

In order to detect classes that must be clustered, dynamic analysis generate execution traces (i.e. sequences of method calls that implement a use case scenario or system functionality) that are mined for the discovery of functionally related classes. We apply a data mining algorithm based on Apriori [7] to mine the generated execution traces. Figure 1 depicts ArchMine architecture recovery process. All the described processes are represented following OMG SPEM (Software Process Engineering Metamodel) notation.

In order to perform dynamic analysis, use case scenarios to guide application execution must be defined. This activity is performed in parallel with *Static Structure Extraction*, which extracts a low-abstraction level class model from Java source code.

A use case is a description of sequences of actions, including variants, that an entity (e.g. a system) performs to produce an observable result of value to an actor [8]. It represents a functional requirement of a system. Each sequence of actions in a use case is called a use case scenario and represents one means for obtaining that functional requirement [8]. For example, in a school domain, a use case could be "Inform Students Grade", and two use case scenarios "Inform Graduate Students Grade" and "Inform Undergraduate Students Grade". ArchMine proposes a set of heuristics to derive use case scenarios, such as: derive one use case scenario for each main menu and popup menu option; consider the last level of the hierarchy for nested menus; derive one scenario for each tool bar button; options semantically equivalent may derive only one scenario; login must derive a scenario; tabs in tabbed panes and buttons in panels may derive scenarios if they correspond to a distinct functionality in respect to the functionality of their container panes.

Dynamic Analysis is performed with the support of the Tracer tool [9], that uses AspectJ technology to trace application execution and generates XML execution traces related to use case scenarios. The greater the coverage achieved in use case scenarios monitoring, the higher is the amount and quality of architectural elements recovered. Coverage can be inferred by comparing classes in the static and dynamic models, and by interviewing a system stakeholder (i.e. programmer, designer, developer).

Architectural Elements Reconstruction involves mining the gathered execution traces, by applying an Apriori-like algorithm, and discovering related classes based on the functionality they support. Classes that appear together in a "X" number of execution traces, i.e. classes that together support a set of related system functionalities, are indicated to compose an architectural element. This "X" number is the minimum confidence value that must be provided to the algorithm, which is based on the following concepts:

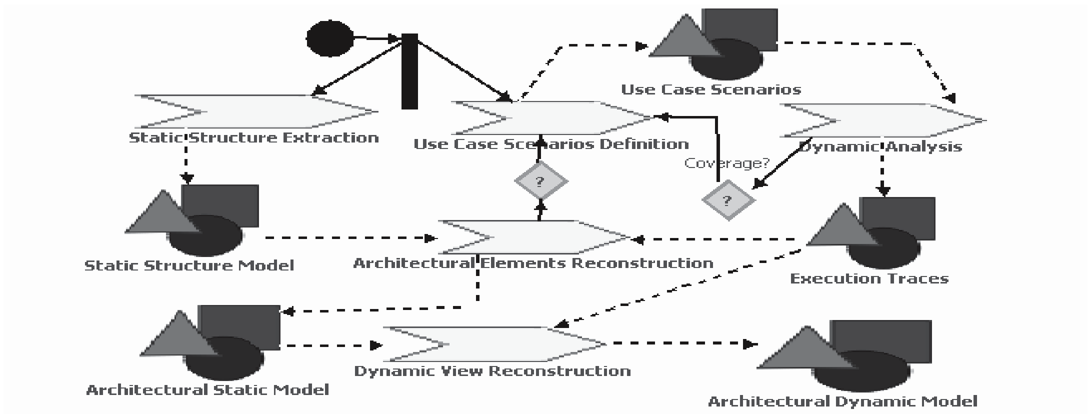

Use Case Scenarios
Static Structure Extraction
Use Case Scenarios Definition
Dynamic Analysis
Coverage?
?
?
Static Structure Model
Architectural Elements Reconstruction
Execution Traces
Architectural Static Model
Dynamic View Reconstruction
Architectural Dynamic Model

Figure 1. Reverse engineering and architectural elements recovery activities.

- *Association rule:* an implication of the form $X \Rightarrow Y$, where X and Y are items of the database and $X \cap Y = \emptyset$. X is the *antecedent* of the rule, while Y is its *consequent*.
- Apriori requires two threshold values: *minimum support* and *minimum confidence*. *Support* "s" means that s% of the transactions in the database contains X and Y. *Confidence* "c" implies that c% of the transactions that contain X also contain Y. Given a set of transactions τ, the problem of mining association rules is to generate all rules that have support and confidence equals or greater than the user specified *minimum support* and *minimum confidence*.

In order to mine association rules, some concepts from the database domain are mapped to the dynamic analysis context (Table 1). Our Apriori-like algorithm incrementally queries specific antecedents discovering classes that must be clustered.

Mining is semi-automated and supported by the TraceMining tool, an Odyssey plug-in, which can randomly suggest mining *antecedents* from higher to lower support values. Therefore, architectural elements generation starts from more general to more specific architectural elements. *Minimum support* value must be 0% since all the monitored classes must be clustered into architectural elements, even if it appears in only one use case scenario. Minimum confidence, on the other hand, must be tuned along the mining process with a system stakeholder, although 60% proved to be a good value along the experimental studies.

Table 2 presents an example set of execution traces for a school domain and Table 3 presents the cycles followed in the incremental mining approach, based on the explained mining principles, with their results.

Table 1: Mappings of database concepts to dynamic analysis

Data Mining Concepts	Mapping to Dynamic Analysis
Transaction	A use-case scenario implemented by an execution trace.
Data Item	A class supporting a use case scenario.
Support	Percentage of use case scenarios in which the classes in an association rule appear together.
Minimum Support	The minimum percentage of use case scenarios in which the classes in an association rule must appear together.
Confidence	Percentage of use case scenarios of class X in which a class Y also appears.
Minimum Confidence	The minimum percentage of use case scenarios of class X in which the class Y must also appear for them to be included in an association rule.
Antecedent	The class that is used as input to discover the association rules.
Consequent	The classes that are associated to the antecedent with support and confidence greater or equal to the minimum percentages.

As presented in Table 3, classes are mined from higher to lower support values, i.e. starting from class Student which has 100% of support, and finishing with class PrinterUtils that has 33,3% of support. Already grouped classes are filtered from subsequent mining cycles. The minimum confidence used was 60%, therefore GraduateStudent and UnderGraduateStudent were suggested for clustering with Student. At the end, three architectural elements were recovered, namely: Student (classes: Student, GraduateStudent, and UndergraduateStudent), ClassesSubscription (classes: Classes and Subscription), and Printer (classes: PrinterUtils and PrinterConfig). Architectural element names are derived based on the most common substrings in their class names. Class Grade was not covered by any element and can be manually clustered.

The reconstructed architectural elements are exported to the Odyssey environment, and represented through UML packages. Relationships among them are derived based on the relationships among their constituent classes. Whenever there is a relationship between classes of distinct architectural elements, a dependency is derived between them. Intermediate results can be discussed with a system stakeholder, leading, in some cases, to the need of re-defining current or identifying new use case scenarios.

Table 2: Execution traces for a school domain

Use Case scenario	Supporting Classes
1. Inform Graduate Students Grade	Student, GraduateStudent, Grade
2. Inform Undergraduate Students Grade	Student, UndergraduateStudent, Grade
3. Print Students Grade	Student, GraduateStudent, UndergraduateStudent, PrinterUtils, PrinterConfig, Grade
4. Subscribe Graduate Students	Student, GraduateStudent, Classes, Subscription
5. Subscribe Undergraduate Students	Student, UndergraduateStudent, Classes, Subscription
6. Print Students Subscriptions	Student, GraduateStudent, UndergraduateStudent, Classes, Subscription, PrinterUtils, PrinterConfig

Table 3: Mining results

Cycle	Antecedent	Support	Association Rules/Confidence
1	Student	100%	Student $\Rightarrow$ GraduateStudent, UndergraduateStudent – 66,7%
2	Classes	50%	Classes $\Rightarrow$ Subscription – 100%
3	PrinterUtils	33,3%	PrinterUtils $\Rightarrow$ PrinterConfig – 100%

Finally, *Dynamic View Reconstruction* involves generating UML sequence diagrams in the Odyssey environment, associated to the monitored use case scenarios and showing the system behavior in its architecture. Further details can be obtained in [9].

2.2 Architecture evaluation

Architecture evaluation is performed with an extended version of ArqCheck [10]. ArqCheck is an architecture evaluation method based on inspection which uses a checklist as the defect detection technique. ArqCheck identifies three kinds of architectural defects, namely: architectural representation inconsistencies, functional and non-functional requirements conformance.

Concerning non-functional requirements, ArqCheck originally identified conformance to Availablity, Performance, Modifiability, Usability, Security, and Testability. We extended ArqCheck to evaluate conformance to Reusability, since reusing recovered architectural elements in domain reference architectures specification is our main goal. Reusability questions in the checklist are mainly based on functional cohesion and low coupling among architectural elements.

ArqCheck follows a traditional inspection process [11] and adapt some activities, e.g. *Inspection Planning* which also covers the configuration of the checklist to the architectural representation and non-functional requirements of interest. After evaluation by system experts, who apply ArqCheck, recovered architectural elements are corrected for domain reference architectures specification.

2.3 Domain variability identification

Domain commonalities and variability identification encompasses three activities (Figure 2), being supported by the ArchToDSSA tool, an Odyssey plug-in. *Optionality Detection* takes as input the recovered and corrected architectural models, and identifies mandatory and optional architectural elements. ArchToDSSA tool compares the architectural elements through their names and creates a match whenever there is a correspondence in at least two architectures. The comparison is performed in both levels: at the architectural level and at the detailed design level, in which internal classes of architectural elements are also compared and indicated as optional or mandatory.

Whenever an architectural element is found in all the architectures, it is indicated as a mandatory candidate element. On the other hand, if it is found in only some of the compared architectures, it is identified as an optional candidate element.

In order to provide semantics to this comparison, ArchToDSSA tool encompasses a domain dictionary in which synonyms are stored. The goal of this domain dictionary is to allow identifying semantic equivalent architectural elements that have different names. For example, Teacher and Professor should be identified as the same element in a hypothetical school domain.

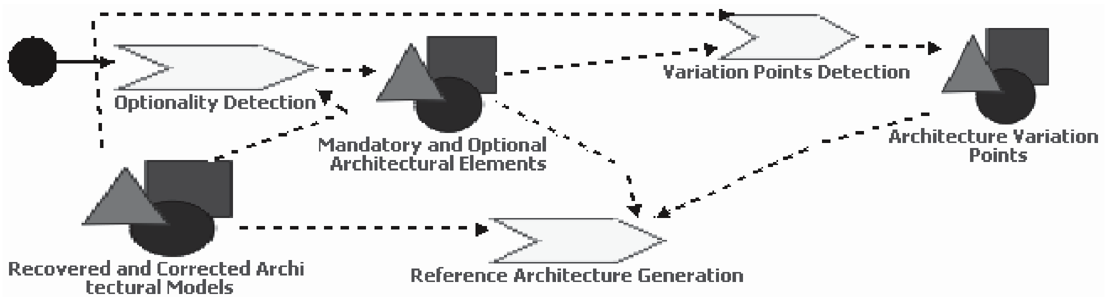

Optionality Detection
Mandatory and Optional Architectural Elements
Variation Points Detection
Architecture Variation Points
Recovered and Corrected Archi tectural Models
Reference Architecture Generation

Figure 2. Domain commonalities and variability identification activities.

Besides providing a synonym dictionary, ArchToDSSA tool allows the user to inform some filters, eliminating strings that do not help in the comparison, such as: manager, mgr, controller, utils etc., which don't have meaningful semantics. Finally, comparisons can be performed at the substring level, i.e. architectural element names can be broken into substrings and if the architectural elements have the same amount of substring in common in their names, the tool indicates that they compose a matching. All options can be configured by the user.

Variation Points Detection involves identifying inheritance and interfaces in the architectural models. These constructions are indicated as variation point representations in the Odyssey-FEX notation [5]. Variation points can be mandatory or optional, according to the optionality previously identified.

Finally, in the *Reference Architecture Generation* activity, the user must choose one of the compared architectures to serve as the basis to generate a domain reference architecture. This choice is not currently supported by the ArchToDSSA tool. The selected architecture is exported to Odyssey, to be reused in domain applications instantiation. Before exportation, the user can select optional elements from the other architectures to compose the selected reference architecture. However, the tool doesn't export the relationships among these new optional elements and the ones in the selected architecture, once the structures in which they were originally developed may diverge. These optional elements are marked as "not completely defined", and must be further specified by the user.

3. Evaluation studies

In order to evaluate ArchMine approach, two case studies involving applications of different sizes and domains were conducted. The approach was refined along the case studies, and its final version was applied in a third case study, in which the main goal was to evaluate the feasibility of evaluating and correcting the recovered architectures by applying an extended version of ArqCheck.

In the first study, one system expert agreed that ArchMine could correctly recover the whole application architecture, and for the other two system experts it could only recover some architectural elements. In the second case study, ArchMine performance had improved. Performance was evaluated by comparing precision, i.e. correctness of the recovered architectural elements, and recall, i.e. coverage of the recovered architectural elements, against the values obtained in the first study and against values available in the literature [12]. Precision and recall were computed with the system expert who evaluated the recovered architectural elements by comparing it to the original ones. We also compared precision and recall with the values obtained by applying another architecture recovery approach [13], obtaining better results.

In these studies, architectural evaluation was performed in an ad-hoc fashion and required a huge human effort. Therefore, extended ArqCheck was incorporated to architecture evaluation. In the final evaluation study, the results showed that ArqCheck could reduce the recovered architectures evaluation effort and improve architecture quality for reuse.

4. Related work

In [3], Gomaa emphasizes the importance of making the consistency among existing system models in order to determine the kernel, optional, and alternative PL elements. Although some guidelines are provided, legacy models recovery is not encompassed by the approach and important issues, like name divergences among existing systems, are not considered.

In [14], PL variabilities are refactored from existing products and represented through aspects. The main focus of the work is on variability refactoring and representation, not on variability detection. Stoermer and In [4] the MAP (Mining Architectures for Product Lines) method for migrating individual products to a PL is proposed. Although MAP proposes a more global process, including, for example, a preparation phase in which product candidates are selected, there are no well-established criteria for architectural elements recovery and comparison that can be applied to different application domains.

In [15], PL variation points are identified by comparing execution traces gathered from different product versions. In [16], application reusable domain components are extracted by applying metrics that indicate components quality attributes adequacy. Quality attributes that indicate Reusability are functional utility, correctness, adaptation cost etc. In our work, Reusability is evaluated through a systematic method (i.e. ArqCheck), which evaluates architectural elements functional cohesion and coupling.

5. Conclusions

In this paper we presented an approach to assist in domain reference architectures specification, by performing three macro-activities, i.e. architectural elements recovery, architecture evaluation, and domain variability identification. Its main contributions are a set of systematic activities, with general supporting criteria and tool set, that can guide existing systems evaluation during a DE or PL process. Moreover, it can assist in program comprehension by recovering updated documentation for existing systems.

However, some limitations are also outlined, such as the impact of the selected use case scenarios, minimum confidence and mining antecedents in the recovered architecture quality. Moreover, it is necessary to identify commonalities and variability also in the other models, i.e. use case and dynamic models. As future work, experimental studies to evaluate the feasibility of the whole process, including domain commonalities and variability identification, will be performed.

6. References

[1] M. Shaw and D. Garlan, *Software Architecture: Perspectives on an Emerging Discipline,* New Jersey: Prentice-Hall., 1996.

[2] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, *Feature-Oriented Domain Analysis (FODA): Feasibility Study*, Software Engineering Institute, Pittsburgh CMU/SEI-90-TR-21, 1990.

[3] H. Gomaa, *Designing Software Product Lines with UML: from Use Cases to Pattern-Based Software Architectures*, Addison-Wesley Professional, 2004.

[4] C. Stoermer and L. O'Brien, "MAP: Mining Architectures for Product Line Evaluations," *In: 3rd Working IFIP Conference on Software Architecture*, Amsterdam, Holland, August, 2001, pp. 35-44.

[5] Odyssey, "Reuse Infrastructure based on Domain Models", In: http://reuse.cos.ufrj.br/odyssey.

[6] J. Sametinger, *Software Engineering with Reusable Components,* Springer-Verlag, New York, Inc., 1997.

[7] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," In: *20th VLDB*, Santiago, Chile, September, 1994, pp. 487-499.

[8] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, 1st ed: Addison-Wesley, 1998.

[9] A. Vasconcelos, R. Cepêda, and C. Werner, "An Approach to Program Comprehension through Reverse Engineering of Complementary Software Views," In: *1st International Workshop on Program Comprehension through Dynamic Analysis*, Pittsburgh, PA, USA, November, 2005, pp. 58-62.

[10] R. Barcelos and G. Travassos, "Evaluation Approaches for Software Architectural Documents: A Systematic Review," In: *9º Workshop Iberoamericano de Ingenieria de Requisitos y Ambientes de Software*, La Plata, Argentina, 2006, pp. 433-446.

[11] M. Fagan, *Design and Code Inspection to Reduce Errors in Program Development*, IBM Systems Journal, vol. 15, pp. 182-211, 1976.

[12] K. Sartipi, *Software Architecture Recovery based on Pattern Matching*, PhD Thesis, *School of Computer Science*: University of Waterloo, 2003.

[13] B. Mitchell and S. Mancoridis, *On the Automatic Modularization of Software Systems Using the Bunch Tool*, IEEE Transactions on Software Engineering, vol. 32, pp. 193-208, 2006.

[14] V. Alves, P. Matos, L. Cole, et al., "Extracting and Evolving Mobile Games Product Lines," In: *9th International Software Product Line Conference*, Rennes, France, September, 2005, pp. 70-81.

[15] B. Cornelissen, B. Graaf, and L. Moonen, "Identification of Variation Points Using Dynamic Analysis," In: *1st International Workshop on Reengineering Towards Product Lines (R2PL)*, Pittsburgh, PA, USA, November, 2005, pp. 9-13.

[16] D. Ganesan and J. Knodel, "Identifying Domain-Specific Reusable Components from Existing OO Systems to Support Product Line Migration," In: *1st R2PL*, Pittsburgh, PA, USA, November, 2005, pp. 16-20.

EvoSpaces: 3D Visualization of Software Architecture

Sazzadul Alam, Philippe Dugerdil
HEG - Univ. of Applied Sciences, Geneva, Switzerland
Sazzadul.Alam@hesge.ch, Philippe.Dugerdil@hesge.ch

Abstract

This paper presents the Evospaces reverse-engineering tool that represents the architecture and metrics of complex software systems in a 3D virtual world. The main goal of our project is to exploit familiar metaphors (analogical representations borrowed from another domain) and sophisticated interactions modes to help the user understand complex systems. First, we present the general architecture of the Evospaces tool. Second, we show the metaphors we have implemented to help the user to quickly get an overview of a complex system. Then we present the interaction mode we designed to let the user explore such a complex system. Finally, we show an example of a virtual space we have designed to represent the architecture of Mozilla. We conclude with an account of our current research directions. The main contributions of this paper are the metaphors that we have used and the types of interaction modes we have implemented. They represent a real contribution to the set of tools that can help a maintenance engineer to understand a large system such as Mozilla.

1. Introduction

Since software is a formal and abstract construct, there is no "natural" representation for it. On the other hand a visual representation is an appealing way to represent lots of information simultaneously. Today's industrial software systems are tremendously complex, with size counting in millions of lines of code. One way to cope with complexity is to represent information hierarchically in several levels of abstraction [17]. Fortunately, software systems are often structured hierarchically as systems, subsystems and components located in packages and/or directories. However containment information does not tell us much about the complexity of the contained elements. Then, metrics can be used to attach measures to the software elements. But metrics are most useful if one can compare their values among several components located in different parts of the system. Then, we must find ways to represent structural and metrics information on the system in the same visual space [10]. Moreover these visual representations should be easily interpretable by the user, to let him decide what components to investigate further. However, if the unsophisticated display of a few dozen of classes in a diagram can provide some insight to the structure and behavior of a system, we must find way to meaningfully represent hundreds of classes or components. This is a complexity one cannot avoid for industrial-size systems. Although it is clear that a good drawing can replace a thousand words, it all depends on the drawing. Then, the central research questions are, first: how can we exploit familiar visual metaphors (analogical representations borrowed from domain of which the user has direct sensible experience) to help the user grasp a myriad of information in a single view? Second, how can we provide the user with powerful navigation and interaction techniques to let him dig the system and discover information as needed while staying aware of the context? Third, how can we meaningfully represent relationship between software objects?

Our system basically rests on the representation of a 3D landscape in which the user can navigate and investigate software objects. It is implemented in Java and JOGL (OpenGL) [13] under Eclipse. As a test bench, we investigated the source code of Mozilla that contains about eleven thousand of files in its latest release. This paper is organized as follow. Section 2 presents the general architecture of our platform called "Evospaces". Section 3 presents the interaction mechanisms with the virtual landcape we have built. Section 4 presents an example where Mozilla is displayed in a virtual world. As a conclusion, section 5 presents an outlook of the future work.

2. Evospaces tool architecture

2.1 Introduction

To allow our tool to display systems written in different programming paradigms and to be able to quickly integrate new visualization metaphors we have built it in five layers (fig 1). Because of the well-defined interfaces between modules, changes made inside a given layer have a limited impact on the other layers. For example, the rendering engine will not be affected when changing the programming language of the program to analyze provided it follows the same paradigm (object oriented for example).

2.2 Source code layer

This layer represents the raw source code of the software under investigation, structured as files. Those files are parsed off-line to fill the database of code elements. Since we do not know at parsing time what information the users will look for, we have chosen to extract as much structural information as possible from the source code. In particular, a set of widely used metrics is computed on the target system while the database is loaded. This layer is also used when displaying the source code corresponding to some selected element in the views.

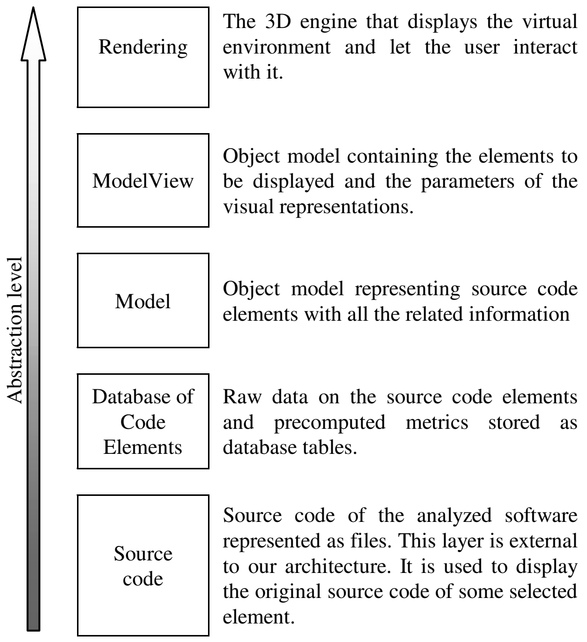

Abstraction level
Rendering
The 3D engine that displays the virtual environment and let the user interact with it.
ModelView
Object model containing the elements to be displayed and the parameters of the visual representations.
Model
Object model representing source code elements with all the related information
Database of Code Elements
Raw data on the source code elements and precomputed metrics stored as database tables.
Source code
Source code of the analyzed software represented as files. This layer is external to our architecture. It is used to display the original source code of some selected element.

Figure 1. The Evospaces' layered architecture

2.3 Database layer

At the database level, the source code elements are represented in the entity - relationship paradigm. Elements like classes, methods, variables, attributes, packages, files or modules are entities. The way those entities are structured (containment relations and programming-language level relations), communicate or work together is represented as relationship. The database contains one table per software entity and one table per "relationships" between software entities. Then, at this level, the software under investigation is modeled as a huge entity-relationship diagram like the one presented in figure 2. Moreover, all entities and relationships have extra properties like source level information (names, labels, parameters) and metrics values. These properties depend on the type of the entity or relationship considered. The classes in the *Database* layer implement a generic access to the tables. Basically they consists of "builders" [6], that instantiate the objects representing entities and relationships.

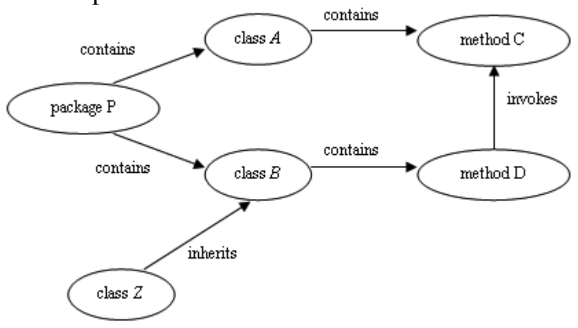


Figure 2. Source code data structure

2.3 Model layer

The *Model* layer implements the object representation of the loaded entities and relationships in the Evospaces system. Each kind of entity or relationship is represented by its own class. Consequently, this layer contains two hierarchies of classes, one for the entities and one for the relationships, following the Famix metamodel for object-oriented programming languages [1]. For example, in figure 3, we show the Famix model for entities.

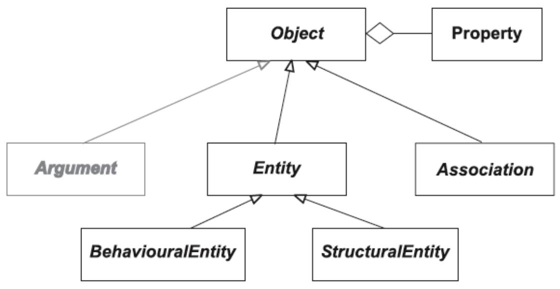


Figure 3. Famix model for entities

2.4 ModelView layer

This layer is the first to deal with visualization issues. It contains all the values and parameters used for the 3D rendering of the entities and relationships. Each object in the *ModelView* layer has a counterpart in the *Model* layer. Then, the *ModelView* layer contains two hierarchies of classes, one for the entities and one for the relationships that are similar to the hierarchies in the *Model* layer. However the entities and the relationships of the *ModelView* layer only contain visualizable data. The *ModelView* layer works as a visual abstraction of the raw data stored in the model layer: it maps the data of the model layer to displayable elements. In particular, this is where :

- The glyphs (graphical objects representing data through visual parameters [1]) are mapped to a given type of entity or relationships;
- The values of the metrics are mapped to some visual scale (saturation of colors for example) and positions in the 3D space (layout).

Moreover since we wish our system to let us experiment with different representations of the same set of entities and relationships, a given entity or relationship in the *Model* layer may me mapped to different visual objects in the *ModelView* layer. This makes it possible to maintain several concurrent views of the same set of software elements. However, at any given time, only one view will be displayed. This layer, which represents the largest part of our system, also contains the classes that control the interaction with the user. Any entity or relationship in this layer is associated with four objects: a shape, a color, a layout and a list of "reactions" to user actions. The "shape" defines how the element will look like in the 3D view (the glyph). It can be as simple as a fixed size geometric volume, like a cube or a cylinder, or be a much more sophisticated visual element, using transparency effects and textures. The dimensions of the glyphs are set proportionally to the value of one or more metrics.

Among the metaphors we experimented one of the most appealing is the modern city. Then, the classes and files are represented as buildings and the relationships as solid pipes between the buildings. On the other hands, metrics values intervals are mapped to different textures of buildings. For example, we split the files in three categories depending on their number of lines of code (LOC). Each category is represented by a different kind of building (different texture) (fig.4). In this example we used:

- a house for files with 0 to 50 LOC;
- an apartment block for files with 51 to 200 LOC;
- an office building for files with more than 200 LOC;

Morevoer we represented header files (.h) as a city hall with columns and a stickman for the functions and methods in classes and files.

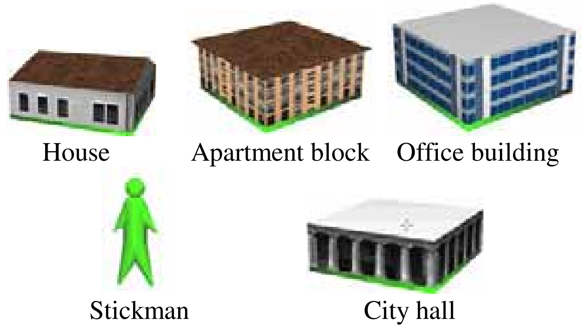


Figure 4. Examples of Glyphs

Then, another metrics can be mapped to the height of the building. We then set the number of floors to represent the number of global variables declared in the file (fig 5). We also split the files in three categories according to their number of global variables:

- small building for files with 0 or 1 variable;
- medium size building for files with 2 to 4 variables;
- tall building for files with 5 or more variables.

On the other hand, the height of the city hall depends on the number of functions in the header file. (small: 0 to 5 functions, medium: 6 to 15 functions, tall: more than 15 function).

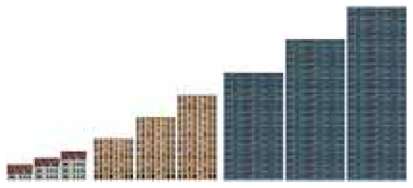


Figure 5. Mapping of metrics to the size

The objects representing the entities are distributed in the 3D space using a specific topology (layout). For example we could arrange the entities in rows and columns, in concentric circles, in spiral, etc. Since the layout of the objects also conveys information, we must find a good map of the chosen metrics to the layout so that an intuitive interpretation is possible. In figure 6 we present two layouts among those we investigated. In the concentric layout (left), a possible mapping could be: the older the class the closer to the center. In the chessboard layout (right), the mapping could be the static coupling between classes: the closer the classes the tightly coupled.

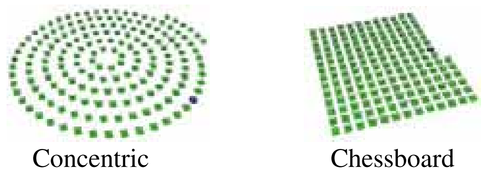


Figure 6. Examples of layouts

Once visualized, the user can interact with the displayed objects. Each visual element has a list of potential actions that the user may perform on it. For example, the user could request to display the value of some metrics, to change the visual appearance the object, to load the related elements, to open the corresponding source file, etc. The list of possible actions is defined for each type of elements and is accessible through a contextual menu.

2.5 Rendering layer

All classes responsible for the actual display of the views on the screen are located in the *Rendering* layer.

The 3D rendering library used is JOGL [7], a binding of OpenGL for Java, which has been released by Sun for Windows, Solaris, Linux and Mac OS platforms. The rendering engine, which is responsible for the drawing of the 3D scene on the screen, uses the data stored in the *ModelView* objects. This engine also catches the actions of the user and executes the corresponding operations. The *Rendering* layer also implements some Eclipse plug-in features, like a property page to specify the environment parameters for the Evospaces tool.

3. Interactions in the 3D view

3.1 Interaction with the visual objects

So far, our investigations on the interactions modes with the tool went along three directions. First, we studied the way to display the information retrieved from the database and pertaining to a given element. Second we investigated the ways to dynamically change the viewing parameters of the entities and relationships in order to find the best metaphors for the software elements in given situation. Third we investigated the navigation among the displayed software elements. As a result we implemented a context sensitive menu in the 3D space. Almost all actions available for an element are accessible trough its contextual menu**.** For example, all the available metrics stored in the database for a given element can be represented as a table that is displayed by selecting an item in the contextual menu. On the other hand, the relationships between the elements are manifold. The user can then select the relationship he wants to display by selecting it in a preference window. Then, the relationship will be displayed on demand by clicking the element in the 3D view. The relationships are represented as a solid pipe between the associated elements (fig.7).

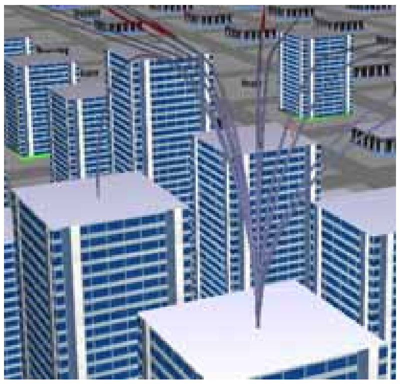


Figure 7: Relationships and directionality

A colored segment moving along the pipe from the origin to the destination of the relation represents the directionality of the relation. This gives the impression of an information flow between the connected elements. The red segments drawn on the gray pipe in figure 7 are the moving segments that represent directionality. For any display element on the screen, the user can ask the system to display the corresponding source code using its contextual menu.

3.2 Interaction modes

To set up and orient the camera in the 3D scene the user can use the buttons in a navigation panel (figure 8) or use their mouse and keyboard counterpart. Beyond the parameters of the camera, the navigation panel is used to select the relationship to display in the view and ask for the directionality of the relation to be "animated" (red segment moving).

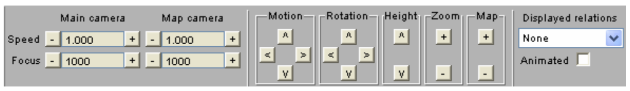

Main camera / Map camera
Speed 1.000 / 1.000
Focus 1000 / 1000
Displayed relations: None
Animated

Figure 8. The navigation panel

3.3 Zooming inside objects

Since objects representing files of classes can contain other objects (methods and variables), the user can zoom into the objects to display their contents. Then the methods and functions, which are drawn as "Stickmen" of different colors, represent the "workforce" inside the buildings. Each stickmen is surrounded by yellow boxes (its resources) representing the local variables used by the method. Like for the buildings, the user can ask the system to display the relationships associated to a given method (figure 9)

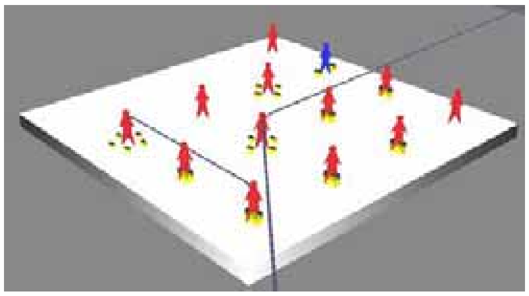


Figure 9. Methods, variables and relationships inside a buildings

If some relationship is displayed at the level of the methods, then it is also represented at the level of its containing building building when zooming out (Figure 10).

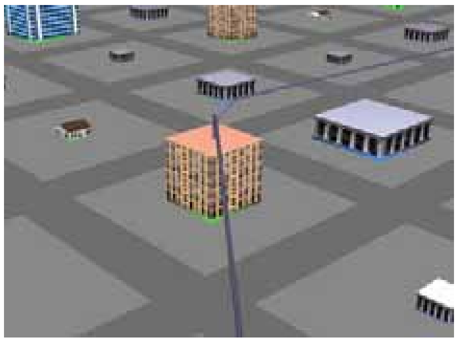


Figure 10. Relationship between buildings

3.4 Navigating the city with a road map

When travelling in a big and unknown city, it is easy to get lost. Better to have a map handy. In the Evospaces system, such a map can be displayed in one of the corners of the screen to show the user his current position in the city (Figure 11). Unlike real paper maps, the user can zoom into this road map to show it at different scales. This idea has been borrowed from the computer game technology where the user can display "radar views" to get a global awareness of the environment. In this figure we also display the name of the software elements as labels attached to the buildings.

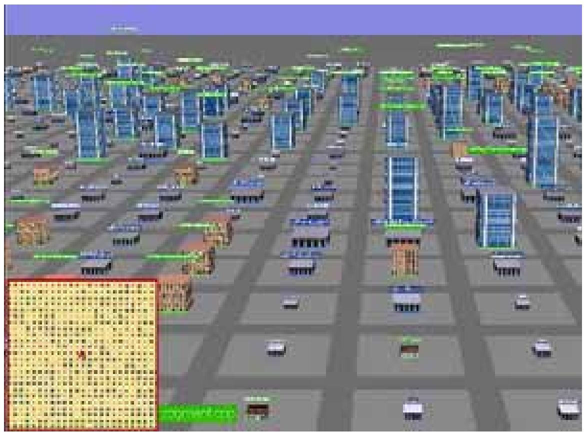


Figure 11: The file city and its road map.

5. Visualizing and navigating Mozilla's city

As an experiment, we displayed the source code of Mozilla, which is written in C/C++. The parsing and loading of the database has been done as part of another project [15]. Since all the versions of Mozilla are accessible, they have also been stored in the database. We then adapted the *Database* layer to access it. In its latest release, Mozilla contains more than thirty thousands methods, three thousands classes located in more than two thousands of files. In figure 12 we present the header and C++ files of Mozilla as a huge city. They are distributed on a grid layout according to the containment metrics (files in the same package are displayed close to each other).

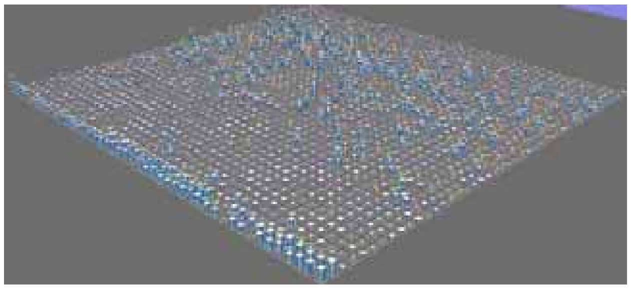


Figure 12. Mozilla as a huge city

In Figure 13 we show the contextual menu associated to an object. Through this menu we asked for the display of the values for all the metrics associated to the object. They are displayed as a table on the left of the view.

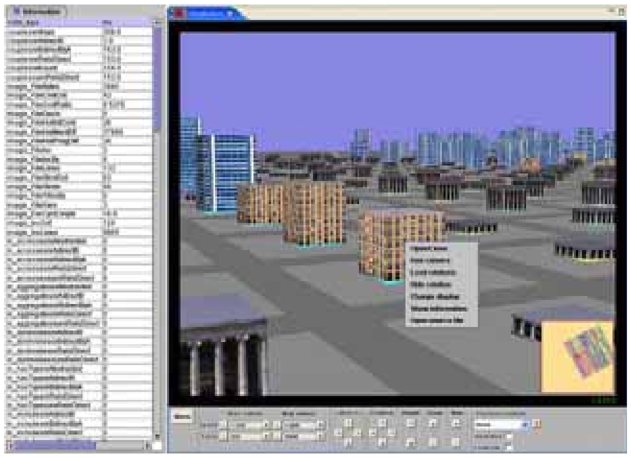


Figure 13. Displaying the values for all the metrics

6. Related work

Graphical representations of software have long been accepted as comprehension aids. Many tools enable the user to visualize software using static information, e.g., Rigi [12] , Hy+ [1], SeeSoft [4] , or ShrimpViews [18]. The PolymetricViews system of Lanza [11] was a first step in presenting multiple metrics on the same 2D view. On the other hand Chuah and Eick [1] were among the firsts to use sophisticated glyphs to represent complex information on software objects. The use of 3D views to represent software architecture has been advocated by Feijs L. and De Jongin [5]. However they focused on the representation of the relations between modules, themselves represented as Lego bricks distributed in the

3D space. They did not investigate the use of "familiar" metaphors such as the city. The idea of the city metaphor to represent software objects has been proposed recently by Panas [14]. But, in this work the views are static i.e. non navigable. Then, the quantity of information that can be represented is limited. Langelier et al. [9] presented a visualization of software quality made of 3D boxes representing classes, whose dimensions are mapped to quality metrics. But they did not exploit the city metaphor to ease the interpretation of the view nor did they implement the "vertical" navigation inside the boxes to show their contents. Finally, they did not investigate the visualization of the relationships between the classes. On the other hand 3D visualizations have been used by Jazayeri et al. [8] to represent version and release information of software systems. Recently Pinzger et al. showed the use of 2D Kiviat diagrams also to represent software evolution [16]. The latter is a dimension of software systems that we have not yet investigated nor represented. However, since the database we are using contains all the releases of Mozilla, we will display this information in the future.

7. Conclusions and future work

In this paper we presented the Evospaces reverse-engineering tool, which allows its users to investigate large software systems by navigating through a virtual city in a 3D space. The main contribution of this paper is in the use of the "familiar" city metaphor with buildings and people, the mapping of metrics value categories to the texture of the buildings and the navigation mode we implemented. In particular we showed how objects could be zoomed in to represent their contents as "workers". We also showed the use of pipes with a simulation of the flow of information to represent the relationships with their directionality. Then we presented the idea of the "road map" to keep the user informed of his global position when traveling the city. Finally we showed how these 3D techniques could be used to represent a substantial part of a very large system such as Mozilla. Our current work concentrates on the representation of the dynamic information on a system (i.e. its working). We are then looking for supplementary metaphors to represent the interactions (the traffic) in between buildings when objects are sending messages to each other.

8. Acknowledgements

We gratefully acknowledge the financial support of the Hasler Foundation for the "Evospaces – Multi-dimensional navigation spaces for software evolution (project MMI-1976).

9. References

[1] Chuah, M. C., Eick, S. G. - Information rich glyphs for software management data. *IEEE Computer Graphics and Applications*, July1998.

[2] Consens, M. P., Mendelzon, A. O. - Hy+: A hygraph-based query and visualisation system. In *Proc. of the ACM SIGMOD Int. Conf. on Management Data, SIGMOD Record Volume 22, No. 2, 511–516*, 1993.

[3] Demeyer S., Tichelaar S., and Ducasse S. - FAMIX 2.1 — The FAMOOS Information Exchange Model. *Technical report, University of Bern*, 2001

[4] Eick, S. C., Steffen, J. L., Summer E. - Seesoft - a tool for visualizing line oriented software statistics. *IEEE Trans. on Soft. Engineering 18(1),* Nov. 1992

[5] Feijs L., De Jongin R. – 3D Visualizations of Software Architectures. *CACM 41(12), Dec.* 1998.

[6] Gamma E., Helm R., Johnson R., Vlissides J. – Design Patterns. Elements of Reusable Object Oriented Software. *Addison-Wesley Inc*. 1995.

[7] Java binding for OpenGL, https://jogl.dev.java.net/

[8] Jazayeri M., Gall H., Riva C. - Visualizing software release histories: The use of color and third dimension. Proc. *IEEE Int. Conf. on Software Maintenance. ICSM*, 1999.

[9] Langelier, G., Sahraoui, H., and Poulin, P. - Visualization-based analysis of quality for large-scale software systems. In *Proc. of the IEEE Int. Conf. on Automated Software Engineering ASE '05*. 2005

[10] Lanza M. - Object-Oriented Reverse Engineering. PhD Thesis, *Univ. of Bern, Switzerland*, May 2003.

[11] Lanza M., Ducasse S. – Polymetric Views – A Lightweight Visual Approach to Reverse Engineering. *IEEE Trans. on Software Engineering* 29(9):782-795, Sept.2003.

[12] Mueller, H. A. - Rigi - A Model for Software System Construction, Integration, and Evaluation based on Module Interface Specifications. *PhD thesis, Rice University*. 1986

[13] www.opengl.org

[14] Panas Th. - A Framework for Reverse Engineering, *PhD Thesis, Växjö University*, Dec. 2005

[15] Pinzger M. - ArchView : Analyzing Evolutionary Aspects of Complex Software Systems, *PhD Thesis , Vienna Univ. of Technology*, May 2005.

[16] Pinzger M., Gall M., Fischer M., Lanza M. - Visualizing multiple evolution metrics. *Proc. of the ACM symposium on Software visualization* 2005.

[17] Simon H.A. - The architecture of complexity. In: The Sciences of the Artificial, *MIT Press*, 1969.

[18] Storey, M.-A. D., Mueller, H. A. - Manipulating and documenting software structures using shrimp views. In *Proc. IEEE Int. Conf. on Soft. Maintenance*, ICSM, 1995.

Ontobrowse: A Semantic Wiki for Sharing Knowledge about Software Architectures

Hans-Jörg Happel*) and Stefan Seedorf†)

*) *FZI Research Center for Information Technologies*
Research Group Information Process Engineering (IPE)
Haid-und-Neu-Str. 10-14
D-76137 Karlsruhe, Germany
happel@fzi.de

†) *University of Mannheim*
Lehrstuhl für Wirtschaftsinformatik III
Schloss, L 5,5
D-68131 Mannheim, Germany
seedorf@wifo.uni-mannheim.de

Abstract

The development and maintenance of a software architecture involves various stakeholders with different interests. While developers primarily require technical support and guidance for their implementation tasks, architects need means for analysis and documentation. We argue that most approaches and tools insufficiently support the requirements of both groups appropriately, leading to a scattering of architectural information into different information spaces. To resolve this problem we propose Ontobrowse – a lightweight solution which is based upon ontologies and the recent paradigm of semantic wikis. Ontobrowse allows the combination of informal with more formal documentation together with the integration of asserted knowledge from external specification resources. The system architecture and prototypical implementation are described. To illustrate the benefits of our approach, we introduce the case of documenting service-oriented architectures in an enterprise setting.

1. Introduction

A software architecture is the gross organization of a software as a collection of interacting components [11], functioning as a bridge between requirements engineering and system design [11, 29]. Its building blocks are arranged in such a way that both the system requirements and architectural constraints are met [9].

However, it is misleading to talk about *the* software architecture. While every software has an architecture, the actual representations of such an architecture matter. Those representations are a “set of structures” [4] that provide certain “views” such as functional, physical or logical [16]. Most of those views can be assigned a certain purpose, such as quality, communication, analysis or reuse [4, 5, 11].

In this context, two schools of thought can be identified. The developer-centric perspective regards software architecture as a shared mental model [13]. Its purpose is to facilitate communication among the developers by providing a shared understanding of the system under development and enforcing conceptual integrity [29]. In contrast, the system-centric perspective seeks to formalize as much constraints as possible, to allow validation and verification of the implemented system. Various so-called architecture description languages (ADL, see e.g. [19]) have been created to model systems in form of components and connectors, and have formally defined semantics for tool-supported analysis.

Both perspectives are valuable, since the development of software is a process of increasing formalization – from fuzzy user requirements to concrete bits and bytes. As bridge between requirements engineering and concrete design, software architecture has to address both issues. What is sought is thus a solution that offers flexibility in documentation and collaboration as well as a sound formal basis for leveraging machine-interpretable semantics. Although this requirement sounds contradictory, we are suggesting semantic wikis – an extension of the well-known wiki technology – as a candidate technology for solving this trade-off. From our point of view semantic wikis are well-suited to bridge the gap between technical and business documentation, since they a) provide means for collaborative documentation and information exchange regarding a certain subject and b) provide a formal foundation for handling technical descriptions.

The remainder of this paper is structured as follows: First, we introduce the application scenario of enterprise service-oriented architectures (SOA) to further characterize the problem. Second, we describe the basic ideas and concepts behind semantic wikis. In chapter 2, we present the conceptual architecture and prototypical implementation of Ontobrowse. In chapter 3, it is shown how it can be applied to the aforementioned application scenario by providing a SOA ontology for imposing a knowledge structure and plugins for integrating external specifications. After an overview of related work in chapter 4, we conclude by summarizing the potential benefits of a semantic wiki approach.

1.1 SOA Application Scenario

Before we propose semantic wikis as a solution we first need to delve into the main issues addressed by this work. An application scenario which nicely illustrates the common problems of documenting and maintaining architectural knowledge in enterprises is service-oriented architecture (SOA).

Service-oriented computing [14] is an emerging paradigm which is built on the notion of enterprise application systems being assembled from independent, loosely-coupled services. It has lifted the development of business applications to a higher level of abstraction. Instead of thinking in design and implementation categories like components or objects, software functionality is bundled in services that correspond to business operations of the organization. Complex workflows can be realized by aggregating functionality from simple services. A service is a coarse-grained, discoverable software entity which provides its logically-cohesive business functionality through well-defined interfaces. A concrete software infrastructure implementing this paradigm is described as SOA [14].

In an organization pursuing the realization of a SOA, the standard working processes change for both developers and business experts. Service developers have to think in specification terms rather than taking an implementation view. Due to the black-box realization of services, metadata describing their properties is crucial. Also, the enterprise-wide deployment of services calls for better documentation and communication among the responsible developers. Both aspects lead to new kinds of information needs for developers.

Business experts on the other hand are interested in available functionality and operational efficiency. Since service-orientation leads to a rising level of alignment between business processes and IT implementation, it is important to monitor and guide the development of the service landscape. Governing the evolution of a service-oriented architecture becomes important, because changes at the service level may have a direct impact on business processes.

So the paradigm of service-orientation ties the individual workflows of software developers and business experts much closer together. This situation of multiple stakeholders results in diverse requirements for tool support. Besides managing the appropriate technical and business aspects this includes communication and documentation among the various participants involved in the development process. Moreover, different tools and description formats may be used.

This heterogeneous, dynamic environment motivates the two building blocks of our application. Unlike tools that concentrate on specific aspects such as service orchestration or lifecycle management, we intend to provide an integration space for both developers and business experts, since both worlds often maintain their separate information set about the same subject. Therefore we propose a semantic wiki approach which is able to integrate both informal and formal descriptions typically managed within a SOA project.

1.2 Semantic Wikis

The software genre of a "Wiki" describes a lean approach to web-based content management, allowing multiple users to collaborate on the creation of a document. Basically, a "small web" is imposed by the titled wiki pages and their contained hyperlinks. However, in contrast to the World Wide Web, wikis provide editing capabilities and enforce some conceptual coherence. Due to these characteristics, wikis have become popular in various application areas, in particular software engineering (see e.g. [7] for an overview).

Although traditional wikis provide a top-level structure by its separation in "pages", the actual information on a certain page is stored in an unstructured manner. Classical wiki software does not provide a standardized way to add structured information to a certain topic. However, if some information was made available in a machine-interpretable format, a site like the Wikipedia could heavily benefit because its pages contain a lot of potentially structured information [21].

Thus, several projects started to implement semantic extensions to the wiki approach[1]. Because most of the implementations are still in an experimental stage, there is no clear definition of what a semantic wiki exactly is. However, some shared characteristics can be identified: Semantic wikis extend traditional wikis in the way that they allow structured knowledge to be described in a formal language, instead of processing solely hypermedia-based content. This is either be done by appending metadata to wiki pages or by including knowledge inside the unstructured text by using extensions to the wiki markup language. The latter approach is used by the SemanticMediaWiki project [21], which extends the existing wiki markup to enrich hyperlinks between wiki pages with semantic relations. The approaches have in common that they interpret the existing wiki pages as entities, and hyperlinks as relations among them. Adding semantics just formalizes this implicit structure, thus transforming the knowledge inside the wiki into a kind of "ontology", which is defined as "an explicit specification of a conceptualization" [12]. Since most semantic wiki implementations are rooted in the semantic web community, they adopt existing standards for ontology representation such as the Resource Description Framework (RDF) and the Web Ontology Language (OWL). RDF is a simple graph-like format for describing metadata about resources [22] that are described using a Uniform Resource Identifier (URI). This makes it easy to annotate wiki pages with arbitrary metadata. OWL is defined on top of RDF(S) and provides a standard ontology vocabulary for describing ontologies based on description logics [23].

While the knowledge representation community typically differentiates between structural knowledge, describing concepts or classes ("TBox") and assertional knowledge, describing instances of those concepts ("ABox"), this distinction is rather blurred in current semantic wikis. For application scenarios like formalizing the knowledge inside Wikipedia, this is perfectly suited, since "conceptual" knowledge may emerge as well as "assertional" knowledge.

However, in a more focused application scenario, a more restrained conceptual structure, specifying concepts and their relations, might be useful to predetermine the initial structure of the information space. In opposite to semantic wikis supporting the emergence of knowledge structures out of existing wiki content, this alternative proposal results in a semantic wiki based on a predefined knowledge structure, that provides a frame for adding further assertional knowledge inside a limited domain. We consider this approach to be viable for the purpose of bringing together formal specification with informal documentation of architectural knowledge. We will now describe the Ontobrowse semantic wiki and then illustrate its contributions by proposing an exemplary knowledge structure for the SOA scenario.

[1] http://wiki.ontoworld.org

2. The Ontobrowse Semantic Wiki

There are various kinds of tools and description formats that can be constructed for sharing architectural knowledge. The main purpose here is to provide a lightweight solution that can be adapted to integrate information from existing environments, e.g. service descriptions managed in a SOA repository.

On top of the problem description in the chapter 1 three main requirements are identified:

R.1 Browse, search and query architectural knowledge
R.2 Manage (informal and formal) documentation
R.3 Enable consistency checking and verification

Although R.3 is not discussed in this paper, the proposed ontology-based solution provides general support for it. Moreover, there are two key constraints for a practical solution:

C.1 Combine formal and informal knowledge of a software architecture
C.2 Enable the integration and augmentation of knowledge from external sources

Semantic wikis make it possible to combine unstructured and machine-interpretable knowledge (C.1), and ontologies define a knowledge structure, which can serve as a contract for integrating instance data from external architectural description resources (C.2). Both semantic wiki and ontologies therefore constitute the building blocks of the Ontobrowse architecture.

2.1 Wiki architecture

The core of our wiki architecture consists of one or more ontologies and a corresponding knowledge base. While the ontologies define the knowledge structure, i.e. the boundaries in which instances can be described; the knowledge base holds the instances. Within the given knowledge structure, it is possible to create or augment instance descriptions in two different ways: First, external tools can plug into the wiki application and map architectural description resources to instances in the knowledge base. Second, a wiki user can use the interface to describe properties – may it be formal or informal – about instances of concepts.

In our SOA scenario, a knowledge structure may be described by the concepts "service" and "business object" together with their properties and axioms. Another example is a concept in a domain vocabulary, which can be used to describe SOA elements with additional semantics. The instances are represented by actual services and business objects developed in a SOA project. Each concept, relation or individual is visible to the user as a "wiki page". A wiki page typically consists of unstructured content and properties that make statements about this page, e.g. a business object which is semantically described by a domain concept. We also refer to a wiki page as an "entity", because it is contained in the knowledge base and can be requested with a unique identifier (URI).

Following the characterization of the wiki structure, we now introduce the key components of the system architecture as depicted in Figure 1: a Web interface, a wiki manager, an ontology API to access the knowledge base and a plugin manager.

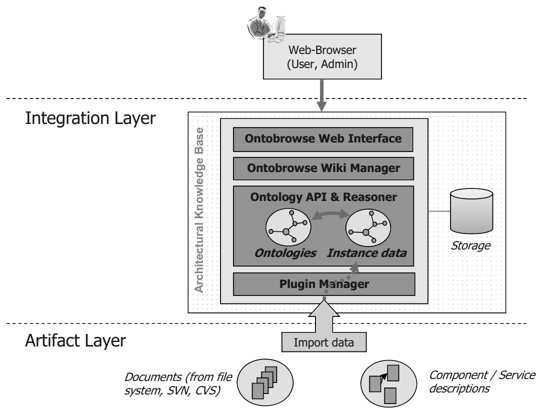


Figure 1: Ontobrowse architecture

Starting with the client's perspective, all functionality is exposed by a Web interface to both users and administrators. In the application layer, a wiki manager bundles the functions for fulfilling the requirements, such as processing page requests, editing textual documentation and instance property values, searching and deductive querying, and verifying user authorizations. Entity (page) descriptions are returned by an ontology API, which wraps the underlying reasoner and ontology processing tools.

Administration tasks include ontology management and plug-in management. In most cases, ontologies are constructed during the setup phase using an ontology editor such as Protégé[2] and then uploaded by an administrator using the wiki manager. However, it is still possible to add new properties to concepts or entire ontologies throughout the operation phase.

Plug-in management refers to a distinctive feature, which allows mapping and importing instance data from external sources. To a great extent the instance data will be embodied in applications and artifacts that are managed outside the wiki, e.g. service specifications in the case of SOA documentation. The data thus has to be imported from external sources, such as configuration management systems (see Figure 1). Therefore, the plug-in manager exposes standard interfaces that allow tools to retrieve artifacts, map them according to an ontology, and create or update instance data in the knowledge base. As described in more detail in the SOA case, imported instance data can be augmented and referenced for further documentation.

2.2 Prototypical Implementation

The system architecture described above was implemented in Java. Our main concern was to achieve a clean separation into loosely coupled components so that some parts can be easily substituted by other implementations, e.g. using a different reasoner in the ontology API. The application and persistence layer were implemented with Spring[3] and Hibernate[4], for the Web interface we used Java Server Faces[5].

The Web Ontology Language (OWL) was employed as knowledge representation format. For processing OWL files and reasoning in the ontology API we worked with the Jena 2 Semantic Web framework[6]. Deductive queries with SparQL [25] are supported accordingly. Jena also supports user defined rules for knowledge generation and consistency checking, however, this feature is not discussed in this paper. A feature that had to be added on top of the ontology API is full text search spanning both entity descriptions in the knowledge base and textual descriptions.

One important aspect for the user acceptance of a Wiki is an easy to understand Web interface. If the wiki contains hundreds of concepts, the knowledge space will quickly get too difficult to navigate. That is why administrators can assign an OWL annotation property to a concept in order to mark it as visible on top-level. Moreover, annotations have been used to control editing of instance properties. In the SOA case, a service instance could be augmented with additional metadata, e.g. about the responsible developer. Instance properties that have been imported via plug-ins are thus marked as "non modifiable". This way (one-way) consistency with external sources is ensured.

[2] http://protege.stanford.edu

[3] http://www.springframework.org/
[4] http://www.hibernate.org/
[5] http://java.sun.com/javaee/javaserverfaces/
[6] http://jena.sourceforge.net/

3. Application to the SOA scenario

The primary goal of Ontobrowse is to provide a non-invasive solution, which can be extended and tailored to individual project needs. Depending on the enterprise setting project-specific architectural knowledge may be distributed throughout various sources.

Concerning the SOA scenario we assume that architectural descriptions are managed in a single file system or alternatively in a service repository. A typical candidate artifact is the specification of a service together with its interfaces. The services may either be specified in a custom or standardized format, e.g. proprietary XML or Web Service Description Language (WSDL). Other artifacts also hold valuable knowledge. A SOA project may use a considerable number of WS* standards, e.g. Business Process Execution Language (BPEL4WS), WS-Policy or WS-Security, to cover important aspects such as service orchestration and non-functional properties.

Two steps are necessary for integrating architectural descriptions into the wiki: First, we require a conceptual mapping from a description format to a unifying *SOA ontology*. Since more than one format might be used to describe a "service", the ontology reduces conceptual ambiguity and enables information integration. Second, a *plug-in* has to be defined, which performs the actual mapping of instances from a source into the knowledge base.

3.1 SOA ontology

The purpose of the SOA ontology is to supply the initial structure to the semantic wiki enabling the documentation of services by both business experts and developers. For this reason, it has to include central concepts of a SOA like services, business objects and domains concepts. Moreover, the ontology should provide a basic abstraction, in which the actual information about SOA elements from the external data sources is mapped onto.

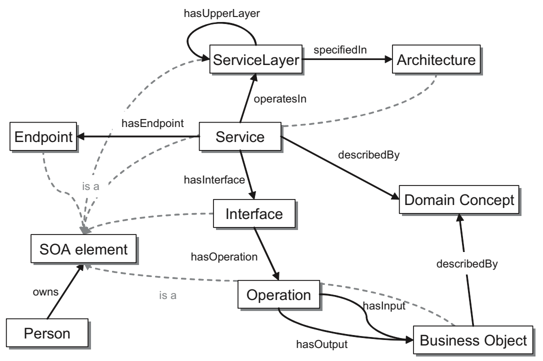
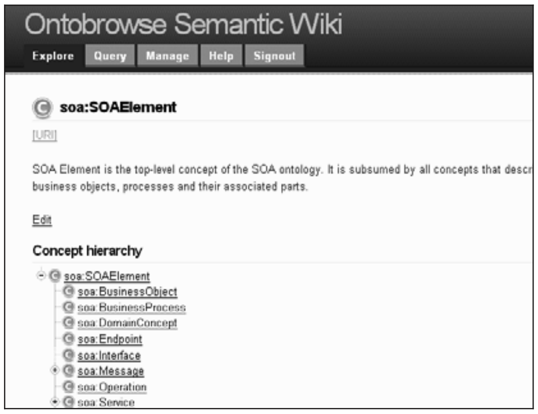


Figure 2: SOA ontology

There exist a number of specification standards and ontologies which provide valuable input during ontology development. For example, the service component model in WSDL 2.0 is partly reflected in the ontology. Nevertheless a strict ontology mapping proposed for WSDL [26] is not applicable here because it would lead to scattering of the page structure. Other useful sources are the foundational ontologies being developed within Semantic Web Web Services [2, 10, 20] and Web services architecture [24].

The SOA ontology is visualized in Figure 2. The presented ontology is generic in the way that it incorporates common characteristics of widely-accepted standards. Figure 3 shows an example wiki page returned for the concept "SOAElement" with its direct subconcepts. Other information displayed for a concept are its instances (or individuals), object properties (e.g. "hasInterface") and data type properties (e.g. "version").



soa:SOAElement

[URI]

SOA Element is the top-level concept of the SOA ontology. It is subsumed by all concepts that descr
business objects, processes and their associated parts.

Edit

Concept hierarchy

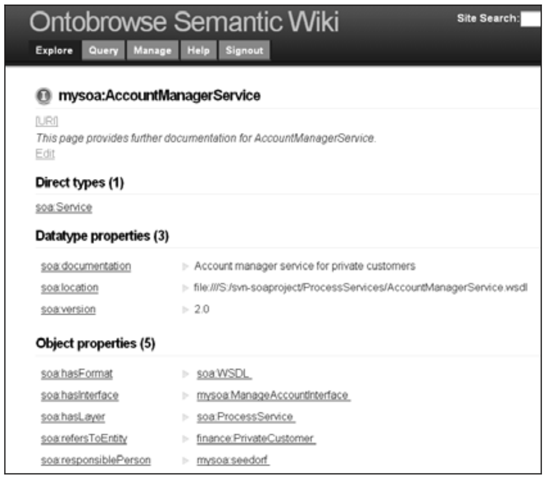
- soa:SOAElement
  - soa:BusinessObject
  - soa:BusinessProcess
  - soa:DomainConcept
  - soa:Endpoint
  - soa:Interface
  - soa:Message
  - soa:Operation
  - soa:Service

Figure 3: Concept hierarchy of SOA Element

It is possible to develop additional ontologies that cover a particular information need in the SOA project or organization. Instance data corresponding to the ontology may either be maintained in the wiki or imported from external sources by creating appropriate plugins. This ensures high flexibility and enables to augment SOA elements with further knowledge. This may e.g. include domain concepts given by a domain ontology or organizational knowledge such as persons responsible for SOA elements.

3.2 Mapping SOA artifacts

We now explain how actual service descriptions are imported into the wiki and enriched with additional metadata. WSDL 2.0 service descriptions serve as an example, while the process is analogous for other formats and source types. First a one-way mapping between WSDL service descriptions and the SOA ontology has to be defined. We extended the WSDL format to accommodate additional service properties such as version and architectural layer. The actual mapping is executed by a Java program which conforms to the Ontobrowse plug-in interface. It takes a WSDL file as input and produces an OWL file conforming to the SOA ontology. A wiki administrator is then responsible for configuring input sources (CVS, file system) and update types (manual, timer task, update event). Based on this configuration the plugin manager component is responsible for updating the knowledge base automatically.



mysoa:AccountManagerService

[URI]

This page provides further documentation for AccountManagerService.

Edit

Direct types (1)

soa:Service

Datatype properties (3)

soa:documentation	Account manager service for private customers
soa:location	file:///S:/svn-soaproject/ProcessServices/AccountManagerService.wsdl
soa:version	2.0

Object properties (5)

soa:hasFormat	soa:WSDL
soa:hasInterface	mysoa:ManageAccountInterface
soa:hasLayer	soa:ProcessService
soa:refersToEntity	finance:PrivateCustomer
soa:responsiblePerson	mysoa:seedorf

Figure 4: Service instance

Figure 4 shows a service instance imported from a WSDL file. Wiki users can now browse and search the information space, edit textual descriptions and assign additional property values (metadata) to an instance, e.g. the responsible person for a service. However, editing is restricted to especially annotated properties. Moreover, these property values are exclusively visible *within* the wiki.

A query interface enables users to define chained queries consisting of sentences with *subject*, *predicate* and *object* (e.g. all services "x" defining interface operations with the output "Customer"). Matching entities are returned for the variables defined by the query.

4. Related Work

The basic building blocks of our approach – wikis for software documentation and ontologies as formal models for software systems – have been used in some earlier works. Aguiar and David present a wiki-based approach to integrate heterogeneous software specification resources into a single document [1], while Bachmann and Merson investigate the advantages of wikis compared to other architecture documentation tools [3]. However both approaches lack a formal model – the information is managed in an unstructured way.

On the other hand, formal ontologies have been presented for architectural documentation [28] and for building "software information systems", describing the interrelationships of domain models and source code [8, 27]. These works either lack appropriate tool support or follow a very strict philosophy of software architecture.

Some aspects of a software architecture that have been covered in this paper might also be described using an ADL [19]. However, the case described in this paper substantially differs from the purpose of architecture description languages. Whereas ADLs solely focus on the formal specification of concrete architectures, our approach is also capable of supporting informal aspects of software architecture. Our model of architectural description allows exchanging knowledge between different architectures and is not limited to a fixed set of language elements. It may be extended to support further architectural dimensions such as organizational issues or requirements traceability.

The notion of "architectural knowledge" is currently discussed in terms of representing decisions in the architecture development process and the "rationale" behind them [15, 18]. While our tool is flexible enough to incorporate such information (e.g. based on [17]) it was not the focus of our underlying use case. Thus, the interpretation of architectural knowledge in the context of our work is a much broader one, incorporating unstructured textual documentation as well as formal architectural specifications.

5. Conclusion and Outlook

In this paper, we presented semantic wikis as a novel approach to share knowledge about software architectures. The semantic wiki architecture and functionality was illustrated by the example of service-oriented architectures. However, the selected application scenario can easily be generalized to support arbitrary architectural styles. It can be tailored to project-specific needs by providing an ontology to set up the initial structure of the wiki.

We think that the Ontobrowse semantic wiki approach contributes to several issues in architecture documentation and knowledge sharing. First, it builds upon the general advantages in software documentation offered by wikis. Wikis respond well to collaborative settings and provide a scalable way for the documentation of large software projects.

Second, it bridges the gap between informal documentation and technical service descriptions. We consider this an important issue, since software architectures operate at the intersection of user requirements and system design. Semantic wikis enable an informal style of documentation while also supporting to incorporate machine-interpretable knowledge about technical descriptions. Thus, they allow collecting relevant information about a software architecture at a central place that has so far been maintained separately.
Third, the formal model underlying the semantic wiki supports a better searching and browsing of architectural elements, ensuring semantic consistency and the incorporation of content from external repositories. The formal model can also be easily extended. This may be used to include external knowledge about standards or information about organizational structure, which is particularly important in distributed development settings [6]. Finally, referencing specification and implementation artifacts contributes to improving traceability throughout different stages of the software lifecycle.

6. Acknowledgements

This work was partly supported by the European Commission (IST-35111-TEAM), the BMBF-funded project WAVES and by the Landesstiftung Baden-Württemberg foundation (Project CollaBaWue). The authors are responsible for the content of this publication.

7. References

[1] Aguiar, A., and David, G.: WikiWiki weaving heterogeneous software artifact. In: Proc. of the 2005 international symposium on Wikis, San Diego, CA, 2005, pp. 67-74.
[2] Akkiraju, R., et al.: Web Service Semantics - WSDL-S, W3C Member Submission, 7 Nov. 2005.
[3] Bachmann F., and Merson, P.: Experience Using the Web-Based Tool Wiki for Architecture Documentation. Technical Note CMU/SEI-2005-TN-041. September 2005.
[4] Bass, Len; Clements, Paul; Kazman, Rick: Software Architecture in Practice. 2. Addison Wesley, 2003.
[5] Bosch, J.: Design and use of software architectures: adopting and evolving a product-line approach. ACM Press/Addison-Wesley Publishing Co., 2000.
[6] Cockburn, A.: The interaction of social issues and software architecture. In: Commun. ACM 39, October, Nr. 10, 1996, pp. 40-46.
[7] Decker, B., Rech, J., Ras, E., Klein, B., Hoecht, C.: Self-organized Reuse of Software Engineering Knowledge supported by Semantic Wikis. In: Proc. of Workshop on Semantic Web Enabled Software Engineering, November 2005.
[8] Devanbu, R. J. Brachman, P. G. Selfridge and B. W. Ballard: LaSSIE - A Knowledge-Based Software Information System, ACM Comm., 34(5), 1991, pp. 34-49.
[9] Eden, A.H., and Kazman, R.: Architecture, design, implementation. In: Proc.of the 25th International Conference on Software Engineering (ICSE-03). Piscataway, NJ: IEEE Computer Society, May 3-10 2003, S. 149-159.
[10] ESSI WSMO: Web Service Modeling Ontology (WSMO). http://www.wsmo.org/, 2005.
[11] Garlan, D.: Software Architecture: A Roadmap. In: Proceedings of the 22th International Conference on Software Engineering (ICSE-2000), ACM Press, 2000, pp. 91-101.
[12] Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. 5, 1993, 199-220.
[13] Holt, R.C.: Software Architecture as a Shared Mental Model. In: ASERC Workhop on Software Architecture, University of Alberta, August 2001.
[14] Huhns, M.H., and Singh, M.P.: Service-Oriented Computing: Key Concepts and Principles. IEEE Internet Computing, vol. 9, no. 1, 2005, pp. 75-81.
[15] Jansen, A. and Bosch, J.: Software Architecture as a Set of Architectural Design Decisions. In: Proc. of the 5th Working IEEE/IFIP Conference on Software Architecture (Wicsa'05), Washington DC, 2005, pp. 109-120.
[16] Kruchten, P.: The 4+1 View Model of Architecture. In: IEEE Softw. 12 November, Nr. 6, 1995, pp. 42-50.
[17] Kruchten, P.: An Ontology of Architectural Design Decisions. In: Proc. of 2nd Groningen Workshop on Software Variability Management, Groningen, NL, 2004, Rijksuniversiteit Groningen.
[18] Kruchten, P., Lago, P., van Vliet, H., and Wolf, T.: Building up and Exploiting Architectural Knowledge. In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (Wicsa'05), Washington DC, 2005, pp. 291-292.
[19] Medvidovic, N., and Taylor, R. N.: A Classification and Comparison Framework for Software Architecture Description Languages. In: IEEE Trans. Software Eng. 26(1): 2000, pp. 70-93.
[20] OWL Services Coalition: OWL-S Semantic Markup for Web Services. http://www.daml.org/services/owl-s/, 2004.
[21] Völkel, M., Krötzsch, M., Vrandecic, D., Haller, H., Studer, R.: Semantic Wikipedia. In: Proceedings of the 15th international conference on World Wide Web, WWW 2006, Edinburgh, Scotland, May 23-26, 2006.
[22] W3C: Resource Description Framework (RDF), 2004.
[23] W3C: Web Ontology Language (OWL), 2004.
[24] W3C: Web Services Architecture. W3C Working Group Note, 11 February, 2004.
[25] W3C: SPARQL Query Language for RDF. W3C Working Draft 4 October 2006.
[26] W3C: Web Services Description Language (WSDL) Version 2.0 - RDF Mapping, 2006.
[27] Welty, C.A.: Software Engineering. In: Description Logic Handbook, 2003, pp. 373-387.
[28] Welty, C.A., and Ferrucci D.A.: A Formal Ontology for Re-Use of Software Architecture Documents. ASE, 1999, pp. 259-262.
[29] Witt, B., Baker, F. and Merritt, E.: Software Architecture and Design: Principles, Models and Methods, Van Nostrand Reinhold, 1994.

Building Business Considerations into Enterprise Application Designs

Rattikorn Hewett and Aashay Thipse
Department of Computer Science, Texas Tech University
rattikorn.hewett@ttu.edu, aashay.thipse@gmail.com

Abstract

Despite of many technology advances, enterprise application designers are facing with organizational challenges in customizing the application to fit with existing business processes and rules in order to ensure effective coordination of work across the enterprise. Much research has advanced the technical aspect of the enterprise application design and development. However, business implications of the design products are seldom addressed or analyzed structurally. A perfectly performed enterprise application can still put the business of the enterprise deploying it at risk if it does not support sound business logic. This paper proposes building business considerations into the software design by using a business risk as a means for evaluating and monitoring the design of an enterprise application under development. To do this, we present an analytical approach to systematically analyze business risks from characteristics of an early high-level enterprise application design. The paper describes the approach and validates it with an illustration on enterprise application design for online shopping.

Keywords: risk-based software design, software risks, enterprise applications.

1. Introduction

Application development is driven by the demands of modern enterprises to gain competitive advantages. *Enterprise applications* are the software that performs business functions in order for an enterprise to obtain higher operational efficiency, lower cost and consequently increased profitability [1, 9]. Services provided by enterprise applications include online shopping and online payment processing, interactive product catalogue, and automated billing systems. From a system's perspective, enterprise application software has to work for a large number of users, on a large amount of data, with a rapid response times. Enterprise applications can be categorized into various types including transaction-oriented (e.g., accounting), reporting (e.g., customer information tracking), batch (e.g., production scheduling), and real-time applications (e.g., online shopping).

Real-world enterprise applications are highly complex and rarely monolithic. They often deal with various platforms and data stores both internal and external to an organization. Designing an enterprise application is intrinsically difficult since it involves complex large-scale integration of multiple applications in multiple environments with a large number of diverse users across the enterprise. Despite many technological advances, enterprise application designers are facing with organizational challenges in customizing the application to fit in with existing business processes and rules in order to ensure effective coordination of work across the enterprise. Developers recognize the need to keep up with today's business challenges to project enterprise systems into various client channels in a way that is reliable, productive, and sustainable to frequent updates [14].

Much progress has been made on the technical aspects of enterprise application design and development including enterprise architectures, environments (e.g. java-based J2EE, or a Pearl-based P5EE platforms) and methodologies for distributed application design and integration [1, 9, 10, 14]. For business challenges, efforts on incorporating business processes into enterprise applications are mostly in a requirement stage in order to create a design [4, 8, 12, 14]. However, business implications of the design products are seldom addressed or analyzed structurally. A perfectly performed enterprise application can still put the business of the enterprise deploying it at risk if it does not support sound business logic. The ability to assess this risk at an early stage of software development life cycle can result in cost savings.

In this paper, *business risk* refers to risk associated with behavior of software that has adverse consequences to business objectives of an enterprise. Our research proposes building business considerations into enterprise application design by using a business risk as a means for evaluating and monitoring an evolving design of an enterprise application under development. The intent of our work is to assess quality of the design in a business context and to use the findings to create better design. We present an analytical approach that employs risk methodology to systematically analyze and quantify business risks from characteristics of a high-level early application.

The rest of the paper is organized as follows. Section 2 presents related work. Section 3 gives a motivating scenario of cases on Business to Customer (B2C) e-commerce order processing. Section 4 describes the proposed analysis approach. Section 5 gives illustrations on the scenario and the paper concludes in Section 6.

2. Related Work

Much work in enterprise applications has focused on development of enterprise systems [1, 9, 10, 14] including business-oriented aspects of how to integrate business logic into design specifications [4, 8, 12, 14]. Most of these work is concerned with enterprise business process modeling [4, 11], requirements and design specifications using the UML (Unified Modeling Language) [2, 6] activity diagrams or its variants to represent enterprise workflows [12]. While we share similar principles on design representation, our work is different in that we focus on assessing quality of the enterprise application design products rather than how to specify the design.

Risk methodologies have been employed to assess safety and performance associated with software systems in various application domains including industrial engineering, business and space science [7, 15, 16]. FMEA (Failure Mode and Effect Analysis), a risk assessment approach has been applied to gain understanding about usability of an online enterprise system [17]. In the work in value based software engineering Risk-based approach have been introduced in software project management [5] Unlike our approach, none of these work address risks associated with software components in business contexts. Risk concepts are commonly used in enterprise management [13]. However, techniques for identifying likelihoods of business risks in associated software systems are often subjective rather than analytical and objective. Csertan et al. [8] uses an extended UML profile to design the business process and employs FMEA to assess dependability of an enterprise online system. Unlike our proposed approach, they focus on fault identification rather than risk quantification, which is a crucial step in risk analysis.

Our work is most similar to the methodology proposed by Yacoub et al. [16]. Both use features of design dynamic diagrams (i.e., UML sequential diagrams, activity diagram) to estimate likelihood of access of software components. However, their terminating condition for risk propagation is pre-specified, whereas ours uses the worst case estimates. Furthermore, instead of assessing the reliability of software components, we assess the business risks associated with enterprise applications deployed. Their reliability risks are estimated from the complexity of software components whereas our business risks are based on characteristics of software design as well as business logic to be deployed.

3. Motivating cases: Online Shopping Process

Online shopping is an important enterprise application in e-business and e-commerce that is increasingly becoming a necessary business function of modern enterprises.

Typically, the online shopping business process involves the following activities: browse products, see product details, add selected products to a shopping cart, enter payment and shipping information, and submit the payment. The business objective of an enterprise is to sell products to the customer visiting the website. Different organizations may have different business rules or policies that enterprise application design must adhere to.

Case 1: A company *A* has a strict customer access policy that requires a customer to register and login before he/she can perform any online shopping activity.

Case 2: A company *B* has a more open access policy that only requires a customer to provide credentials when they are ready to purchase and make the payment.

Company *A* sees benefits of validating customer credentials (for identification) at an early stage in order to solicit sales only with previous or new serious buyers. Having customer credentials makes it easy to keep track of the items they are looking for. Even if they do not purchase any item at that time, the online shopping system can remind them of what they were looking for, the next time they log in. Furthermore, if a customer looks for items in a specific category, the system can notify him/her when that category gets updated.

Unfortunately, while this policy has some advantages, it also has certain disadvantages. By not allowing a customer to perform any activity (e.g., browse, see product details) unless he/she logs in, the company loses a chance to sell more products. The situation can be more damaging if the system is dealing with first time visitors, where requirements for opening new accounts may not be welcomed. Most customers would not want to share their credentials and open the account unless they have some ideas of what the company has to offer. As a result, these potential customers may decide to shop elsewhere resulting in a loss of customers.

On the other hand, company *B* will not face the above issue since it requires customer credentials (for login or creating a login account) only if the customer wants to buy products and makes a payment. Here, once a customer knows that the company has products of his choice at a competitive price, he/she will not mind spending time to create an account. Thus, a policy for company B trades a chance to increase sales to new customers with a chance to be more selective on customers, particularly those who already know of the company's reputation.

At first, it may not be obvious for novices if such a small difference in an access policy can make different impacts on success or failures of an enterprise. Earlier designs of online retail shopping applications tend to follow Case 1. However, experienced designers can see that *B*'s policy in Case 2 would yield better. This is evidenced by many of today's online retail shopping applications. They mostly follow *B*'s policy and even with an option that allows customer to purchase without creating an account.

An enterprise application design must support the application's business logic, which may have high impacts on success or failure of the task at hand. Therefore, having the approach to analyze business implications of the design is crucial especially when business processes, rules, or business logic are not identified or well understood.

4. Proposed Analytical Approach

4.1 Risks and Business Considerations in Designs

Risk refers the possibility of some undesirable event along with the likelihood of its occurrence. Risk can be quantified by taking an estimate of the subsequent cost of the undesirable event and multiplying it by an assumed probability for the event [7, 15]. We refer to *business (software) risk* as risk associated with behavior of software that has adverse consequences to business objectives of an enterprise. *Risk analysis* is a process that aims to assess, manage and communicate risk information in order to assist decision-making. *Risk assessment* involves identification of adverse situations (or *hazards*) and quantifying risks by determining the *likelihoods* of their causes and the *severity* of their impacts. The risk measures obtained can assist in cost/benefit analysis for resource planning to prevent, eliminate or mitigate the undesired situations. These are the activities of *risk management*.

Risk analysis ties technical issues and business considerations directly to the operations of the enterprise using the application software. Thus, risk methodologies can be applied to build business considerations into the design by using a business risk as a means for evaluating and monitoring the design of an enterprise application under development. Figure 1 gives an overview of our proposed design process. Given a specific model of enterprise application design at the component level, risk quantification estimates business risks and results in a risk ranking of software components, based on their application usage (business processes). Risk mitigation selects a strategy and identifies appropriate countermeasures for high-risk components. A cost/benefit tradeoff analysis helps practitioners evaluate the selection of technologies to mitigate risks. If the selected technology is incorporated into a design, the next cycle of risk analysis for a modified design will begin. The cycle can loop until the design is acceptable.

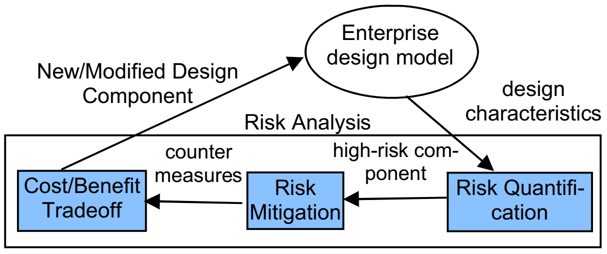


Fig 1. Evolving enterprise application design.

4.2 Business Risk Quantification

Our main objective is to gain understanding of business implications resulting directly from a given early high-level enterprise application design. To do this, we propose using business risk as an objective means for assessing business implications. Unlike existing traditional risk methodologies, our approach aims to develop a systematic methodology for analyzing the design in an objective and structured (as opposed to ad-hoc) manner. We also want the results obtained to be informative in that they can pinpoint where the design deficiencies are (high risk design component) for further examination and improvement.

Risk computation requires two factors: the likelihood of an undesirable event along with its subsequent cost. In a business context of an enterprise application, undesirable events relate to failures to satisfy business objectives, which rely on a specific task and application. Thus, the cost estimates are enterprise context dependent. On the other hand, the likelihood of an undesirable event depends on dependability (i.e., integrity, reliability and availability) of the enterprise application as well as its underlying business logic used in the enterprise. For simplicity, without loss of generality (of our approach), we assume that the system is dependable so that we can focus on how the enterprise application design (with underlying business logic) impacts on success of the enterprise business. To estimate the likelihood of an undesirable event objectively, we employ heuristics based on relevant design attributes, in particular, number of interactions among the design components.

An early stage of enterprise application design starts after business processes modeling and identification of business rules of the enterprise. Business rules govern business structures, which include control constraints and business policies. They can often be identified from use case scenarios and included as parts of the data and activity controls. Business processes of the organization are typically described in terms of workflows, which can be modeled by UML activity diagrams [11]. Thus, to be specification language independent, it is reasonable to assume that an early high-level enterprise design is expressed in terms of an extended activity or workflow diagram, where software components of the enterprise applications and use case activities in use cases are represented [3]. The software design includes all use cases, each of which may have one or more scenarios.

Given a high-level enterprise application component-based design expressed in terms of an extended workflow [3] and let s_k be the likelihood of scenario k. The following describes our proposed approach to estimate design-level business risks.

(1) **Determine interaction rates in each scenario:** For each use case, for each scenario k, for any two components i and j, compute $I_{ijk} = n_{ijk}/N_{ik}$, where I_{ijk} is the *in-*

teraction rate from *i* to *j* in *k*, n_{ijk} is the number of interactions from *i* to *j* in *k*, and N_{ik} is the number of all interactions out of *i* in *k*.

(2) **Determine transition rates in all scenarios:** Construct a *dependency graph* whose nodes include an initial state, final states and software components. Component transitions are based on interaction rates from all scenarios. I.e., t_{ij}, the *transition rate* from components *i* to *j*, can be calculated from $\sum_k s_k I_{ijk}$. Normalize t_{ij} so that a total sum of transition rates from each component is one.

(3) **Compute usage likelihood estimates**: The likelihood of each component usage can be estimated by identifying all access paths from the starting point of the application execution to the component. For each path, estimate its likelihood by multiplying all of the transition rates (t_{ij}'s) along the path to the component. For conservative estimates, select the maximum likelihood estimate obtained. Specifically, we are interested in a path that leads to a final state that does not fulfill enterprise business goals and thus, constitutes a failure state.

(4) **Severity Analysis**: Determine severity of each adverse action in each component. In particular, we are interested in analyzing consequences of a business failure state. The analysis can help identify countermeasures for design improvement. Quantify the severity and select the worst case in each of the failure state.

(5) **Compute business risks**: For each failure state, multiply the resulting likelihood obtained in Step (3) by its corresponding severity obtained in Step (4). Summation of these products gives an estimate of business risk associated with the application software. That is, for a set of business failure states, *F*, we obtain a total business risk, *R* as follows: $R = \sum_{i \in F} p_i s_i$, where p_i is the maximum likelihood to reach *i* and s_i is the severity cost of failure in *i,* for a failure state $i \in F$.

Note that in Step (3), since the transition rate is no greater than one, thus, the longer the path to reach a component is, the less chance for the component to have influences on the business process. This is true, for examples, for components that are never or rarely used. Furthermore, by considering the maximum likelihood to reach a state, we end up choosing the shortest path to the state. Therefore, this eliminates an issue of likelihood estimates on a cyclic path. Thus, the heuristic function we use for estimating the likelihoods appears to be effective and consistent with intuitions.

In Step (4), because failure at different states has different implications for business, it is important to study impact of failure at each of these states explicitly. Moreover, there can be more than one consequence associated with each failure state. Thus, our approach considers all the consequences associated with each failure state and selects the most severe effect to impact the enterprise business.

5. Illustrations

This section illustrates and tests the proposed approach with the two motivating cases described in Section 3. To focus on the methodology, we consider a simplified application for online shopping. To evaluate the approach, we apply the proposed approach to the two cases, analyze and compute business risks. The resulting business risks are validated with anticipated result as described in Section 3.

5.1 Case 1: Login Before Enter

Figure 2 shows a high-level application design of the online shopping (Case 1 of Section 3), expressed in terms of an extended activity diagram. A customer is required to login before he can perform the online shopping activity. Some customer may quit at this point (state q_1), otherwise he/she can carry on to login process.

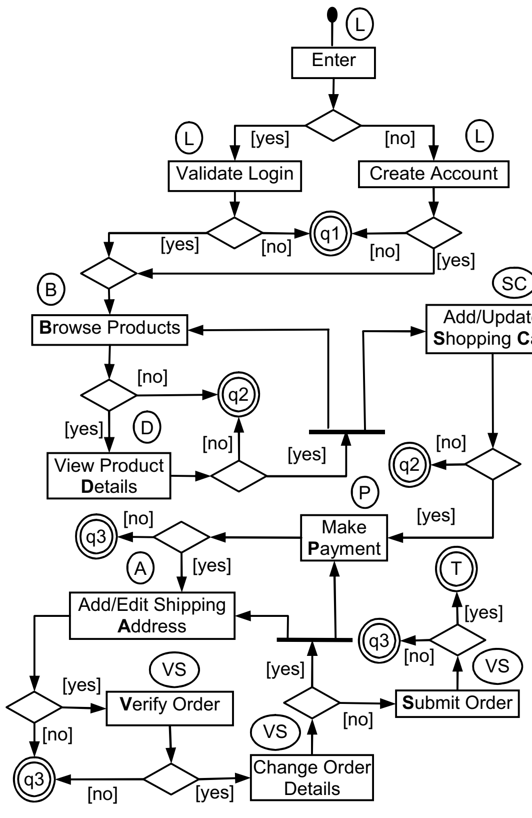


Fig 2. The online shopping Case 1: login before enter.

A new customer will be directed to create a new account by providing his/her credentials. The system validates the login credentials of the customer who already has an account. We refer to all the above activities as "login". Next, the application navigates a customer to browse products, search and view the product details. Some customer may leave the site without buying any item (state q_2) or change his/her mind to leave even after selecting some items and having added them in a shopping cart. Otherwise, a customer precedes to payment, which requires customer to enter his/her payment method and relevant financial information. The customer continues to provide shipping information and verifies the order before submitting and completes the transaction (state *T*). Any point after the payment activity and before the submit order activity, the customer may change his/her mind and choose to cancel and leave the shopping application (state q_3).

Each activity in Figure 2 is annotated with a corresponding software component responsible to implement the activity. For example, the *submit order* activity is to be implemented by the *VS* component. Clearly, exit at state *T* signifies business success, while exit at q_1, q_2 and q_3 signify failures. Each of these failing points has different implications on the business. Thus, we consider them separately below.

Quit before enter (q_1): The customer did not like the early login compulsion (especially for a first time visitor who will be asked to open an account), had no time or did not have login information at hand. For business perspectives, one counter-measure to prevent this event is by removing the early login requirement.

Quit before payment (q_2): The customer could not get what he/she was looking for (price, or item), alternatively the inventory could be running out of items. A counter measure for this quit is by marketing research and in creasing inventory or supply of high demand items.

Quit after payment (q_3): The customer did not have sufficient payment, or changed decisions (e.g., after adding shipment cost). One strategy to counter-measure failure at this stage could be to give customers more delivery and shipment options including discounts on large orders.

In general, there are various other reasons for which a customer can quit at any stage e.g. low bandwidth, lack of time, broken internet, etc. However, these considerations have been analyzed in the context of usability and dependability [7, 16]. While our approach can be extended to include them, they are not our focus here.

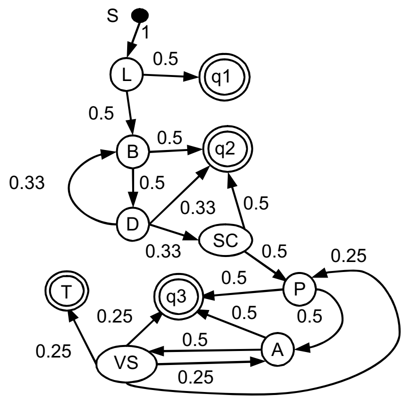


Fig 3. Component graph of Case 1: login before enter.

By applying Steps (1) and (2) of the proposed approach (Section 4.2), we obtain a dependency graph as shown in Figure 3. For simplicity, we consider one scenario and hence Step (2) has no effect. We also assume that all interactions from a component occur equally likely. However, if the knowledge about interaction usage is known, appropriate likelihoods can be assigned. As shown in Figure 2, q_1 has two incoming arrows from the *L* (login) component, which has two outgoing arrows (one from *Validate Login* and the other from *Create Account*) to *Browse Products* (the *B* component). The *L* component has no interactions with other component. Thus, each of the likelihood of usage from *L* to q_1 and from *L* to *B* is 0.5 as shown in Figure 3.

By applying Step (3), we have a set of failure states, $F = \{q_1, q_2, q_3\}$. As shown in Figure 3, since there is only one path from *S* to q_1, the likelihood of usage for a customer to reach q_1 is estimated to be $1 \cdot 0.5$ (via $S \rightarrow L \rightarrow q_1$). The maximum likelihood of failure in q_2 is estimated by considering the likelihoods of each path to reach q_2 (e.g., $S \rightarrow L \rightarrow B \rightarrow q_2$, $S \rightarrow L \rightarrow B \rightarrow D \rightarrow q_2$, $S \rightarrow L \rightarrow B \rightarrow D \rightarrow SC \rightarrow q_2$). It is clear that the maximum likelihood of failure in q_2 is 0.25 (via $S \rightarrow L \rightarrow B \rightarrow q_2$). Similarly, the maximum likelihood of failure in q_3 is estimated as 0.02.

Failure State	Business Consequences	Severity	Value	Max
q_1	Loose Potential Customer	Catastrophic	0.95	0.95
	Loose Potential Order	Marginal	0.5	
q_2	Loose Potential Order	Marginal	0.5	0.5
	Loose Reputation	Marginal	0.5	
q_3	Loose Current Order	Critical	0.75	0.75

Table 1. Severity of different stages of business failures.

Step (4) analyzes consequences of failures. Table 1 gives a summary of failure consequences in business contexts. Severity analysis is organization dependent and in practice quantifying severity can be assessed from past experiences (e.g., using expected loss per year). If such is data is not available, a common practice is to categorize severity into four standard categories: catastrophic, critical, marginal, and minor, each of which is respectively quantified by 0.95, 0.75, 0.50 and 0.25 [7]. For example, as shown in Table 1, quitting after payment in q_3 results in loss of one sale transaction, which is critical but not as severe as loss of opportunity for numerous new customers by quitting in q_1.

Finally, by Step (5) using results from Steps (3) and (4), a total business risk in this design case can be estimated to be $0.5 \cdot 0.95 + 0.25 \cdot 0.5 + 0.02 \cdot 0.75 = 0.62$.

5.2 Case 2: Login Before Payment – Comparison

To test if our approach produces valid results in this online shopping design, we repeat the same steps to a design in Case 2, where login is not required until a customer is ready to make a purchase. Similarly, the application design and its corresponding dependency graph can be constructed for the Case 2. For a fair comparison, the same failure states and their severity analysis remain the same as in Case 1. Thus, the only differences between the two cases are interaction rates that result in different failure likelihood in each failure state. Table 2 shows a comparison of these likelihoods along with business risks in both cases.

Failure State	Failure Likelihood Case 1	Failure Likelihood Case 2
q_1	0.5	0.04
q_2	0.25	0.5
q_3	0.02	0.02
Risk	**0.62**	**0.30**

Table 2. Comparison of failure likelihoods.

Our results show that business risk in Case 1 is about twice as much as that of Case 2. This validates our previous hypothesis, which confirms current practices of enterprise application design for online shopping that delay "login" till payment or make "login" optional.

6. Concluding Remarks

We present a systematic approach to analyze business risks and implications from a high-level enterprise application design. The approach is inherently limited due to lack of information on design characteristics at an early stage. However, it serves the purpose to provide a quick and early rough estimate of business risks by means of heuristics based on characteristics of software design. These risks should only be used as *relative* measures for assisting decision-makings on different design options. The approach is specification independent and general in that it can be applied to any early software design other than enterprise applications. The approach is driven by design models and thus, constitutes a structured process making our analysis non ad-hoc and less error prone. Future work includes study of this approach to the large-scale systems.

References

[1] Adkins, S., P5EE – Perls5 Enterprise Environment. (http://www. officevision.com/pub/p5ee/, February, 2007).

[2] Ambler, S., *The Elements of UML 2.0 Style,* Cambridge Press, 2006.

[3] Barna, P., F. Frasincar and G. Houben. A Workflow-driven Design of Web Information Systems, in *Proc. of Inter. Conf. on Web Engineering*, pp. 321-328, 2006.

[4] Bleistein, S., K. Cox and J. Verner, Integrating Jackson Problem Diagrams with Goal Modeling and Business Process Modeling in e-Business System in Requirements Analysis, in *Proc. of the 11th Asia-Pacific Software Engineering Conference,* pp. 410-417, 2004.

[5] Boehm, B., 1991. Software risk management: Principles and practices, IEEE Software, 8:32-41.

[6] Booch, G., J. Rumbaugh and I. Jacobson, *The Unified Modeling Language User Guide,* Addison-Wesley, 1999.

[7] Cortellessa, V., et al., Model-Based Performance Risk Analysis, *IEEE Transactions on Software Engineering,* 31(1), pp. 3-20, IEEE Computer Society, 2005.

[8] Csertan, G., A. Pataricza, P. Harang, O. Doban, G. Biros, A. Dancsecz, and F. Friedler, BPM Based Robust E-business Application Development, in *Proc. 4th European Dependable Computing Conference,* vol. 2485 of LNCS, pp. 32-43. Springer, 2002.

[9] Fower, M. *Patterns of Enterprise Application Architecture,* Addison-Wesley Professional, 2002.

[10] Hohpe, G. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions,* the Addison-Wesley Signature Series, 2003.

[11] Molina, J., M. Ortin, B. Moros, J. Nicolas, A. Toval, Towards Use Case and Conceptual Models through Business Modeling, in *Proc. of Conf. on Conceptual Model./Entity Relationship Approach,* pp. 281-294, 2000.

[12] Russell, N., van der Aalst, A Hofstede and P. Wohed, On the suitability of UML 2.0 activity diagrams for business process modeling, in *Proc. of the 3rd Asia-Pacific Conference on Conceptual Modeling - Vol 53*, Australian Computer Society, 2006.

[13] Pickett, S., *Enterprise Risk Management: A manager's Journey,* Wiley, 2006.

[14] Singh, I., B. Stearns, M. Johnson, and the Enterprise Team, *Designing Enterprise Applications with the J2EE™ Platform*, Second Edition, Sun Microsystems, Inc., 2002.

[15] Shahrokhi, M. and A. Bernard, Risk assessment/ prevention in industrial design process, in *Proc. IEEE Conf. on Syst., Man and Cyber.* pp. 2592-2598, 2004.

[16] Yacoub, S., B. Cokic and H. Ammar, 1999. Scenario-based Reliability Analysis of Component-based Software, in *Proc. of the 10th International Symposium on Software Reliability Engineering* pp. 22-31.

[17] Zhang, Y., H. Zhu, S. Greenwood and Q. Huo, Quality Modelling for Web-based Information Systems, in *Proc. of 8th IEEE Workshop on Future Trends of Distributed Computing Systems*, 2001.

Incremental effort prediction models in Agile Development using Radial Basis Functions

Raimund Moser[A], Witold Pedrycz[B], Giancarlo Succi[A]
[A]*Free University of Bolzano, Italy,* [B]*University of Alberta, Canada*
rmoser@unibz.it, pedrycz@ee.ualberta.ca, gsucci@unibz.it

Abstract

Despite significant investment in research, the lightweight estimation of development effort is still an unsolved problem in software engineering.

This study proposes a new, lightweight effort estimation model aimed at iterative development environments, as Agile Processes. The model is based on Radial Basis Functions. It is experimented in two semi-industrial projects conducted using a customized version of Extreme Programming (XP). The results are promising and evidence that the proposed model can be developed incrementally and from scratch for new projects without resorting to historical data.

1. Introduction

Effort prediction has always been perceived as a major topic in software engineering. The reason is quite evident: many software projects run out of budget and schedule because of an underestimation of the development effort. Since the pioneering work by Putnam [12], Boehm [4], and Albrecht [2], there have been many attempts to construct prediction models of software cost determination. An overview of current effort estimation techniques, their application in industry, and their drawbacks regarding accuracy and applicability can be found in [8]. Models such as COCOMO II depend quite heavily on many project-specific settings and adjustments, whose impact is difficult to assess, collect, and quantify [11]. What makes the situation even worse, is the fact that in agile processes an effective collection of such metrics and the ensuing tedious calibration of the models are quite unrealistic. As far as we know, no specific models have been developed for agile and iterative development processes. Only a few studies deal with the idea of updating or refining prediction models during project evolution or include the effort of previous development phases as an additional predictor variable. Closest to our work is a recent study by Trendowicz *et al.* [15] who incorporate into a hybrid cost estimation model feedback cycles and the possibility for iterative refinement. MacDonell and Shepperd [9] use project effort of previous phases of development as predictor for a simple regression model and show that it yields better results than expert opinion. However, both studies do not address the peculiarities of agile processes and use a different modeling approach.

Traditional effort estimation works as follows. Some predictor variables are collected or estimated at the beginning of a project and fed into a model. The model, which is usually built upon historical data using similar projects, predicts the total development effort. While this approach is reasonable for traditional development processes where common predictor variables such as function points, software size, design specifications, etc. are known at the beginning of a project and typically do not change too much throughout the overall project this is not the case for agile development processes. In agile development, a project is realized in iterations and requirements usually change from one iteration to the next. At the end of each iteration, developers release the software to the customer who will eventually require new features, and change or removal of already implemented functionalities. Therefore, standard predictor variables proposed in the literature, in particular the ones derived from design documents, are only known at the beginning of the next development iteration and not a priori for the whole project.

Being cognizant of the existing challenges as outlined above, the key objectives of our study are outlined as follows:

- We propose a new type of prediction model, suited to iterative processes and referred to as incremental model.
- We carry out a thorough experimental validation of a specific implementation of the incremental model using Radial Basis Functions (RBF).

We aim at answering the following research question: In agile, iterative software development processes, are incremental effort prediction models efficient for iterative effort prediction and do they perform better than traditional models?

Our proposed incremental approach addresses a crucial point in effort estimation as in general, managers tend to be over-optimistic and over-confident in estimation and scheduling, and are normally reluctant to move from initial estimates and schedules when progress slips [10]. An estimate should be dynamic – as the project progresses more information becomes available.

The remainder of the paper is organized as follows. In Section 2, we propose and elaborate on the concept of incremental prediction models. Section 3 discusses the use of RBF models for effort prediction. In Section 4, we present a case study. Finally, in Section 5 we draw the conclusions.

2. Incremental prediction models

There are crucial issues in the development and utilization of traditional, monolithic effort estimation models as described earlier that have to be addressed. First, the choice of the predictor variables is highly demanding as at the beginning of the project we may not know which variables of the project could prove to be good effort estimators. While we might be tempted to collect a lot of variables to compensate, it could be time consuming, costly, and at the end lead to overly complicated models. Second, in the construction of global models we rely on historical data or/and expert opinion. Given the unstable and highly non-stationary environment of software development, this may lead to models whose predictive capabilities are questionable. Moreover, the software industry is moving in a direction where projects are not completed but constantly evolving with new updates and deliveries in response to market demands. In such a scenario it is not obvious when to freeze the project for model building purposes.

Agile software development brings another problem for traditional effort estimation. Predictor variables usually are not known at the beginning of a project, but become available at the beginning of a new iteration. Under these circumstances a long-term model of effort estimation that naturally relies on information available at the beginning of a project seems to be of limited applicability.

Taking into account the limitations pointed out above, we assume a different development position by focusing on the incremental mode of model development.

The main idea of an incremental prediction model is that it is built **after each iteration** instead of at the end of the project. Thus, it is able to adapt to any changes during development in a much smoother way than a global model. Moreover, we endow the incremental model with a dynamic character by using the effort determined in the previous iterations as an additional input variable. The incremental model operates only for iterative effort prediction as it cannot be used to predict total development effort at the beginning of a project.

The essence of the incremental model can be explained as follows.

- Model building: At the **end of each iteration** a new model is built using as input the predictor values for that iteration and the development effort of previous iterations.
- Iterative effort prediction: At the **beginning of a new iteration** predictor variables are collected and fed together with past effort into the newly built incremental model. The output of the model produces an estimation of the total development effort from project start to the end of this iteration.

Clearly, with an incremental effort prediction model a company cannot estimate the total development effort at the beginning of a project. This may be an important issue for a company that is trying to decide whether or not they want to take on a given software project.

3. An implementation using Radial Basis Functions (RBF) and design metrics

Radial Basis Functions (RBF) provide a flexible way to generalize linear regression functions and show some properties, which make them suitable for modeling software engineering data. An RBF network functions as follows: First, input data are mapped in a non-linear way using basis functions (we use Gaussians); then, the final response variable is built as a linear combination of the output of the basis functions with the network's weight vector. RBF's have been used for effort estimation showing very promising results [13]. In general, the performance of RBF models depends highly on the chosen network architecture (number of layers, number of neurons, activation function, number of receptive fields, spread etc.). While there is no general theory behind the structural optimization of the topology of the networks, they are developed as a result of some trial and error process. For each set of parameters, that specify the model, we compute the leave-one-out cross-validation error [3]. We keep the set that produces the lowest error and this topology of the network is deemed optimal. We start with one receptive field and add one at a time until the cross-validation error stops decreasing. We repeat this procedure for a range of spread parameters and keep the spread and number of receptive fields that return the absolute smallest cross-validation error.

As predictor variables we use the Chidamber and Kemerer (CK) set of object-oriented design metrics [5]. The CK metrics have some nice properties, which make them in particular attractive for the kind of prediction model we propose:

- They are widely known by practitioners and in the research community and have been validated by other researchers.
- For the purpose of model building the CK metrics can be extracted automatically from source code.

We do not use all 6 CK metrics as predictors but exclude the NOC (number of children) and LCOM (lack of cohesion of methods) metrics. We exclude NOC because in both projects it is almost 0 in all classes and hence does not contribute to the variation of effort data. As for LCOM several researchers have questioned its meaning and the way it is defined by Chidamber and Kemerer [6].

4. Case study

The case study concerns two commercial software projects – we refer to them as project *A* and project *B* - developed at VTT Technical Research Centre of Finland in Oulu, Finland. The programming language used in both projects was Java. Project *A* delivered a production monitoring application for mobile, Java enabled devices. Project *B* delivered a project management tool for agile projects. For both projects the development process followed a tailored version of Extreme Programming practices [1], in project *A* two pairs of programmers (four people) and in project *B* three pairs of programmers (six people) have worked for a total of eight weeks. The projects were divided into five iterations, starting with a 1-week iteration, followed by three 2-week iterations, with the project concluding in a final 1-week iteration.

4.1. Data collection process and data

For both case studies we used our in-house developed tool PROM [14] for automatic and non-invasive data collection. We adopted the following data collection procedure. Every day at midnight various source code metrics (among them are the CK metrics) are extracted from a code repository. A plug-in for Eclipse (the IDE used by developers) collects automatically the time spent for coding on individual classes and methods.

For project *A* the total coding effort recorded by the PROM tool is about 305 hours. Project *A* has 1776 lines of code (counted as Java statements in the source code) divided in 30 classes. Project *B* has 3426 lines of code and 52 classes. The total coding effort for project *B* is about 664 hours. Due to space constraints we do not report descriptive statistics and box-plots for the different variables: They evidence that data have a few outliers and are highly skewed - two conditions, which would be problematic for ordinary least square regression models but are mitigated by RBF networks.

4.2. Results

In Table 1 we present the results for effort prediction using an incremental RBF model. For assessing prediction accuracy we report 4 different criteria – used exclusively none of them is reliable [7]: Two give the error relative to the true value (MRE) respective the estimate (MER); one criterion is the usual standard deviation (SD). Finally, the last criterion is the percentage of predictions, which have a relative error less than 25% (PRED(25%)). In general, accuracy of predicting total effort per iteration is higher than predicting effort for single classes. This can be explained by the fact that by summing up all classes errors due to underestimation respective overestimation annihilate in part.

Table 1: Prediction of effort per class and total effort per iteration with the use of incremental RBF models.

It.	Median MRE	Median MER	SD	PRED 25%	Total MRE
Project *A*					
3	65%	59%	2.6	0%	54%
4	42%	66%	2.0	21%	33%
5	14%	14%	0.6	73%	6%
Project *B*					
3	89%	98%	3.1	9%	81%
4	55%	111%	3.3	12%	54%
5	30%	37%	1.5	40%	16%

The prediction accuracy improves from iteration to iteration indicating that the incremental model stabilizes during development and prediction errors decrease and converge. Already for iteration 5 the model provides, for software engineering standards, accurate predictions (MRE around or less than 25%).

In order to compare our proposed incremental approach with the traditional, monolithic model we proceed as follows. We combine both data sets (from project *A* and project *B*) for building an RBF model using as predictors CK metrics extracted from source code at project conclusion and as dependent variable total coding effort. Then, we apply such model for predicting development effort for both projects in an iterative way. The result is that the predictions provided by the traditional model are in all cases, both at a class and system level, less accurate, sometimes even of the order of magnitude, than the ones obtained by the incremental model. For project *A* the average relative error for the global model is around 280%, while for project *B* it is around 295%. These numbers evidence that by using traditional models for iterative effort prediction we do not get any useful results for our projects.

Overall, the results enable us to answer our research question. We can state that for the two projects in this study the incremental model provides accurate effort estimations for later development iterations and is superior to the traditional, monolithic model.

5. Conclusions

In this study, we propose a new approach for lightweight, iterative effort prediction. We have identified a number of reasons why monolithic prediction models are limited when dealing with agile software development. We propose an incremental approach and apply it - using RBF networks - to two agile, semi-industrial development projects. The results are promising. Incremental models are stable and convergent in the sense that their prediction error decreases from iteration to iteration. They can be used from the start of development, without the need for historical data, and improve prediction accuracy throughout project evolution due to their iterative nature.

References

[1] P. Abrahamsson, A. Hanhineva, H. Hulkko, T. Ihme, J. Jäälinoja, M. Korkala, J. Koskela, P. Kyllönen, and O. Salo, "Mobile-D: An Agile Approach for Mobile Application Development", *Proceedings of the 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA'04*, Vancouver, British Columbia, Canada, 2004.

[2] A.J. Albrecht, J.E. Gaffney, "Software function, source lines of code, and development effort prediction", *IEEE Transactions on Software Engineering*, **9**(6): 639-648, 1983.

[3] C.M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, UK, 1994.

[4] B.W. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.

[5] S. Chidamber, C.F. Kemerer, "A metrics suite for object-oriented design", *IEEE Transactions on Software Engineering*, **20**(6): 476-493, 1994.

[6] S. Counsell, S. Swift, J. Crampton, "The interpretation and utility of three cohesion metrics for object-oriented design", *ACM Trans. Softw. Eng. Methodol.*, **15**(2): 123-149, 2006.

[7] T. Foss, I. Myrtveit, E. and Stensrud, "A comparison of LAD and OLS Regression for Effort Prediction of Software Projects," *Proc. 12th European Software Control and Metrics Conf.*, 9-15, 2001.

[8] M. Jørgensen, and M. Shepperd, "A Systematic Review of Software Development Cost Estimation Studies Document Actions", *IEEE Transactions on Software Engineering*, **33**(1): 33-53, 2007.

[9] S. MacDonell, M.J. Shepperd, "Using Prior-Phase Effort Records for Re-estimation During Software Projects", *Proceedings of the Ninth International Software Metrics Symposium (METRICS'03)*, Sydney, Australia, 2003.

[10] S. McConnell, "Avoiding classic mistakes", *IEEE Software*, 1996, pp. 111-112.

[11] T. Menzies, D. Port, Z. Chen, J. Hihn, S. Stukes, "Validation methods for calibrating software effort models", *Proceedings of the 27th International Conference on Software Engineering*, St. Louis, MO, USA, 2005.

[12] L.H.A. Putnam, "A general empirical solution to the macro software sizing and estimation problem", *IEEE Transactions on Software Engineering*, **4**(4): 345-381, 1978.

[13] M. Shin and A.L. Goel, "Empirical Data Modeling in Software Engineering Using Radial Basis Functions", *IEEE Transactions on Software Engineering*, **26**(6): 567-576, 2000.

[14] A. Sillitti, A. Janes, G. Succi, T. Vernazza, "Collecting, Integrating and Analyzing Software Metrics and Personal Software Process Data", *Proceedings of the 29th EUROMICRO*, Antalya, Turkey, 2003.

[15] A. Trendowicz, J. Heidrich, J. Münch, Y. Ishigai, K. Yokoyama, N. Kikuchi, "Development of a hybrid cost estimation model in an iterative manner," *Proceeding of the 28th International Conference on Software Engineering*, Shanghai, China, 2006.

BASS: Business Application Support through Software Services

Mateus B. Costa[*,†]
[*]Federal Center of Technological Education of Espírito Santo
Serra, ES, Brazil
mcosta@dcc.ufmg.br

Rodolfo F. Resende[†]
[†]Federal University of Minas Gerais
Department of Computer Science
Belo Horizonte, MG, Brazil
rodolfo@dcc.ufmg.br

Marcelo V. Segatto[‡]
[‡]Federal University of Espírito Santo
Department of Electrical Engineering
Vitoria, ES, Brazil
segatto@ele.ufes.br

Eduardo F. Nakamura[⋆]
[⋆]FUCAPI – Research and Technological Innovation Center
Manaus, AM, Brazil
eduardo.nakamura@fucapi.br

Nahur Fonseca[ψ]
[ψ]Boston University
Department of Computer Science
Boston, MA, USA
nahur@cs.bu.edu

Abstract

In this paper, we introduce a software framework called Business Application Support through Software Services - BASS. BASS supports the development of information system applications. BASS support aims at decoupling the business logic description of the applications from the supporting implementation, e.g. software services. BASS also allows us to define service access elements without programming activities. By using the BASS framework, target applications can dynamically select and specify service invocations according to the execution of the supported business processes.

1. Introduction

Information systems integration, driven from increasing demands for collaborative activities, must consider important interoperability aspects and, at the same time, should not impose hard technological or business constraints [22]. Therefore information systems development must be supported by software solutions considering such interoperability aspects. Service Oriented Computing (SOC) is considered a major paradigm to support their development [19].

In the SOC paradigm, a software service is an element with functions available to other networked software elements. A software service can also be seen as a software element that encapsulates one or more business functions [8]. An application supporting an information system can play the role of service provider or service client. A client application can use a business function available through a service once guaranteed the integration at the message exchange level, at the application architecture level, and by the supported business processes [13].

Web Services are one of the important sets of technologies that implements the software service concept. Web Services technologies provide a high degree of interoperability at the message exchange level and among different development platforms, facilitating the integration at these levels [8]. However, the interoperability aspects that remain at business process level must be considered in the development of the involved applications.

Applications that provide services are developed to enable run time service discovery [21] and composition [16] to be performed by unknown client applications. Client applications should therefore deal with service discovering, selection and execution driven by their business processes, Business Transaction support [6, 18], and several other aspects, e.g., vocabulary mediation [9].

In this work, we present the Business Application Support through Software Services - BASS, a software framework [11], for developing the software infrastructure for information system integration. BASS provides an Application Programming Interface (API) that simplifies the mapping of the business process level onto the application's architecture level, allows the definition of the application business logic to be independent from the software services aspects and encapsulates the complexity of dealing with the software services and the service's business functions idiosyncrasies in the application development. The use of BASS also permits the definition of service access aspects without the need of program encoding. One important contribution of BASS is to show that a service can be treated during software development as a logical and physical element absent from the conceptual models.

The remainder of this paper is organized as follows. In Section 2, we discuss the aspects of applications, which are client of software services, that can be developed by using

BASS. In Section 2 we also introduce our running example. In Section 3 we discuss the application development using BASS. In Section 4, we present some related work. Section 5 presents future directions and concludes the paper.

2. Client Applications

Software services can be used in different application domains, such as scientific applications and information systems for E-Commerce. The development of BASS is focused on the information system domain. To illustrate the use of BASS we adopt as an example the development of a simple Business-to-Business application to manage orders among a client and several suppliers. Despite its simplicity this example follows some ideas presented by Papazoglou [18]. The Order Management includes the following functionalities:

- **Create Order.** The client creates an order following the main flow:
  1. Create an order; 2. Place the order.
- **Verify order.** The client verifies an order following the main flow:
  1. Get the order ID; 2. Retrieve the order; 3. Present the order.
- **Cancel an order.** The client cancels an order following the main flow:
  1. Get the order ID; 2. Cancel the order.
- **Update an order.** The client updates an order following the main flow:
  1. Get the order ID; 2. Retrieve the order; 3. Update the order data; 4 Re-place the order.
- **List a set of orders.** The client queries the orders following the main flow:
  1. Get the order list selection criterion; 2. Retrieve the order list based on the selection criterion; 3. Present the order list.

Herein we consider a software development process based on the Unified Process [15]. The Unified Process prescribes the use of Object Oriented Modeling and Design, UML notation and Use Case based development. In order to represent the functional requirements of the Order Management, its functionalities can be translated in terms of an use case diagram.

One aspect of using the Unified Process is the development of: a) an analysis model, represented by diagrams that include the elements of the problem domain; b) a design model, with diagrams representing, in high level, the elements of the solution domain for the chosen development platform; and c) an implementation model that details the solution described by the design model. Here we consider a process based on the Unified Process and customized for the BASS framework, but we focus on the design model aspects.

A preliminar design model for Order Management could be represented in terms of local software elements, without

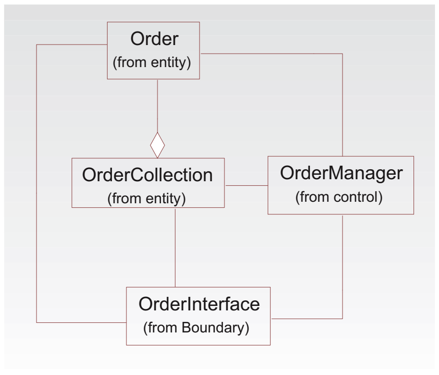


Figure 1. Classes involved in the Order Management.

considering software Services. Figure 1 presents a possible class diagram of the Order Management design model including these elements.

To support the Order functionality, the **OrderManager** class can implement the operation **saveOrder(Order order)**. Similarly, the remaining functionalities can be supported by operations that form together a set of operations much like the CRUD (Create, Retrieve, Update, and Delete) operations set [5]. Such operations would be: **retrieveOrder(OrderId id)**, **listOrders(OrderCriterion criterion)**, **saveOrder(Order order)**, **deleteOrder(OrderId id)**, and **updateOrder(Order order)**.

We refer to this set as RLSDU operations (**R**etrieve, **L**ist, **S**ave (create), **D**elete, and **U**pdate). In a local implementation, such operations can be supported by a persistence layer [1, 14] that encapsulates the adopted storage mechanism and allow these operations to be performed exclusively by means of operations involving application objects.

To illustrate how these RLSDU operations are implemented in terms of software services, we consider herein that the orders are sent the suppliers' information systems through software services.

In the development based on software services, services are usually accessed by elements, called *stubs*, and local representations of the *data types* defined by the services. The majority of Web service platforms provide a tool to automatically generate these elements (stubs and data types) from the service interface descriptions. This style allows client applications to be developed using the operations available in the stubs and abstracting the adopted service access platform.

Let us consider the layered architectural style [20] where we define two layers: an Application and an Integration layer. In the application layer, we have the application local elements and representations of elements external to the application. In the integration layer, we have elements to support the service access.

Figure 2 presents a diagram containing the Application and the Integration layers represented as packages. The package representing the Application Layer has the sub-packages Entity, Control and Boundary as derived from the Analysis Model following the prescription of the Unified Processs. In the Application layer we have also the stubs corresponding to the services of each supplier. The integration Layer presents a high level API built with APIs such as the one defined by Apache AXIS[2].

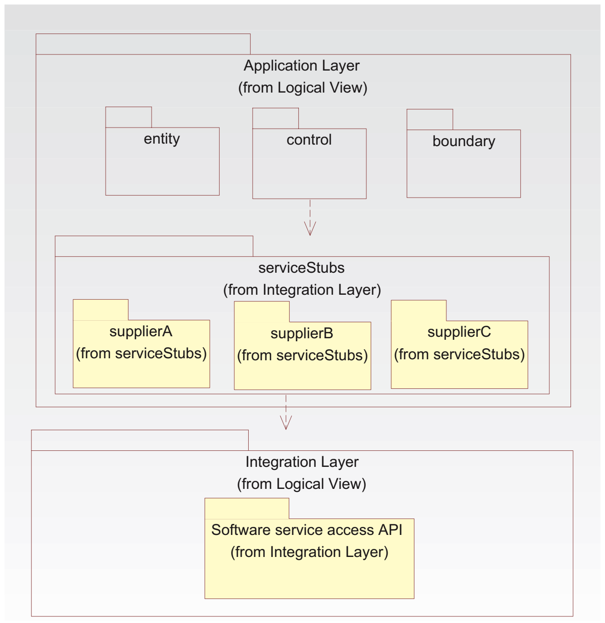


Figure 2. Order Management Layered Architecture

Figure 3 presents the class diagram for the Order Management considering the use of stubs. In this example, we consider one client and three suppliers. Each supplier has its own stub and its data type set. To represent an order, **SupplierA** uses the Invoice data type, **SupplierB** uses the Order data type, and **SupplierC** uses the Request data type. All these data types differ, in terms of attributes, from the Order data type defined in the client application.

The implementation of the **saveOrder(Order order)** and the other RLSDU operations imposes specific logical elements (stubs and data types) dependencies to represent the business logic. These dependencies introduce complexity in the development of the business process driven service selection and execution support. This approach also introduces difficulties for application change management, scalability and flexibility [12].

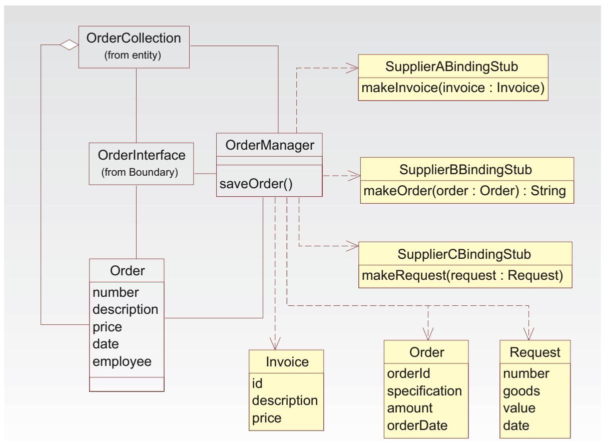


Figure 3. Class Diagram for the Order Management Application Layer.

3. Using BASS to develop applications

Similarly to persistence layers tools, BASS provides an API that allows the development of the Application layer design model without logical and physical dependencies of the services' access. The RLSDU operations are the substitutes for the direct service access.

In our example, an order is represented by an instance of a class called Order. The RLSDU operations deals with instances of classes such as Order. Classes such as Order are referred to as a proxy entity. BASS implements RLSDU operations for proxy entities. Each proxy entity has a corresponding external representation and the set of values associated to external representations define an external state. The semantics of these operations is specified as follows:

- **Retrieve:** retrieves the entity proxy external state. This external state is retrieved and converted into an entity proxy instance;
- **List:** retrieves the external state of a set of entity proxy instances;
- **Save:** Saves the state of an entity proxy instance into an external representation. The representation is created if it still does not exists;
- **Delete:** Removes the external representation associated with an entity proxy instance;
- **Update:** Updates the state of an existing entity proxy instance based on the state of its corresponding external representation.

The Update operation is provided, except for inexistent objects, as a convenient option to the pair retrieve and save.

The semantics for the RLSDU operations does not define the concrete actions that will be performed out of the application bounds. As we will see later, this task is part of the configuration of BASS.

In the context of a layered architecture with an Application layer and Integration layer, by using BASS, the development of the Application layer does not depend on the stubs and data types used to access software services. Figure 4 illustrates this architecture and also shows the dependency relations among the elements of the Application and Integration layers. The elements of the Integration layer referred to in the Application layer are the **Session** and **OCLExpression** classes.

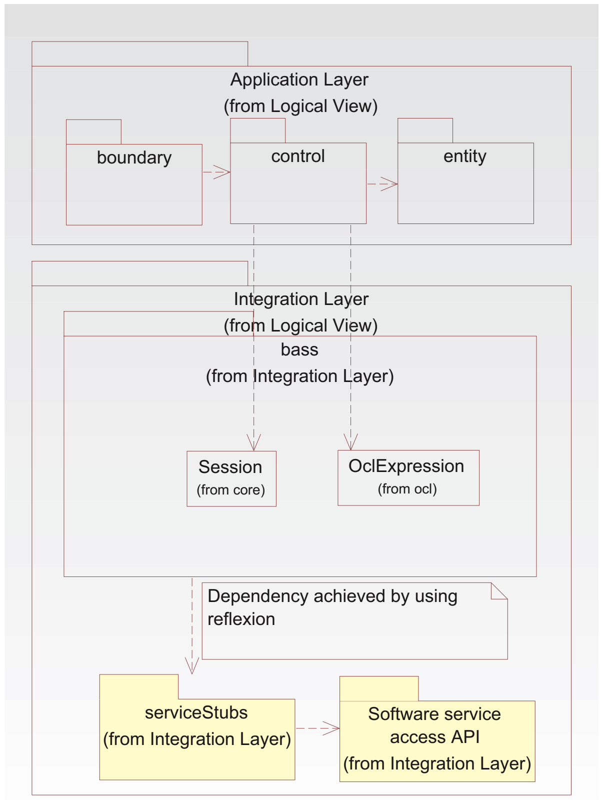


Figure 4. A Layered Architecture using BASS.

Figure 5 depicts the elements of the Application layer and the elements of BASS used to implement the **Order Management**. The RLSDU operations will be performed taking an Order proxy entity (a class or an instance, depending on the operation) and a set of business rules. The Session class implements the generic RLSDU operations. The **OCLExpression** class supports the specification of the business rules that will be used to perform those operations. The services, services' operations, and operations' arguments used to perform the RLSDU operations will be defined based on the class or object that executes the operation and on the business rules expressed using the Object Constraint Language (OCL) [17].

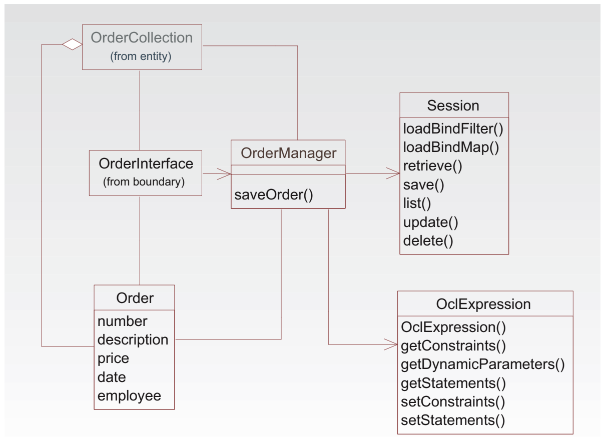


Figure 5. Class Diagram for the Order Management Application Layer by using BASS.

To save an order, an OrderManager object will perform the following steps:

1. Instantiate an object of *Order* class and set its values;
2. Choose the target supplier;
3. Specify the supplier in terms of an OCL expression;
4. Invoke the save operation by using a Session object and by passing the Order and OCLExpression objects as arguments.

The supplier choice is a decision made during the business process execution and can be guided by the user interaction with the objects of the **OrderInterface** class. This aspect forces the software service interactions to be automatically guided by the business process of the client application. Another task performed by BASS is the vocabulary mediation. As we can see in the example, the application Order class differs, in term of attributes, from the equivalent suppliers' classes. Mediation between these vocabularies is automatically performed by BASS.

The configuration of the Integration layer involves two main tasks:

Specify the bindings of every proxy entity: In the context of this work, **binding** is an implementation of an abstract data type that encapsulates an association between an entity proxy and a software service and the details of such an association.

Specify the application vocabulary: The application vocabulary includes the definition of a set of terms and mor-

```
<!-- SAVE BindEntryCollection --> <save>
  <bindEntry className= "SupplierABindingStub">
    <operations>
      <operation name="makeInvoice"
        returnType="" MappedToField="">
       <parameters>
         <parameter name="invoice"
         paramType="Invoice" category="javabean" >
         </parameter>
       </parameters>
     </operation>
   </operations>
 </bindEntry>
 <bindEntry  className="SupplierBBindingStub">
 <!--same structure of the bind entries above-->
 </bindEntry>
 <bindEntry className="SupplierCBindingStub">
<!--same structure of the bind entry above-->
 </bindEntry>
</save>
```

Figure 6. Excerpt of a binding file related to class Order showing the {Order,save} binding.

phological relationships among these terms, such as synonymity.

Every pair *{proxy entity, RLSDU operation}* has a set of bindings. For instance, the pair *{Order, save}*, has a **binding** with **SupplierA**, **SupplierB** and **SupplierC**. In our current implementation of BASS, the set of bindings related to an entity proxy is represented by an XML file that we call *binding file*. Figure 6 presents the set of bindings corresponding to the *{Order, save}* pair.

The specification mechanism of the vocabulary used by the application is a framework hotspot [11]. A hotspot is a customizable (programmable) framework element. This hotspot allows the incorporation of different mechanisms for defining and analyzing the application vocabulary and mediating the vocabularies used by the application and the software services. In our current implementation, the vocabulary mediation mechanism is based on lists composed of: a) one local term, used for conceptual modeling of the client application and b) a set of synonyms from data types and parameters used by the software services. Figure 7 illustrates terms used in our Order Management example.

```
order invoice request
number id orderId
description specification
price amount value
date orderDate
empployee
```

Figure 7. Order Management vocabulary.

3.1. Observed Aspects in the Development using BASS

BASS was defined together with a method for designing and implementing applications that are clients of software services. In this method, we observe the following features:

1. The modularization of the application development by separating the design and implementation of software service interactions from the remaining parts. This aspect can be observed in the separation of the design of the Order Management in an Application layer and an Integration layer. Among the benefits of such a separation, we can observe the simplification not only of the work division and testing but also a simplification of the mapping of the application analysis model onto its correspondent design model without the need to consider the aspects of the software services being used.

2. The specification of the service interactions is done without programming activities. In reality, instead of procedural code programming, BASS specifies a type of declarative programming using configuration files. As we showed, the Integration layer is configured by using binding files and the vocabulary definitions. Such configuration, can be aided by a tool that automatically gets the stubs and data types of the associated services, and simplifies the definition of the binding files (e.g., by using a graphical interface). This tool can also aid the vocabulary entry generation based on the business rules and on the application proxy entities, stubs, and data types.

3. The service selection and execution guided by business processes. As we observed, the service selection and execution occurs transparently based on instances of business rules and processes executed in the application layer. For instance, in the execution of an order placement, the determination of which service should be executed is done through user interactions. Based on that choice, a business rule instance containing the supplier specification is created, and BASS determines the bind entry associated to the supplier (see Figure 6).

4. Related Work

Software development tools and platforms that support Software Services, such as .NET and J2EE, have been used within development models in which computations that use Web services have been explicitly programmed in the application modules by means of Stubs or remote procedure calls. However, some of the problems resulting from such an approach have been pointed out in the literature. In the remainder of this section we briefly discuss some approaches to these problems.

Eberhart [10] proposes the so called Web Service Description Framework (WSDF) that handles the aspects of the dynamics of service-oriented applications by using an approach based on Web Semantics and ontology. The main purpose of the WSDF is to allow a client application to access a service without knowing its description.

Verheecke and Cibrï¿½n [3] address the problem of the applications' lack of flexibility caused by the explicit coding of computations, in client applications, that use Web Services. The authors propose an intermediate layer between applications and services called Web Service Management Layer (WSML). The WSML allows the use of Web Services without hard coding the invocation of the services by using Aspect Oriented Programming.

Baligand and Monfort [4] present a similar approach, but they focus on including policies for specifying non-functional requirements also in the the description and in the invocation of services.

We have also experimented, in early work, the use of Aspects for dealing with software services selection and invocation [7]. Our solutions based on Aspects introduced elements nonintuitive to the business logic definition, whereas, the use of Object Oriented frameworks were more appropriate in creating the abstraction view of the integration layer in the application development.

5. Conclusion and Future Work

The use of technologies based on software services aids the development of applications for different domains, such as E-Commerce, Collaborative Work, and Information Distribution. However, the difficulties for this development are not negligible.

In this work we presented BASS, a software framework to support the development of applications that are clients of software services. The use of BASS decouples the business application aspects from the implementation details of software services. The definition of the services related to the application are specified in a customizable way, without the need of programming activities. BASS enables the business process-driven software selection and execution, and eases the services run time configuration.

As future work, we plan to extend the current version of BASS with long-running transactions [18]. The support of such transactions aims at allowing the definition of the treatment of possible transaction variations. Other future work includes the definition of a GUI-based configuration tool for BASS and a customization of a software process including BASS. Our current results are very promising and we believe that the benefits obtained by using BASS are similar to the benefits obtained by using a persistence layer instead of hard-coding SQL commands in an application. Nevertheless, we are designing an experiment that will characterize the benefits of using BASS instead of hard-coding a network solution.

References

[1] S. Ambler. Mapping Objects to Relational Databases, 2000. [Online] Available: http://www.AmbySoft.com/ mappingObjects.pdf.

[2] APACHE. Apache Axis, 2006. [Online] Available: http://ws.apache.org/axis/.

[3] M. A. C. B. Verheecke and V. Jonckers. AOP for Dynamic Configuration and Management of Web Services. In *ICWS'03)*, September 2003.

[4] F. Baligand and V. Monfort. A Concrete Solution for Web Services Adaptability using Policies and Aspects. In *ICSOC '04*, pages 134–142, New York, NY, USA, 2004. ACM Press.

[5] A. Cockburn. *Writing Effective Use Cases*. Addison Wesley, 2001.

[6] M. B. Costa, R. F. Resende, M. F. Alves, and M. V. Segatto. Business-to-Business Transaction Modeling and WWW Support. In *BPM'2*, pages 132–147. LNCS - Springer Verlag, June 2004.

[7] M. B. Costa, R. F. Resende, P. S. Neto, and M. H. F. Alves. Utilização de Aspectos no Desenvolvimento de Aplicações baseadas em Serviços Web. In *WASP'04*, 2004.

[8] F. Curbera, W. Nagy, and S. Weerawarana. Web services: Why and how. In *OOPSLA'01: Proceedings of the Workshop on Object-Oriented Web Services.* ACM, 2001.

[9] A. Dey, J. Mankoff, G. Abowd, and S. Carter. Distributed Mediation of Ambiguous Context in Aware Environments. In *UIST '02*, pages 121–130, New York, NY, USA, 2002. ACM Press.

[10] A. Eberhart. Towards Universal Web Service Clients. In *Proceedings of the Euroweb*, 2002.

[11] M. E. Fayad, D. C. Schmidt, and R. E. Johnson. *Building Application Frameworks: Object-oriented Foundations of Framework Design*. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[12] R. B. Grady. *Practical Software Metrics for Project Management and Process Improvement*. Prentice-Hall Inc, Upper Saddle River, NJ, 1 edition, 1992.

[13] W. Hasselbring. Information system integration. *Commun. ACM*, 43(6):32–38, 2000.

[14] Hibernate. Relational Persistence for Java and .Net, 2006. [Online] Available: http://www.hibernate.org/.

[15] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process*. Addison Wesley, 1998.

[16] N. Milanovic and M. Malek. Current Solutions for Web Service Composition. *IEEE I. Computing*, 8(6):51–59, 2004.

[17] OMG. Object Constraint Language Description, 2003.

[18] M. P. Papazoglou. Web Services and Business Transactions. *World Wide Web: Internet and Web Information Systems*, 6(1):49–91, 2003.

[19] M. P. Papazoglou and D. Georgakopoulos. Service-Oriented Computing: Introduction. *Communications of the ACM*, 46(10):24–28, October 2003.

[20] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall Publishing, 1996.

[21] G. Spanoudakis, A. Zisman, and A. Kozlenkov. A Service Discovery Framework for Service Centric Systems. In *SCC '05*, pages 251–259, Washington, DC, USA, 2005. IEEE Computer Society.

[22] S. Zang, A. Hofer, and O. Adam. Cross-enterprise business process management architecture- methods and tools for flexible collaboration. In *OTM Workshops*, volume 3292 of *Lecture Notes in Computer Science*, pages 483–494. Springer, 2004.

Using Model-Driven Pattern Matching to Derive Functionalities in Models

García-Rodríguez de Guzmán, I., Polo, M. and Piattini, M.

Abstract— **Today, software engineering is evolving from source code to the model realm. As objects have been considered "atomic units" in software development and maintenance, models are becoming first order citizens in the well-known MDA paradigm. New languages and metamodel families are appearing to support the management of models and the new capabilities arising from this technological evolution. In this paper we present the MDPEM (Model-Driven Pattern Matching) technique and a possible application for models.**

***Index Terms*— MDA, Model instrumentation, Model-Driven Pattern Matching, OCL, QVT**

I. Introduction

Software evolution is turning classic artefacts into models as a first order element in the development process. Current development paradigms are going through a change from object orientation to model-driven trends [1]. This evolution is partially supported by many proposals such as standard languages and metamodels. Perhaps one of the most popular initiative is OMG (*Object Management Group*), consisting of UML2 (*Unified Modelling Language*) [2], OCL2 (*Object Constraint Language*) [3], MOF2 (*Meta-Object Facility*) [4], CWM (*Common Warehouse Specification*) [5] and the recently appeared KDM (*Knowledge Discovery Metamodel*) [6] (intended to support the Architecture-Driven Modernization process [7]).

Since languages such as Java, SmallTalk and C++ met the new requirements when object orientation appeared, other kinds of languages are currently appearing to fulfil the new requirements of the MDA (*Model-Driven Architecture*) [8] paradigm. These languages are capable of managing models and metamodels as other languages, such as Java or C++, deal with objects and classes.

The MDA philosophy is creating new opportunities for software engineering. For example, software development can be supported by means of models [9, 10], reengineering for software modernization [11], system integration [12], etc.

Ignacio García-Rodriguez de Guzmán is with the Alarcos Research Group. Information Systems and Technologies Department, UCLM-SOLUZIONA Research and Development Institute. University of Castilla-La Mancha, Ciudad Real, Spain. (e-mail: Ignacio.GRodriguez@uclm.es).

Macario Polo is with the Alarcos Research Group. Information Systems and Technologies Department, UCLM-SOLUZIONA Research and Development Institute. University of Castilla-La Mancha, Ciudad Real, Spain. (e-mail: Coral.Calero@uclm.es).

Mario Piattini is with the Alarcos Research Group. Information Systems and Technologies Department, UCLM-SOLUZIONA Research and Development Institute. University of Castilla-La Mancha, Ciudad Real, Spain. (e-mail: Mario.Piattini@uclm.es).

One of the most popular (and one of the most ambitious bets) to deal with models is QVT (Query/View/Transformations) [13]. This language, proposed by the OMG, is intended to perform operations with models. Using QVT, it is possible to throw queries against models, create views from models and of course, carry out transformations among models. One of the most common operations in QVT is pattern matching. For any operation, QVT expresses models as searching patterns. The QVT specification [13] establishes in which context pattern matching is used, but is does not make clear what to do when it is only necessary to look for occurrences of a given pattern.

Computer science has used pattern matching in many domains, such as lexical and syntactical analyzers, data mining, information retrieval, etc. But in most of them, pattern matching is carried out against text, files or other media. Our proposal is to use the latest advances in model technology to facilitate the application of the well-known technique of pattern matching.

This paper is organized as follows: Section II gives an overview of QVT and some related works; Section III introduces the concept of *MDPEM*; Section IV depicts what *MDPEM* could be useful for; Section V tackles a possible application of MDPEM in the aggregation of services to a system; and Section VI gives some conclusions and outlines some future lines of work.

II. Background

A. About QVT

QVT is an initiative of the OMG. In 2002, it was published the QVT RFP. One year later in March 2003, the QVT-Partners published the "Initial submission for MOF 2.0 Query/Views/Transformations RFP" [14]. In November 2003, QVT-Partners published the "Revised submission for MOF 2.0 Query / Views / Transformations RFP" [15]. This specification showed a more complete specification along with the declarative and imperative QVT's languages. In November 2005, OMG published the "MOF QVT Final Adopted Specification" [13].

In spite of the popularization of this language, there is no available any QVT engine implementing the declarative QVT language (up to now). Some projects in the academic world [16] are now underway. There are other initiatives such as the *MODELWARE QVT Tool* [17], based on the *QVT Operational* language, the other sublanguage of QVT.

B. *Pattern Matching*

Traditionally, patterns have been used for many purposes. For example, in some works [18, 19], patterns have constituted the cornerstone of the migration process from one kind of system to another. In other studies [20], the authors use a pattern-oriented language to integrate databases. Additionally, patterns have been used to reduce the inherent complexities that arise when applications must be connected to databases [20-22].

Patterns have been widely used to reengineer legacy systems [23], and particularly, patterns are very useful in the reverse engineering stage [24-26]. The well-known patterns of Gamma [27] have been used not only in the system design stage, but also with the aim of refactoring or restructuring legacy systems.

III. MODEL-DRIVEN PATTERN MATCHING

A. *What does MDPEM consist of?*

Model-Driven Pattern Matching is simply a traditional concept which has evolved together with software trends. The basic idea behind pattern matching is to find occurrences of a given pattern in a data set. Thus, MDPEM uses models to specify both the patterns and the target data set.

According to [28], "*the essential idea behind pattern matching is to allow the succinct expression of complex constraints in an input data type; data which matches the pattern is then picked out and returned to the invoker*". In the MDPEM context, the pattern model specifies the complex constraints and the target model represents the data.

B. *A framework for MDPEM*

As noted above, this paper focuses on the *MDPEM* concept in the QVT language; thus the developed framework to perform MDPEM is based on this model-oriented language.

The QVT language uses the concept of pattern matching in every action that its constructors perform intensively. But due to the novelty of QVT, there is neither foundation nor theory about how to use this language to perform pattern matching, and, additionally, the QVT specification [13] is confusing in this respect.

For this reason, a basic framework has been developed to support (at least on a theoretical level) the MDPEM process. Fig. 1 depicts the framework. For the sake of simplicity all the involved artifacts are located in an MOF-like pyramid. Level M2 of Fig. 1 represents the metamodels for patterns and target models, as well as the model transformations involved in the process; Level M1 represents all the models involved in the MDPEM process, namely the patterns, the target model, and the matchings (occurrences of the pattern in the target model). Since all the steps are developed in M2 and M1, M3 and M0 are not explained. Table I summarizes the elements represented in Fig. 1. For each element ("*Level M2*" column in TABLE I) on the M2 level, there exists a conforming model ("*Level M1*" column in TABLE I) on the M1 level.

The *PatternToQVTPattern* transformation, not included in Table I, plays an important role in the MDPEM process. As noted above, any pattern (*PatternModel*) must be transformed into a different representation that QVT imposes: the *QVT Template* (shown as *QVTPattern Model* in Fig. 1). The *QVT Template* is the structure that QVT uses to represent any pattern that needs to be matched. As a consequence, any pattern we want to match should be transformed to this representation. Therefore, a transformation (*PatternToQVTPattern* in Fig. 1, *Level M2*) involves both the pattern metamodel (PatternMModel) and the QVTPattern MModel (QVT Template Metamodel).

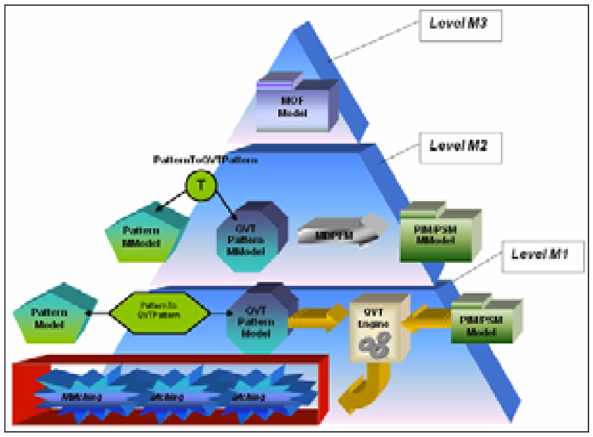


Fig. 1. MDPEM Framework

TABLE I
ELEMENTS INVOLVED IN THE MDPEM PROCESS

Level M2	Level M1	Description
PMM	PM	Patterns (models) are constrained by a given metamodel. This must be compatible with the target model to be able to find matchings.
QVTPMM	QVTPM	Patterns must be translated into a QVT pattern representation (QVT Template [13]), so the QVT Template metamodel (QVTPMM) should be taken into account. As a consequence, each pattern model is transformed to a QVT Template (QVTPM)
PIM/PSMMM	PIM/PSMM	Because in any MDA process models are called PIM or PSM (according to the abstraction level of the model), we suppose that the target model will be a PIM or a PSM
-	Matching	After MDPEM is applied, matchings may be found. This matchings can be seen as fragments of the target model, so the metamodel for these elements can also be the PIM/PSMMM
PMM = Pattern Metamodel; PM = Pattern Model; QVTPMM = QVT Pattern Metamodel; QVTPM = QVT Pattern Model; PIM/PSMMM = PIM/PSM Metamodel; PIM/PSMM = PIM/PSM Model		

The MDPEM process can be divided into four steps:

1. Pattern model and target model are defined.
2. QVT Template model is obtained from Pattern Model.
3. MDPEM is carried out.
4. Matchings (sub-models) are returned to the invoker.

For a deeper explanation of MDPEM and examples of patterns, please see [29].

IV. Finding/defining defining pattern on models

The previous section outlined the MDPEM technique, but the main goal of this technique is not only to find occurrences for a given pattern. MDPEM can be also used to undertake more detailed operations over those models acting as target models.

This technique was conceived as a tool to support some parts of an MDA process to extract services from relational databases [30]. In this process, patterns are used to extract functionality from relational databases. That led us to the question: how to use MDPEM to do something more than find matchings?

A. Patterns plus actions to derive functionalities

In this paper our aim is to provide a mechanism to use MDPEM undertaking actions associated with the set of occurrences obtained in the matching process (Fig. 2).

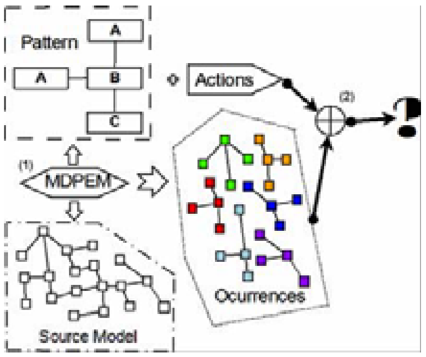


Fig. 2. Extended MDPEM

The basic MDPEM depicted in Fig. 1 can be expressed as a function f_{MDPEM}. This function is defined as follows: $f_{MDPEM}(M_P, M_T) = Set(M_O)$, where M_P represents the pattern, M_T represents the source model, and M_O represents an occurrence of M_P in M_T. Depending on both the pattern model and the target model, f_{MDPEM} may return a set of occurrences ($Set(M_O)$).

But according to Fig. 2, the MDPEM basic concept can be extended to provide an extra functionality. Patterns can be provided together with a description of an abstract action. The basic idea behind Fig. 2 is to have not only a pattern (M_P) but also an associated abstract action, which could be later transformed into a concrete transformation.

However, MDPEM can be specified in a more powerful way. First of all, instead of a pattern M_P, the process takes a pair <*pattern, abstract action*>, algebraically represented as $K = (M_T, A_A)$, where M_T represents the aforementioned pattern model, and A_A represents the abstract action. Therefore, the previously defined function f_{MDPEM} must be redefined according to the new element K. This new function can be considered as follows: $f_{MDPEM}(K, M_T) = Set(A_A(M_O))$. A_A is then applied to each element in the resulting set of occurrences. Thus the abstract action, which is defined in terms of the abstract elements of the pattern, is then "materialized" for each actual occurrence obtained in the MDPEM process.

In Fig. 2, K is represented by the pattern plus the action (top of the figure), and M_T is the source model depicted at the bottom of figure. After applying MDPEM and obtaining the occurrences, the specified action is then applied to each occurrence.

The term "abstract action" may be very vague, but in fact any action or structure can be specified to be applied to each occurrence. As a consequence, certain parts from models (MDPEM occurrences) can be updated or modified according to the instructions given in the abstract action (A_A).

In the following section, we will describe what MDPEM has been used for: to detect services in databases using patterns and abstract descriptions for the services attached to patterns.

V. Using MDPEM to find and create services

This section explains the purpose of this technique. As noted, MDPEM was developed in the context of an MDA process to extract services from relational databases. The lack of tools and experience around languages such as QVT prompted the development of MDPEM to properly manage models and patterns.

In the context of the process proposed in [30], we can define a pattern and attach a particular behavior to this pattern. This behavior is specified in terms of operations, called services. Thus, for a given pattern involving a set of entities, a set of abstract operations involving these entities can be attached. Thus, when an occurrence is found, each abstract operation attached to the pattern is "instantiated" with the entities of the occurrence.

A. Problem definition

MDPEM can be used to discover services in relational databases. Thus, all the artifacts (patterns and target model) will be described in terms of tables, columns, foreign keys and so on.

The aim is to start from a database model, define a set of patterns, find their matches and obtain possible useful services. This case study fits the example depicted in Fig. 2, and:

- the database model plays the role of the source model,
- a pattern describing a particular database fragment structure plays the role of the pattern, and
- the abstract description of an operation involving the elements of the pattern.

B. Pattern model and target model definition

Fig. 3 depicts a fragment from a relational database extracted from an industrial application. This fragment is composed of 10 tables and will form the *target model* for the MDPEM process. This model has been used to carry out a simple case study and illustrates the work presented in this paper. Due to the lack of space, the picture does not include any detail such as columns, primary keys or the like. It can be assumed that a suitable database metamodel is used (in fact, a summarized version of the one presented in [31] is used).

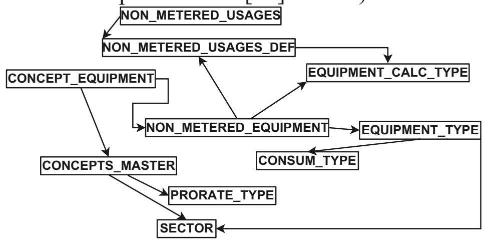


Fig. 3. Case study database

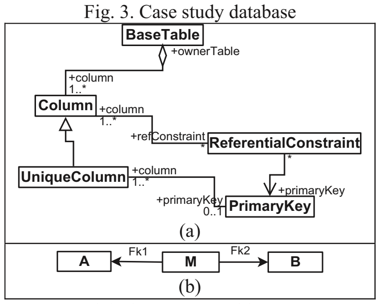


Fig. 4. (a) Pattern metamodel, (b) DFK (Double Foreign Key) Pattern, and (c) DFK pattern conforming to pattern metamodel

Fig. 4 (a) represents the pattern metamodel. Since it may not be sufficiently descriptive, Fig. 4 (b) shows a conceptual representation. Fig. 4 (b) shows the pattern to be applied in the MDPEM process.

Once the source model and pattern are defined, the abstract operation must be defined (the "Action" in Fig. 2). In this context, any service (or operation) against the database is translated into an SQL query. But in spite of existing proposals to represent SQL queries in a metamodel-like representation (PL/SQL Grammar[1]), there is no standardized DML metamodel for SQL. Nonetheless, a suitable solution lies in the OMG's metamodel family, OCL2. OCL2 is as expressive as SQL. Furthermore, the OCL2 specification includes the language metamodel, so that any OCL expression can be represented as a model [32] and thus used in any MDA approach as an artifact. For these reasons, OCL was chosen to describe the abstract operations.

In the pattern (Fig. 4 (b)), table M has two foreign keys: one to table *A* and one to table *B*. The *M* primary key is obtained by linking together the foreign key for table *A* and the foreign key to table *B*. Thus, for a given value of the *B* primary key it is possible to obtain those records from table *A* linked to table *B*. As a consequence, two operations or services can be attached to the pattern in Fig. 4 (b):

- *getA_TupleBy_B(b:B):bag(A)* → Having the values of the *B* table tuple, it is possible to obtain a list of tuples from A by means of table *M*.
- *getB_TupleBy_B(a:A):bag(B)* → As in the previous operation, here it is possible to return a set of tuples to table *B*. It is only necessary to pass the value of an *A* table primary key to perform the search.

Fig. 6. (a) includes the required OCL postcondition to specify the service *getA_TupleBy_B(b:B):bag(A)*. Supposing that these services are allocated in a façade class [27], the OCL expression:

1. Recovers all the instances (M is a class representing a table) of *mM* property,
2. for each instance:
   a. calculate the values or properties which implement the foreign key to *B*,
   b. calculate the values of the properties conforming the primary key of *b* (argument of the operation),
   c. calculate which of these sets have the same values, and
3. for each of these sets, return the corresponding *mA* attribute in the *M* context.

The Fig. 6 (b) represents the same expression, but in a model-like representation. It should be noted that all the elements referred in the expression belong to the pattern (see Fig. 4 (b)).

C. QVT Template generation

Once all the elements involved in the process have been defined, the pattern must be transformed into a suitable representation to perform the MDPEM process.

Pattern =
{
e_1: <classname$_1$>, e_2: <classname$_2$> .... e_n:<classname$_N$>
l_1 : <assoc$_1$> (e_i, e_j) .... l_m:<assoc$_M$>(e_u, e_w)
where <predicate>
}

Fig. 5. *QVT Template* abstract representation

As noted above, QVT defines its own pattern representation: the *QVT Template* [13] (also called *QVT Pattern Model* in Fig. 1). A *QVT Template* (see Fig. 5) represents a pattern on the basis of the elements involved in the pattern and the relationships between them. Thus, before performing the MDPEM process the input pattern must be transformed into a *QVT Template*. In the proposed framework (see Fig. 1), this transformation is carried out by means of a QVT transformation (*PatternToQVTTemplate*).

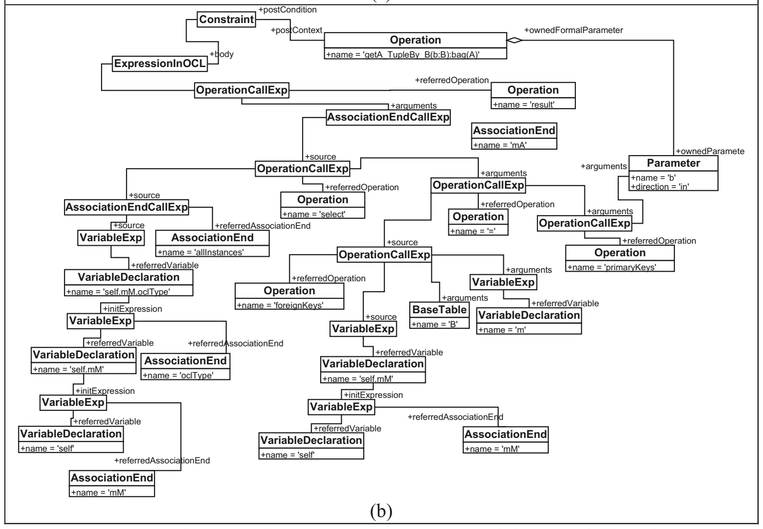
This transformation, outlined in Fig. 7, transforms patterns (based on the metamodel in Fig. 4 (a)) to a *QVT Template* representation. Thus, starting from a pattern and then applying this transformation, a suitable QVT Template is obtained to perform MDPEM. The Fig. 7 transformation is divided into five relations. Each relation is intended to address the transformation from each part of the original metamodel (Fig. 4 (a)) to other part in the QVT Template one (Fig. 5). See [29] for a further description of this transformation.

D. MDPEM application and matchings generation

This step depends entirely on the QVT engine that executes the matching. As can be seen in Fig. 1, the QVT engine takes the source model (referred in the picture as the "PIM/PSM Model") and the QVT Template (obtained from the initial pattern) as input. Then, the QVT engine executes the matching and automatically returns the occurrences.

[1] https://javacc.dev.java.net/files/documents/17/2959/FormsPlSql.jj

```
result = self.mM.oclType.allInstances
        ->select(m:M|foreignKeys(m,mB) = b.primaryKeys()).mA
```
(a)



(b)

Fig. 6. (a) OCL postcondition for the *getA_By_B(p:B):Set(A)* and (b) its model-based representation

The obtained matchings can be seen as constrained views of the target model. These small "sub-models" represent something interesting from point of view of the engineering which is supervising the MDPEM process. In this work, we present one of the possible uses of MDPEM: the inclusion of operations in the recovered matchings.

```
transformation SQL92SchemaPatternToQVTTemplate
   (sql92p: SQL92SchemaPattern, qvtt:QVTTemplate)
{
key ObjectTemplateExp = {name};
key PropertyTemplateItem = {name};
top relation BaseTableToObjectTemplateExp{…}
top relation ColumnToObjectTemplateExp{…}
top relation UniqueColumnToObjectTemplateExp{…}
top relation ReferentialConstraintToObjectTemplateExp{…}
top relation PrimaryKeyToObjectTemplateExp{…}
}
```
Fig. 7. QVT transformation to generate a QVT Template from a patter for database models

As depicted in Fig. 2, patterns are given together with the abstract description of an operation (shown, for example in Fig. 6). After MDPEM is applied and the matchings are recovered, the abstract operation is applied to each obtained occurrence. Thus, since the elements of the matchings belong to the target model, the elements involved the abstract operation will no longer be abstract.

TABLE II
MATCHINGS OBTAINED IN THE MDPEM PROCESS

ID	A	M	B
1	CONCEPTS_ MASTER	CONCEPT_ EQUIPMENT	NON_METERED_ EQUIPMENT
2	CONSUM_TYPE	EQUIPMENT_TYPE	SECTOR
3	PRORATE_TYPE	CONCEPTS_MASTER	SECTOR

For the given pattern in Fig. 4 (a), and the target model in Fig. 3, the set of matchings after applying MDEM are described in the TABLE II, and the resulting services in TABLE III.

TABLE III
SERVICES OBTAINED FROM MATCHINGS AND ABSTRACT OPERATIONS

ID	SERVICE
1	-*getCONCEPTS_MASTER_TuplesBy_NON_METERED_EQUIPMENT* - *getNON_METERED_EQUIPMENT_TuplesBy_CONCEPTS_MASTER*
2	- *get*CONSUM_TYPE_TuplesBy_SECTOR - getSECTOR_TuplesBy_CONSUM_TYPE
3	- *getPRORATE_TYPE_TuplesBy_SECTOR* - *getSECTOR_TuplesBy_PRORATE_TYPE*

Of course, the names of the services in TABLE III are not intuitive and the names of the abstract operations are just illustrative. Services created from matchings and abstract operations must be renamed by the engineer, according to the service's functionality. Therefore, services become intuitive and easy to use. For each service in TABLE III, references to abstract elements (namely A, B, and M) are substituted by the elements in the matching playing this "abstract role" (e.g., in TABLE II, in the occurrence with ID = 2, "*SECTOR*" plays the role of *B*, *"CONSUM_TYPE"* the role of *A*, and "*EQUIPMENT_TYPE*" the role of *M*.

Due to the lack of space, an abstract operation "instantiated" for a matching could not be shown.

VI. CONCLUSIONS AND FUTURE WORK

This paper presents MDPEM, an approach to perform pattern matchings in the context of models. The proposed framework focuses on QVT, a particular language to manipulate models by means of queries, views and transformations.

In addition to MDPEM, this paper focuses on what MDPEM is able to do with models. The main idea behind MDPEM is to use the models obtained as a result of the matchings to perform an action over these models.

Our aim is to use the dual "pattern-abstract operation" against a target model. Thus, portions of the target model (the occurrences of the pattern) can be annotated with operations or services (described as abstract operations related to the searching pattern). The instrumentation of models with these services facilitates the implementation of MDA-based development processes.

As well as development, MDPEM might be useful in other areas. Software maintenance in a model-level can be improved using this technique, e.g., looking for design patterns, as well as looking for bad-smells and correcting them (in a model – level). Every process described as a "*search pattern and apply rule*" sequence can be partially automated and then easily applied.Due to the lack of tools for implementing the QVT language, MDPEM has not been tested in an automated environment. As soon as a tool is available, this theoretical proposal will be migrated to a *runnable* one

For each pattern metamodel, it is necessary to develop a suitable transformation to generate the equivalent *QVT Template* model. Therefore, it may be useful to automate the generation of the *QVT Template* from some pattern metamodels.

Acknowledgments

This work was partially supported by the ENIGMAS poject (PIB-05-058), the FAMOSO Project (2006: FIT-340000-2006-67); and the MECENAS project (PBI06-0024).

References

[1]. Bézivin, J. *Introduction to Model Engineering*. 2006. <www.modelware-ist.org> [Accesed: 14-November-2006]

[2]. OMG. *Unified Modeling Language: Superstructure. Versión 2.0*. 2005. <http://www.omg.org/docs/formal/05-07-04.pdf> [Consultado el: 21 de Noviembre del 2005]

[3]. OMG, *OCL 2.0 Specification. Version 2.0*. 2005, Object Management Group (OMG). p. 185.

[4]. OMG. *Meta Object Facility (MOF) Specification*. 2002. <http://www.omg.org/docs/formal/02-04-03.pdf> [Accessed: 11-05-2005]

[5]. OMG. *Common Warehouse Metamodel (CWM) Specification*. 2003. <http://www.omg.org/docs/formal/03-03-02.pdf> [Consultado el: 29-09-2005]

[6]. OMG. *Architecture-driven Modernization (ADM): Knowledge Discovery Metamodel (KDM) Specification*. 2006. <http://www.omg.org/docs/ptc/06-06-07.pdf> [Accessed in February, 2006]

[7]. OMG. *ADM Task Force*. 2006. <http://adm.omg.org> [Accessed in February 2007]

[8]. OMG, *MDA Guide Version 1.0.1.* 2003, Object Management Group. p. 62.

[9]. Maamar, Z., et al. *Towards a contextual model-driven development approach for Web services*. in *Proceedings of the 8th International Conference on Enterprise Information Systems (ICEIS 2006)*. 2006. Pahos, Cyprus

[10]. Xiaofeng, Y., et al. *A Model Driven Development Framework for Enterprise Web Services*. in *Proceedings of the 10th Conference on Enterprise Distributed Object Computing (EDOC'06)*. 2006. Hong Kong, China: IEEE Computer Society IEEE Computer Society. ISBN. 0-7695-2558-X

[11]. Reus, T., H. Geers, and A. van Deursen. *Harvesting Software for MDA-Based Recovering*. in *European Conference on Model Driven Architecture - Foundations and Applications*. 2006. Bilbao (Spain): Springer-Verlag Berlin Heidelberg. **LNCS 4066** Springer-Verlag Berlin Heidelberg, A. Rensink and J. Warmer, Editors. ISBN. 3-540-35909-5

[12]. van den Heuvel, W.-J. *Matching and Adaptation: Core Techniques for MDA-(ADM)-driven Integration of new Business Applications with Wrapped Legacy Systems*. in *Model-Driven Evolution of Legacy Systems (MELS 2004)*. 2004. Monterey, California, USA

[13]. OMG. *MOF QVT Final Adopted Specification*. 2005. <http://www.omg.org/docs/ptc/05-11-01.pdf> [Accessed in February, 2005]

[14]. OMG. *QVT-Partners initial submission to qvt-rfp*. 2003. <http://www.qvtp.org/downloads/1.0/qvtpartners1.0.pdf> [Accessed in November, 2006]

[15]. OMG. *Revised submission for MOF 2.0 Query/Views/Transformations RFP*. 2003. <http://www.omg.org/docs/ad/04-04-01.pdf> [Accessed in November, 2006]

[16]. Queralt, P., et al. *Un Motor de Transformación de Modelos con soporte para el Lenguaje QVT Relations*. in *Desarrollo del Software Dirigida por Modelos y Aplicaciones 2006 (DSDM´06)*. 2006. Sitges, Barcelona (Spain)

[17]. ModelWare. *The ModelWare QVT Tool*. 2007. <http://www.modelware-ist.org/> [Accesed: 15-02-2007]

[18]. Bodhuin, T., E. Guardabascio, and M. Tortorella. *Migrating COBOL Systems to the WEB by Using the MVC Design Pattern*. in *Ninth Working Conference on Reverse Engineering (WCRE'02)*. 2002. Richmond, Virginia: IEEE Computer Society IEEE Computer Society

[19]. Goedicke, M. and U. Zdun, *Piecemeal legacy migrating with an architectural pattern language: a case study.* Journal of Software Maintenance and Evolution: Research and Practice, 2002. **14**: p. 1-30.

[20]. Brown, K. and B.G. Whitenack, *Crossing Chasms, A Pattern Language for Object-RDBMS Integration*, ed. K.S.C. Editor. 1995. K.S.C. Editor.

[21]. Yoder, J. *Patterns for making business objects persistent in a relational database*. in *Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*. 2002. Tampa Bay (Florida)

[22]. Yoder, J., R.E. Johnson, and Q.D. Wilson. *Connecting Business Objects to Relational Databases*. in *Proceedings of the 5th Conference on the Pattern Languages of Programs*. 1998. Minticello-IL (Estados Unidos)

[23]. Cagnin, M.I., et al. *Reengineering Using Design Patterns*. in *Seventh Working Conference on Reverse Engineering (WCRE'00)*. 2000. Brisbane, Australia: IEEE IEEE

[24]. Sartipi, K. and K. Kontogiannis. *On Modeling Software Architecture Recovery as Graph Matching*. in *International Conference on Software Maintenance (ICSM'03)*. 2003. Amsterdam, The Netherlands: IEEE Computer Society IEEE Computer Society

[25]. Pinzger, M., et al. *Revealer: A Lexical Pattern Matcher for Architecture Recovery*. in *Proceedings of the 9th Working Conference on Reverse Engineering*. 2002. Richmond, Virginia: IEEE Computer Society IEEE Computer Society

[26]. Stoermer, C., L. O´Brien, and C. Verhoef. *Practice Patterns for Architecture Reconstruction*. in *Proceedings of the 9th Working Conference on Reverse Engineering (WCRE´02)*. 2002. Richmond, Virginia: IEEE Computer Society IEEE Computer Society

[27]. Gamma, E., et al., *Design Patterns Elements of Reusable of Object-Oriented Software*. 1995: Addisson-Wesley.

[28]. QVTP. *Revised submission for MOF 2.0 Query / Views /Transformations RFP (Version 1.1)*. 2003. <http://tratt.net/laurie/research/publications/papers/qvtpartners1.1.pdf> [Consultado el:23-06-2005]

[29]. García-Rodríguez de Guzmán, I., M. Polo, and M. Piattini. *A Framework for Model-Driven Pattern Matching*. in *Proceedings of the 9th International Conference on Enterprise Information Systems*. 2007. Funchal, Madeira - Portugal (In press)

[30]. García-Rodríguez de Guzmán, I., M. Polo, and M. Piattini. *A Methodology for Database Reengineering to Web Services*. in *European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA 2006)*. 2006. Bilbao (Spain): Springer-Verlag Berlin Heidelberg. **LNCS 4066** Springer-Verlag Berlin Heidelberg, A. Rensink and J. Warmer, Editors. ISBN. ISBN 3-540-35909-5

[31]. Calero, C., et al., *An Ontological Approach To Describe the SQL:2003 Object-Relational Features.* Accepted in "Computer Standards and Interfaces", 2005: p. 28.

[32]. Reynoso, L., M. Genero, and M. Piattini. *OCL2: Using OCL in the Formal Definition of OCL Expression Measures*. in *Quality in Modeling (QIM´2006)*. 2006. Genova (Italy)

A MODEL-DRIVEN APPROACH TO ARCHITECTING SECURE SOFTWARE

Ebenezer A. Oladimeji
Architecture and eServices, IT
Verizon Communications
Irving, Texas 75038, USA.
email: ebenezer.oladimeji@verizon.com

Sam Supakkul
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas 75083, USA.
email: ssupakkul@ieee.org

Lawrence Chung
Department of Computer Science
The University of Texas at Dallas
Richardson, Texas 75083, USA.
email: chung@utdallas.edu

ABSTRACT

Software architecture provides a high-level description of a software solution in terms of the structure, topology, and interactions between its principal components. While a number of formal architectural description languages have been developed, a visual modeling approach seems to be more suitable for practitioners. There is also a lack of established tools or methodologies for integrating security requirements with software architectural models. Moreover, determining whether or not a given software architectural model realizes a set of security requirements remains a challenging problem. To address these issues, this paper proposes a model-based framework for architecting secure software. Specifically, we present a mapping strategy between the core elements of software architecture and a lightweight extension to the UML metamodel. We then describe how security requirements, captured in the forms of authorization and obligation security policies, can be visually integrated with the software architectural model. We show how this approach enables the early detection of security conflicts or inconsistencies during design, and the traceability of security concerns from requirements to architecture. The feasibility of the proposed approach is illustrated with an example of the design of a simplified health information system.

KEY WORDS
software security, software architecture, architectural models, security policies, security conflicts, security traceability

1 Introduction

As our lives continuously depend on several complex software-based systems, the security of software systems continues to play more significant roles. The need for designing security into software applications rather than retrofitting it as an afterthought has been well discussed [1, 18, 17]. Capturing security concerns during the requirements analysis and architectural design phases, in particular, can improve the overall security of a system. While a number of approaches have addressed how to capture security concerns during the requirements engineering phase [16, 15], more research is needed to enable the design of software architectural models that explicitly capture the security issues identified during requirements analysis, in a systematic manner.

The software architecture provides a description of a software solution at a critical level of abstraction. At the architectural level, software engineers describe the solution domain in terms of the structure and topology of its principal components and the interactions between them [5, 6]. Several architectural description languages (ADL's) [4], with varying expressive powers have been developed. However, there is lack of established tools or methodologies for integrating security policies with software architectural models. We believe that explicit modeling of security issues at the architectural level can make a system more resistant to vulnerabilities.

However, while it has been argued [4] that security concerns should be explicitly addressed during requirements engineering and architecture design stages, in practice, this is seldom the case. One reason for this is the lack of standardized methodology and visual notations for specifying and analyzing security issues at the architectural level. While ADLs have matured to some extent, practitioners are not very excited about using them for architectural descriptions. A visual modeling approach to software architecture design seems to be more suitable in practice. Two approaches have been identified for addressing this problem [6]. One is to use specialized notation for software architectures, and the other is to adapt a general-purpose modeling notation such as the UML. We adopt the latter strategy because of familiarity of the UML to developers and the existence of commercial tool supports. Moreover, determining whether or not a given software architectural model realizes a set of security requirements remains a challenging problem.

In this paper, we present a model-based approach for explicitly capturing security concerns into UML-based models of software architecture. Our goal is to provide practitioners with a useful artifact that can guide the way security issues are handled during the rest of the development process. Specifically, we are concerned with the representation of security requirements and their integration with visual models of software architectures. We illustrate the concepts presented with the example of designing a set of architectural models for securely developing a web-based Simplified Health Information System (*SHIS* hereafter), whose informal architecture is shown in Figure 1.

The rest of the paper is organized as follows: Sections 2 and 3 describe our modeling notations and Section 4 describes the proposed approach. Section 5 discusses related research while Section 6 concludes the paper.

2 Modeling Security Requirements

During the requirements engineering phase, security requirements are usually captured in the form of high-level security policies. These policies typically describe *who* is granted *what* level of access to *what part* of the system. Marriott et al in [12] describe two major categories of security policies for managing distributed systems in general, namely authorization and obligation policies. An *authorization policy* specifies what *actions* a given *subject* (agent, user, role, or process) is *permitted* or *forbidden* to perform on a set of target *objects*. On the other hand, *obligation policies* are rules that specify what activities a subject *must* or *must not* perform on a set of target objects under an optionally specified condition or system event (i.e. they specify job functions related to security management). These policies can be expressed in any suitable policy specification languages such as Ponder [2], and Rei [10]. For convenience, we adapt *Ponder* and denote security policies with the following syntax:

> *modality identifier*: {**on** *event*} [*subjects*; *objects*, *actions*/*] {**when** *condition*}

where *modality* is one of *A+* denoting *positive authorization*, *A-* denoting *negative authorization*, *O+* denoting *positive obligation*, and *O-* denoting *negative obligation*. The terms in curly brackets are optional, while the terms in bold typeface are keywords. For simplicity, identifiers, events, subjects, objects, actions, and conditions are specified in natural language or set notation. An *action* is a high-level description of a set of related operations and an asterisk denotes *all possible actions* on the target object. Consider the following example policies from the *SHIS* application domain:

- A+ P1: [Doctor; MedicalRecord; *]
- A+ P2: [MedicalStaff; MedicalRecord; View]
- A- P3: [Nurse; CreditCardInfo; View]
- O+ P4: on request-for-disclosure [AdminStaff; MedicalHistory; Obtain-Permission]
- O- P5: [Nurse; CreditCardInfo; View]
- A+ P6: [AdminStaff; CreditCardInfo; Update]

Policy *P1* denotes that doctors are authorized to perform all possible actions on their patients' medical records, while policy *P3* specifies that nurses are *forbidden* from viewing patients' credit card data. On the other hand, *P4* is a positive obligation policy denoting that on any event of a request for disclosure from any medical research agency, the admin staff *must* obtain permission from the patient whose information is to be disclosed for research purposes. Similarly, the negative obligation policy *P5* denotes that nurses *must not* even view credit card information of patients.

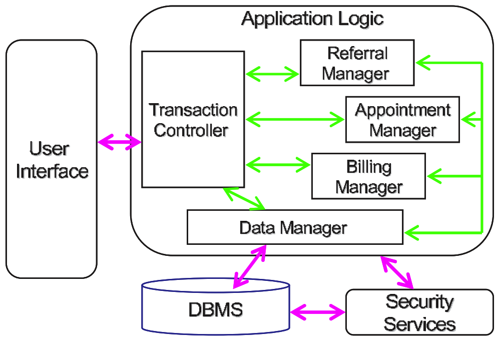


Figure 1. An Informal Representation of the Software Architecture for *SHIS*

3 UML-Based Software Architectural Models

Since software architecture is produced early in the development process, it promises a means for less-expensive design-time analysis and reasoning about emergent system properties such as performance, scalability, and security. However, this promise is only realizable if architectures are described with semantically rich and expressive notations. In practice, architectural designs are usually represented with *box-and-line* diagrams. Figure 1 shows an example model of software architecture for the *SHIS* based on this informal approach. Representing software architectures in this manner, visually intuitive as it may seem, suffers a number of drawbacks. First, these diagrams have no formal semantics and are therefore subject to different interpretations, thus limiting their usage as communication tools among stakeholders. Second, traceability between these informal diagrams and other design models is difficult to ascertain. Furthermore, they are not amenable to any formal reasoning or analysis that may reveal incompleteness, conflicts or inconsistencies.

To address these drawbacks, some ADL's have emerged (see [5] for a list). While these languages vary in their conceptual details, there seem to be an ontological consensus among them about the major elements of a software architectural description. From existing work [6, 14] and from a practitioner's point of view, the core concepts that *must* be modeled in a software architectural description are as follows: *components* (with their required and provided *interfaces, types* and *ports*), *connectors* (with their required and provided *interfaces*), and *configurations*.

Since the UML is the de-facto standard for object modeling, several efforts have been recently made at mapping ADL concepts to the UML [7, 13]. Garlan et-al [6] present a thorough examination of this space and propose some strategies for mapping ADLs concepts to the UML. They argue that the UML (1.x) component lacks semantic match for the component concepts in software architecture. However, with the advent of UML 2.0, a number of revisions and new concepts provide a significant bridge to the gap between the UML component and the architectural component. These revisions are shown in the unshaded portions of the metamodel of Figure 2. It should be noted that in UML 2.0, a *compo-*

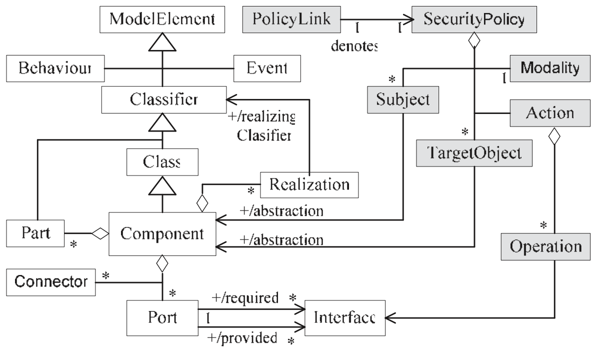


Figure 2. A Metamodel for the Proposed Approach

nent is a subtype of a *class*, which implies it may now have attributes and operations, as well as participate in associations and generalizations. Also, UML 2.0 provide explicit modeling elements for the concepts of *provided* and *required interfaces* and ports. These revisions makes the UML 2.0 *component diagram* suitable as a standard model of software architectures. We therefore use the following mapping strategy for the core elements identified above:

Components: These represent the computational elements and data stores of a system. Components may have multiple points of interactions called *ports* through which the component exposes a number of *provided* and *required interfaces* for interacting with its environment. This architectural conceptual element maps intuitively to the UML 2.0 component model-element. For an example, Figure 3 shows a UML component model for *SecurityServices* component of the *SHIS* application. This component provides two interfaces (*RBAC* and *Encryption*), and requires a *Persistence* interface, all through different ports.

Connectors: These are the means of interactions among components, providing the glues in architectural designs from a run-time standpoint. A connector can be abstract or concrete. An abstract connector represent simple forms of interactions such as pipes, procedure calls, and event broadcasts. We model this type of connectors with UML *assembly* and *delegation* connectors. An assembly connector maps a required interface of a component to a provided interface of another component in a certain context. Similarly, delegation connectors connect the externally provided interfaces of a component to the parts that realize or require them. For example, in Figure 3, the provided interface *RBAC* is delegated to the *RBAC-Controller* sub-component while the *Encryption* interface is realized by the *EncryptionEngine* class. On the other hand, concrete connectors perform complex coordination activities (e.g client-server protocols) through one or more provided and required interfaces. These type of connectors can be modeled as *stereotyped components*.

Configurations: Also known as *architectural styles*, architectural configurations typically define a vocabulary of design element types as a set of component, connector, port, role, binding, and property types, together with rules for composing instances of the types [5]. We model architectural configurations with the topology (arrangement and in-

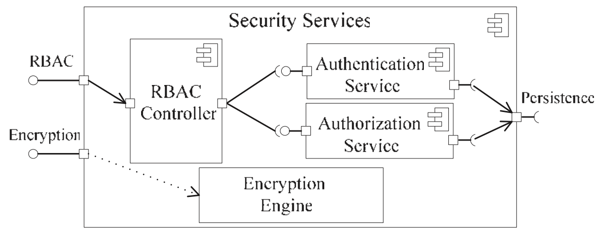


Figure 3. A Component Diagram Example

terconnection) of the component diagram. UML *association links* and *interface bindings* are used to depict the topology of complex components. Also, since the *parts* in a component can be UML classifiers, component diagrams can be drawn at different levels of detail of the inner parts, thus enabling the modeling of complex hierarchical structures.
Next we describe our approach for depicting security abstractions on UML-based models of software architecture.

4 Architecting Secure Systems

To enable the production of semantically rich visual models of software architecture that explicitly depict security concerns, we extend the UML 2.0 component metamodel as illustrated in Figure 2. The extension metaclasses in Figure 2 are shaded to differentiate them from the UML 2.0 standard metaclasses. An architectural model (represented by a component diagram) based on this extended metamodel helps to establish traceability of security objectives from the requirements to the architectural models. It can also help in performing some design-time analysis thus significantly improving the overall security of the software system.

4.1 Representation of Security Concerns

We represent security concerns on the software architectural model in two ways as follows:

Modeling Security Mechanisms: Designers often use established prevention mechanisms (such as encryption, authentication, firewalls, virtual private networks), as well as detection mechanisms (such as content filtering, virus checkers and audit log analyzers). Since most software security problems lie at the points of interactions among components, *ports* constitute the best place to implement these security mechanisms. Therefore, we depict these mechanisms as *tagged-values* of the ports in component diagrams. Appropriate single or multi-valued properties such as '*SecurityMechanisms*' and '*ConnectionType*' can be modeled as attributes of the ports. For example, the *ConnectionType* property of a port can denote complex communication protocols such as *SSH*, *SFTP*, or communication types like *SQL*, that are enforceable in the port. While preserving the semantics of UML tagged-values notation, we show only the values of the properties in order to avoid visual clutter.

Figure 4 depicts the UML-based software architectural model for securely building the *SHIS* application. It shows that the *User Interface* component communicates with the

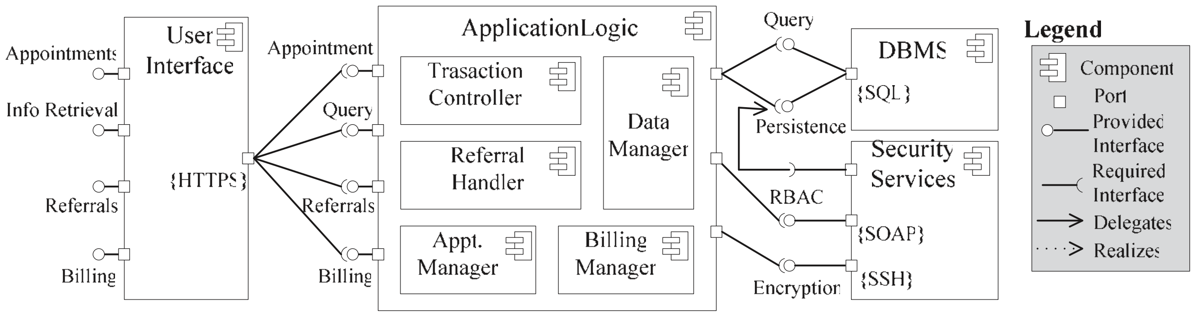


Figure 4. A Software Architectural Model for *SHIS* from a UML Component Diagram Perspective)

ApplicationLogic component using *HTTPS* protocol. Similarly, the *Encryption* interface required by the *ApplicationLogic* and provided by the *Security Services* component is implemented over a port that implements secure shell (*SSH*).

Modeling Security Policies: Apart from the interactions among components, the internal parts of a component need to implement relevant security policies that were elicited during the requirements engineering phase. These policies needs to be captured in the architectural model. To achieve this, we extend the UML 2.0 metamodel to enable the integration with the security-specific terminologies and notations described in Section 2. As illustrated in Figure 2, each *security policy* is denoted on the architectural model with a *policy link*, and is composed of its *modality* and the sets of *subjects, target objects*, and *actions*. The subjects and objects are abstracted into UML components and realized with *classifiers* (classes or components). Each *action* consists of a set of related operations implemented and exposed as interfaces. Since the *parts* of a component can be any classifier, the UML component diagram can be drawn at different levels of abstractions, showing only the most important internal parts at a particular level. Internal details of major complex components can be shown with other models such as class diagrams (or domain models), component diagrams, or collaboration diagrams.

For example, Figure 5 shows how the internal details of the *DataManager* component of the *SHIS* application is modeled. The internal parts and their relationships are depicted with a domain model. The authorization and obligation policies identified in Section 2 are then shown with *policy links*. This method can be applied iteratively at different levels of abstraction when dealing with complex systems.

4.2 Traceability and Consistency Analysis

After an architectural model that captures security concerns is created as described above, it is desirable to be able to reason about the security properties of the system under development. Specifically, we are concerned with the problem of determining whether or not a given software architectural model realizes a set of security policies. Another problem of interest is whether or not such a realization occurs in a consistency preserving manner. To address these problems, we develop the following analysis rules for establishing traceability and detecting conflicts among security policies.

For precision, Let $SECPOL$ be the set of all security policies elicited during the requirements phase, and let S, O, and A be respectively the set of *subjects*, target *objects*, and possible *actions* in the problem domain. Then, each policy $p \in SECPOL$ is a 4-tuple structure denoted by:

$$p = [mod, s, o, a]$$

where $mod \in \{A+, O+, A-, O-\}$ is the *modality* of the security policy, $s \in S, o \in O$, and $a \in A$. Furthermore, suppose $SOFTARCH$ denotes the software architectural model (represented by a component diagram). Let $COMPS$ be the set of components and L be the set of policy links explicitly captured in $SOFTARCH$. We define two functions $M_1 : S \cup O \rightarrow COMPS$ which maps a subject or target object to a given component C in $SOFTARCH$, and $M_2 : SECPOL \rightarrow L$ which maps a given security policy to a particular policy link in $SOFTARCH$. Then, the following rules can be applied:

Completeness Rule: $SOFTARCH$ is *complete* with respect to $SECPOL$ if $SOFTARCH \models \phi_0$ where:

$$\phi_0 = \{\forall p = [mod, s, o, a] \in SECPOL, [\exists c_1, c_2 \in COMPS|((M_1(s) = c_1)\hat{} (M_1(o) = c_2)) \Rightarrow (\exists l \in L|M_2(p) = l)]\}$$

In other words, each subject or target object in the problem domain that has something to do with the requirements specification, must be mapped to a component in the architectural model (though not necessarily one-to-one). Similarly, each security policy identified during requirement analysis must be mapped to some policy link in the architectural model. This rule helps to establish the conceptual traceability from requirements to architecture.

Consistency Rules: Security policy conflicts and inconsistencies may arise due to interactions between components, since these components form the underlying elements of different security policies. A *conflict* occurs whenever there is an interaction between positive and negative policies of the same type (i.e. A+/A- or O+/O- applying to the same subject or object). On the other hand, an *inconsistency* occurs whenever there is a coexistence of positive and negative policies of different types (i.e A+/O- or A-/O+ applying to the same subject or target object). We consider inheritance and composition relationships among compo-

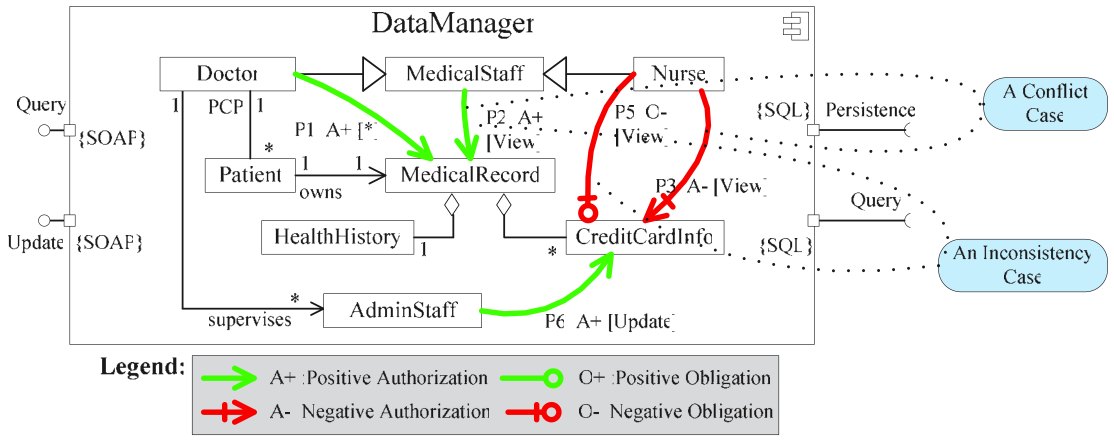


Figure 5. Security Policies Enforced in the *SHIS* Software Architecture

nents. We denote *aggregation* by the function $IsPartOf : COMPS \rightarrow COMPS$, and *specialization* by the function $IsA : COMPS \rightarrow COMPS$. Suppose that $p_1, p_2 \in SECPOL$ are any two security policies such that $p_1 = [mod, s_i, o_m, a]$ and $p_2 = [mod, s_j, o_n, a]$ for some i, j, m, and n. Then, $SOFTARCH$ is said to be *consistent* with respect to $SECPOL$ if it satisfies the following rules:

(a) $SOFTARCH \models \phi_1$ where:

$$\begin{aligned}\phi_1 = \neg\{&(p_1 = [A+, s_i, o_m, a]\,\hat{}\,p_2 = [A-, s_i, o_m, a])\\ &\text{v } (p_1 = [O+, s_i, o_m, a]\,\hat{}\,p_2 = [O-, s_i, o_m, a]) \text{ v}\\ &(p_1 = [O-, s_i, o_m, a]\,\hat{}\,p_2 = [A+, s_i, o_m, a]) \text{ v}\\ &(p_1 = [A-, s_i, o_m, a]\,\hat{}\,p_2 = [O+, s_i, o_m, a])\}\end{aligned}$$

(b) $SOFTARCH \models \phi_2$ where:

$$\begin{aligned}\phi_2 = IsA(o_m, o_n) \Rightarrow&\\ &\neg\{(p_1 = [A-, s_i, o_n, a]\,\hat{}\,p_2 = [A+, s_i, o_m, a]) \text{ v}\\ &(p_1 = [O-, s_i, o_n, a]\,\hat{}\,p_2 = [O+, s_i, o_m, a]) \text{ v}\\ &(p_1 = [O-, s_i, o_n, a]\,\hat{}\,p_2 = [A+, s_i, o_m, a]) \text{ v}\\ &(p_1 = [A-, s_i, o_n, a]\,\hat{}\,p_2 = [O+, s_i, o_m, a])\}\end{aligned}$$

(c) $SOFTARCH \models \phi_3$ where:

$$\begin{aligned}\phi_3 = IsPartOf(o_m, o_n) \Rightarrow&\\ &\neg\{(p_1 = [A+, s_i, o_n, a]\,\hat{}\,p_2 = [A-, s_i, o_m, a]) \text{ v}\\ &(p_1 = [O+, s_i, o_n, a]\,\hat{}\,p_2 = [O-, s_i, o_m, a]) \text{ v}\\ &(p_1 = [O-, s_i, o_n, a]\,\hat{}\,p_2 = [A+, s_i, o_m, a]) \text{ v}\\ &(p_1 = [A-, s_i, o_n, a]\,\hat{}\,p_2 = [O+, s_i, o_m, a])\}\end{aligned}$$

(d) $SOFTARCH \models \phi_4$ where:

$$\begin{aligned}\phi_4 = IsA(s_i, s_j) \Rightarrow&\\ &\neg\{(p_1 = [A+, s_j, o_m, a]\,\hat{}\,p_2 = [A-, s_i, o_m, a]) \text{ v}\\ &(p_1 = [O+, s_j, o_m, a]\,\hat{}\,p_2 = [O-, s_i, o_m, a]) \text{ v}\\ &(p_1 = [A+, s_j, o_m, a]\,\hat{}\,p_2 = [O-, s_i, o_m, a]) \text{ v}\\ &(p_1 = [O+, s_j, o_m, a]\,\hat{}\,p_2 = [A-, s_i, o_m, a])\}\end{aligned}$$

(e) $SOFTARCH \models \phi_5$ where:

$$\begin{aligned}\phi_5 = IsPartOf(s_i, s_j) \Rightarrow&\\ &\neg\{(p_1 = [A+, s_j, o_m, a]\,\hat{}\,p_2 = [A-, s_i, o_m, a]) \text{ v}\\ &(p_1 = [O+, s_j, o_m, a]\,\hat{}\,p_2 = [O-, s_i, o_m, a]) \text{ v}\\ &(p_1 = [A+, s_j, o_m, a]\,\hat{}\,p_2 = [O-, s_i, o_m, a]) \text{ v}\\ &(p_1 = [O+, s_j, o_m, a]\,\hat{}\,p_2 = [A-, s_i, o_m, a])\}\end{aligned}$$

To illustrate how these rules can be applied, consider the security policy *P1* through *P6* identified for the *SHIS* application in Section 2, and suppose they are designed to be implemented by the *DataManager* component. Figure 5 shows the architectural model of the *DataManager* component that captures these security policies. An application of ϕ_3 to this model reveals that policy *P2 conflicts* with policy *P3*. This is because the target object *MedicalRecord* is an aggregation of *CreditCardInfo*. By policy *P3*, a nurse is forbidden from viewing patients' credit card data, but by policy *P2*, the same nurse is permitted to view credit card data (since *Nurse* is a type of *MedicalStaff*). Similarly, *P2* and *P5* are *inconsistent* by a violation of ϕ_4. This is due to the fact that the subject *MedicalStaff* is a generalization of *Nurse*. By policy *P5*, a nurse must not (i.e. under a negative obligation) view patients' credit card information, but by policy *P2*, the nurse (being a type of a medical staff) is permitted to view the credit card information. This implies that the nurse has an implicit authorization to perform an action that another policy obligates her not to perform. This is an example of a violation of the principle of least privilege.

The above rules are non-exhaustive. They represent the rules for detecting security policy conflicts and inconsistencies that results from *only* inheritance and composition relationships among *subjects* and target *objects* in a given UML-based software architectural model. Other forms of relationships (e.g. transitive UML associations among components) can trigger some other types of conflicts and inconsistencies among the underlying security policies.

5 Related Work and Discussions

Software architecture continues to attract attention because it promises to enable design-time analysis, and reasoning with non-functional requirements in general. Related work in this domain are in two broad categories: (1) secure architecture description languages (ADLs) and their mappings to UML, and (2) integration of security into UML or its extension. A number of ADLs have been proposed [5] and some attempts have been made at developing frameworks for mapping concepts expressed in ADLs to the UML modeling notation [7, 13, 17]. Other works including [11, 9] address how security requirements can be incorporated into UML.

More closely related to this research, however, are [8, 17, 3], which all support the notion that software architecture is a convenient abstraction for reasoning about security in distributed applications. Jensen et-al [8], in particular, proposed that security policies should be programmed in the connectors, separate from the application's code. These works however lack expressive visual notations and consistency checking procedures. Our approach complements these proposals by considering the transitions of security concerns from requirements to architectural design, thus offering practitioners a visual modeling technique for designing secure software systems.

We believe that a systematic application of the proposed approach offers a number of benefits. One, the software architectural model so generated can form the basis for reasoning about and discovering security policy errors as described in Section 4. Two, explicitly capturing security concerns at the architecture level can enable practitioners to establish traceability of security concerns from requirements to architecture. In addition, since the proposed approach is based on the a lightweight extension of the UML 2.0 metamodel, existing UML tools can be easily extended to support the approach. However, it is desirable for software architectural models to be amenable to change in order to reflect new realities and threats in the security environment. It is also desirable to visually depict policy conflicts and inconsistencies on the resultant architectural model such as Figure 5. These concerns are not addressed in the current proposal.

6 Conclusions

Security architectural modeling plays a significant role in the design of the overall security model for a system because it can help to ensure that security is built into applications, rather than retrofitted at later stages of the development process. We have described in this paper a framework for creating visual models of software architecture that explicitly captures elicited security policies, based on the UML 2.0. Specifically, we show how security policies captured during requirements analysis phase can be integrated into UML-based models of software architectures. We also demonstrate how some analysis rules are used to verify the completeness and consistency of the resultant architectural model.

This approach not only offers a smooth transition from requirements elicitation to high-level architecture design, but also greatly improves the traceability of security requirements from the problem to the solution domains. Moreover, the software architectural model so produced can serve as a valuable resource for detailed design, and can be applied as a basis for testing and validating the system implementation in terms of desired security features. Future work will be targeted at developing a tool support for the proposed approach. We also plan to extend the current proposal to address dynamic aspects of security, nesting of components, and consideration of the requirement-level object model.

References

[1] R. Crook, D. Ince, L. Lin, and B. Nuseibeh. Security Requirements Engineering: When Anti-Requirements Hit the Fan. In *Proc. of Int'l Requirements Engineering Conference*, 2002.

[2] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. *Policy Workshop 2001, Bristol, U.K., Springer-Verlag, LNCS*, Jan. 2001.

[3] Y. Deng, J. Wang, J. J. P. Tsai, and K. Beznosov. An Approach for Modeling and Analysis of Security System Architectures. *IEEE Transactions on Knowledge and Data Engineering*, 15(5):1099– 1119, Sept/Oct. 2003.

[4] P. T. Devanbu and S. Stubblebine. Software Engineering for Security: A Roadmap. In *The Future of Software Engineering. Special volume of the proceedings of ICSE'2000*, pages 227–239, June 2000.

[5] D. Garlan. Software Architecture: A Roadmap. In *The Future of Software Engineering. Special volume of the proceedings of ICSE'2000*, pages 91–101, June 2000.

[6] D. Garlan, S.-W. Cheng, and A. J. Kompanek. Reconciling the Needs of Architectural Description with Object-Modeling Notations. *Science of Computer Programming. Special Issue on Unified Modeling Language*, 44(1):23–49, July 2002.

[7] C. Hofmeister, R. L. Nord, and D. Soni. Describing Software Architecture with UML. In *Proc. of the 1st Working IFIP Conf. on Software Architecture (WICSA1)*, Feb. 1999.

[8] C. D. Jensen. Secure Software Architectures. In *Proceedings of the 8th Nordic Workshop on Programming Environment Research*, pages 239–246, Aug. 1998.

[9] J. Jrjens. UMLsec: Extending UML for Secure Systems Development. *UML 2002, LNCS 2460, Springer-Verlag*, 2002.

[10] L. Kagal. *Rei: A Policy Specification Language*. PhD thesis, CSEE Department, University of Maryland, Baltimore County, Baltimore, MD 21250, 2005.

[11] T. Lodderstedt, D. Basin, and J. Doser. SecureUML: A UML-Based Modeling Language for Model-Driven Security. *UML 2002, LNCS 2460, Springer-Verlag*, pages 426–441, 2002.

[12] E. Lupu and M. Sloman. Conflicts in Policy-based Distibuted Systems Management. *Transactions on Software Engineering*, 25(6):852–869, Nov. 1999.

[13] N. Medvidovic and D. S. Rosenblum. Assessing the Suitability of a Standard Design Method for Modeling Software Architectures. In *Proc. of the 1st Working IFIP Conf. on Software Architecture (WICSA1)*, San Antonio, TX, Feb. 1999.

[14] H. Muccini, P. Inverardi, and P. Pelliccione. DUALLY: Putting in Synergy UML 2.0 and ADLs. In *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, pages 251–252, LAquila, Italy, Nov. 2005.

[15] S. Myagmar, A. Lee, and W. Yurcik. Threat Modeling as a Basis for Security Requirements. In *Proceedings of the Symposium on Requirements Engineering for Information Security (SREIS'05)*, Paris, France, Aug. 2005.

[16] E. A. Oladimeji and L. Chung. Representing Security Goals, Policies and Objects. In *Proc. of the 5th IEEE/ACIS International Conference on Computer and Information Science (ICIS'06*, pages 160–167, Honolulu, Hawaii, July 2006.

[17] J. J. Pauli and D. Xu. Misuse Case-Based Design and Analysis of Secure Software Architecture. In *Proceedings of the Int. Conference on Information Technology: Coding and Computing (ITCC'05)*. IEEE CS Press, April 2005.

[18] J. Rushby. Security Requirements Specifications: How and What. In *Proc. of the IEEE Symposium on Requirements Eng. for Information Security (SREIS)*, Indianapolis, March 2001.

An Intelligent Agent of Automatically Notify Services

Shuo-Yan Hsu, William C. Chu
TungHai University, Taiwan
E-mail: mohito@itlab.csie.thu.edu.tw, cchu@thu.edu.tw

Abstract

A lifestyle website is an electronic commerce website of providing daily necessities and services. But nowadays, most of them are lack of automatically notifying services and personalized learning mechanism. Therefore, users have to login the website and search suitable services in a large database. This is really a time-wasted and inefficient work.

This paper proposed an extensible structure of the intelligent agent for the lifestyle website. We integrate three techniques to build this structure: (1) Data Cube structure, (2) Bit-Mapping technique, and (3) FP-Tree algorithm. With these techniques, the intelligent agent will not only analyze the user's shopping habits, but automatically notify users of suitable services before the habits happen. With this agent, the lifestyle website can mine more potential customers and let the user feel more convenient.

Keywords: E-Commerce, Partial Periodic Pattern, Intelligent Agent, Push Services

1. Introduction

A lifestyle website is an electronic commerce website providing daily necessities and services. Nowadays, most of the famous lifestyle websites in Taiwan are lack of the "Push" mechanism. Therefore, the user must login the website, and search suitable services in such a large system. This is really a time-wasted and inefficient work.

What is the "Push" mechanism? That is, automatically provide the "right service" to the "right customer" in "right time." Therefore, how to collect user's shopping habits and analyze it correctly and efficiently are the key points to realize it.

This paper proposed an intelligent agent for the lifestyle website. The agent will record the user habits, and automatically notify users of suitable services before the habits happen. For example, Laura has lunch in the restaurant near her home at about 12:10 every noon. After the agent records this habit, it will automatically login the lifestyle website and collect the suitable services for Laura before this habit happens. In order to realize it, the agent must retrieve the user's basic shopping attributes (including *shopping time*, *location*, and *category*) and analyze these attributes in a correct way.

Personal agents are computer programs that can learn users' interests, preferences, and habits and give them proactive, personalized assistance with a computer application [1]. In other words, an agent must have the ability to retrieve basic shopping attributes and translate them to shopping habits in user's shopping transaction list. The shopping habits we called in this paper, is a kind of *Full Periodic Patterns* and *Partial Periodic Patterns*. Both of two patterns belong to Time-Series databases of data mining.

There are a lot of researches about data mining in Time-Series databases [6][7][8][9][10]. Among them, an interesting research [13] was similar to us. The author developed an Apriori-like algorithm to mine imperfect partial periodic pattern. But the Apriori pruning in mining partial periodicity may not be as effective as in mining association rules [11]. We use a more effective approach, FP-Tree [4], to improve the speed of mining. Furthermore, we also use FP-Tree to mine the relations between shopping behaviors to improve the accuracy of predicting habits.

The remaining of the paper is organized as follows. In section 2, we introduce the structure of our system. In section 3, we list two kinds of habits and analyze them in section 4 and 5 respectively. In section 6, we have a discussion for our ideas. Finally, we conclude our study and introduce the future works in section 7.

2. Structure of the System

Figure 1 shows the structure of the system we proposed. There are two ways to provide services for the users. The first way is that the agent receives the users' request "passively", and searches the suitable services from the service profile to notify him. The second way is that when the agent learns the user's habits, it will "actively" notify the user of suitable services before the habits happen. This paper was focus on the personalized learning mechanism and we will not specify other functions in detail. We will discuss them in our future writings.

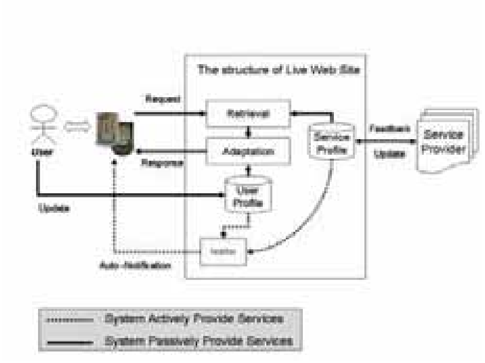


Figure 1: The structure of the system.

The intelligent agent will divide a day into 24 segments (hours). The agent will automatically notify users of suitable services before the old segment pass. Services can be easily pushed to the right users through the agent. In order to realize the "push" ability, the agent must have two mechanisms as follows:

- **Automatic notification mechanism**

  Most of the lifestyle websites are lack of the automatic notification mechanism, so the services were in a passive situation to wait for customers finding. In our system, the agent will use RSS2.0 (Really Simple Syndication 2.0) [12] to notify users.

  There are three notification channels. They are "Advertisement", "User subscribed", and "Personalized" channels. The channel of "Advertisement" and "User subscribed" are not the key points in this paper, so we will not discuss them here. The agent will use personalized learning mechanism to notify users of suitable services through the "Personalized" channel.

- **Personalized learning mechanism**

  The shopping habits we proposed, including **V**ertical **H**abits (VH) and **H**orizontal **H**abits (HH). Vertical habits are the periodic shopping habits and horizontal habits are the relations between shopping behaviors. For example, Laura has lunch at 12:10 every Monday (VH), and she likes to buy drinks after lunch except Monday (HH).

  After integrating vertical habits and horizontal habits, the agent will notify Laura of drinks' services after she finished lunch every Monday. The purpose of this approach is to reduce the deviation when the user had a temporary trip and improve the accuracy of predicting habits.

  In order to realize this mechanism, we integrated three techniques to build it: (1) Data Cube structure, [5][13] (2) Bit-Mapping technique, and (3) FP-Tree algorithm. It shows that data cube structure provides an efficient and effective structure for on-line analytical processing (OLAP) and on-line analytical mining [5]. As to FP-Tree algorithm, it mines frequent patterns without candidate generation. It is also more effective than Apriori algorithm [4].

  We retrieve user's vertical habits based on data cube structure, and use FP-tree to retrieve horizontal habits.

3. Vertical Habits (VH) and Horizontal Habits (HH)

Vertical habits are the periodic shopping habits. We denote VH = (Day, B_i). B_i is the number of shopping **B**ehaviors. "Day" is the day when the shopping habits happened, where Day = {1, 2, …, 7}.

Horizontal habits are the relations between shopping behaviors. We denoted HH = (B_i, {B_j}), where B_j is a set of shopping behaviors, $i \neq j$, $i > 0$, $j \geqq 0$ ($j = 0$ means that there are not any related behaviors with B_i)

After integrating horizontal habits and vertical habits, we can call it **C**yclic **P**attern, CP=(Day, {BH} ∪ {HH}). The cyclic patterns are also the shopping habits we called in this paper.

4. Mining Vertical Habits – Using Data Cube-based Structure

In this section, we discuss how the agent mining vertical habits with data cube-based structure after it record basic shopping attributes (location, category and time).

We construct two data cubes, *template cube* and *mining cube*. Template cube is used to record users' basic attributes and mining cube is used to mine vertical habits.

4.1. Template Cube

Template cube is a 4-D data cube. It records the basic shopping attributes, including *shopping location*, *category of services*, *shopping segments*, and *shopping day*. The concept hierarchies for these attributes are as follows:

- **Shopping location:** shopping district→city
- **Category of services:** food, clothes, live, traffic, education, entertainment, and the others.
- **Shopping segments:** 24 hours
- **Shopping day:** day→week→month→year.

The number of the cell represents the number of **B**ehaviors (B_i), where B_i = (Location, Service, segment, Day), i = 1, 2, 3, …, n, n > 0, and n is the total number of the cells.

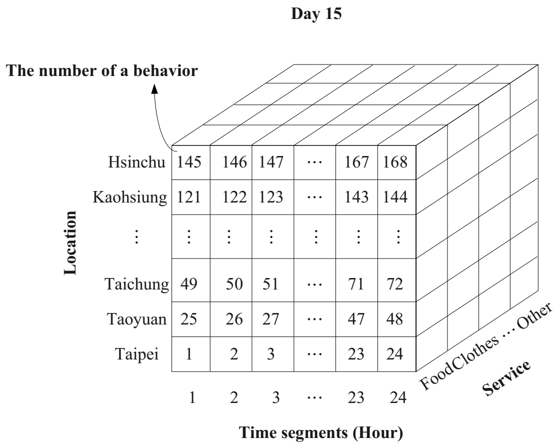


Figure 2: Template Cube

Figure 2 is the template cube based on shopping environment in Taiwan. For example, B_{145} = (Hsinchu, Food, 1, 15). When the agent record B_{145}, it shows that the user *have a night snack* at *1:00* in *Hsinchu* in *Day 15*. After retrieving the shopping information, we can mine vertical habits using mining cube.

4.2. Mining Cube [13]

Mining cube, as **Figure 3**, is used to mine vertical habits. We fold time dimension into two parts: *time-index* dimension and *period-index* dimension. Notice that each cell needs only a bit (existent, nonexistent). Each slice of the mining cube can be implemented as a bit-array, except the last slice, *period-index = All*, which contains the number of nonzero bits of all weeks slices and each cell is an integer.

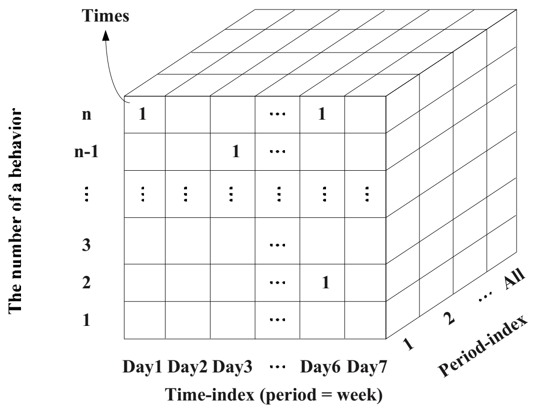


Figure 3: Mining Cube.

Through mining cube, we can easily record the shopping behaviors. But in the real world, a habit will not be happened with 100%. So we must introduce the concept of *confidence* to tolerate misses. We denote *confidence* as α, *total number of period-index* as β, and *minimum support* as α × β. The vertical habits must no less than minimum support.

For example, as **Figure 4**, if we assume α = 50% and β = 4 (weeks), we can figure out that minimum support = 2 (50% × 4). Now, only (Day1, B_n), (Day3, B_{n-1}) and (Day6, B_2) can pass the threshold, and they are the vertical habits.

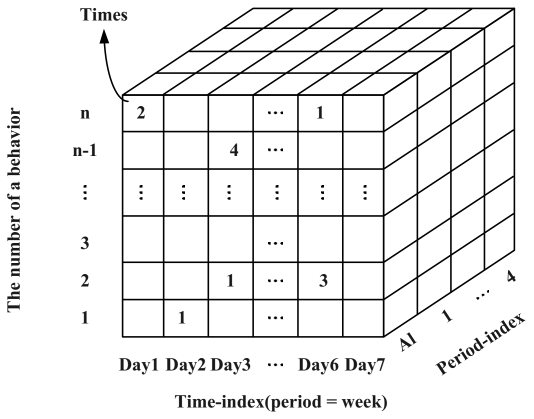


Figure 4: Mining cube where period-index = All

5. Mining Horizontal Habits – Using FP-Tree Algorithm

After analyzing vertical habits, we almost know the user's habits. But if user has a temporary trip, it will induce the deviation. In order to solve this problem, we use FP-Tree to retrieve horizontal habits and improve the accuracy of predication.

We made a table for example, as **Table 1**.

Table 1: An example of vertical habits

Time Period	The number of a Behavior
Day1	B_1, B_2, B_5
Day2	B_2, B_4
Day3	B_2, B_3
Day4	B_1, B_2, B_4
Day5	B_1, B_3, B_5
Day6	B_1, B_2, B_6
Day7	B_1, B_2, B_3, B_5

We assumed that the minimum support is 3. First, we scan **Table 1** and derive **Table 2** after deleting the infrequent behaviors (less than 3). Through **Table 2**, we derive a list of frequent items, {B_2: 6, B_1: 5, B_3: 3, B_5: 3}, and translate it to FP-Tree, as **Figure 5**.

Table 2: The frequent behaviors of vertical habits

Time Period	The number of a Behavior
Day1	B_2, B_1, B_5
Day2	B_2,
Day3	B_2, B_3
Day4	B_2, B_1
Day5	B_1, B_3, B_5
Day6	B_2, B_1
Day7	B_2, B_1, B_3, B_5

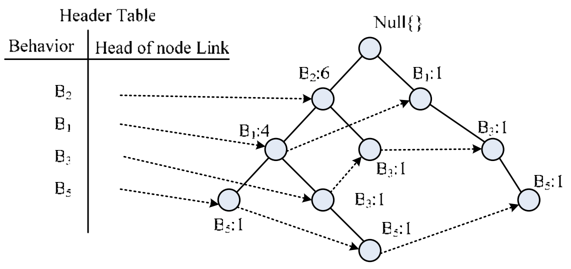


Figure 5: The FP-Tree in Table 2

The conditional pattern bases and the conditional FP-Trees generated are summarized in **Table 3**. Through **Table 3**, we found the horizontal habits, HH = (B_1, B_2). Notice that through template cube, we can know which behaviors happened earlier.

Table 3: Mining of all-patterns by creating conditional (sub) - pattern bases

Behavior ID	Conditional Pattern Based	Conditional FP-Tree	Sequence Pattern
B_5	{(B_2 B_1:1), (B_2 B_1 B_3:1)}	Null	null
B_3	{(B_2 B_1:1), (B_2:1), (B_1:1)}	Null	null
B_1	{(B_2:4)}	(B_2:4)	B_2 B_1:4

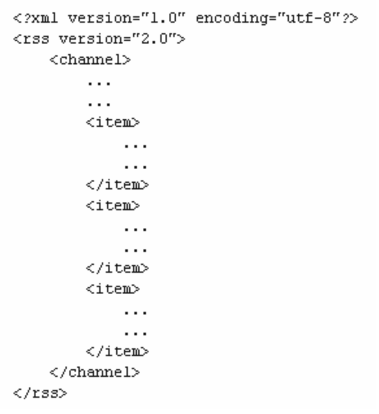
After integrating vertical habits and horizontal habits, we modify the habits of "Day 5", translate to cyclic patterns, as **Table 4**, and it is the final habits we proposed.

Finally, after we retrieve the shopping behaviors (CP), we will automatically notify the user of suitable services. How to automatically notify a user? Because we know the user' shopping behavior hour and the day, the server will automatically generate the RSS Seed in the personalized channel.

Figure 6 shows a template structure of a RSS 2.0 seed. From the specification of RSS2.0, we can know the seed's information [12]. We can know what kind of services the seed belong by the **category** element and what time it broadcast by the **pubDate** element. With these two elements, we can easily search the seed of right category service and right time satisfying the user's habits.

Table 4: Cyclic Patterns

Cyclic Pattern(CP)
Day1, B_1, B_2, B_5
Day2, B_2, B_4
Day3, B_2, B_3
Day4, B_1, B_2, B_4
Day5, B_1, $\boldsymbol{B_2}$, B_3, B_5
Day6, B_1, B_2, B_6
Day7, B_1, B_2, B_3, B_5



```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
    <channel>
        ...
        ...
        <item>
            ...
            ...
        </item>
        <item>
            ...
            ...
        </item>
        <item>
            ...
            ...
        </item>
    </channel>
</rss>
```

Figure 6: A template structure of a RSS2.0 seed

Take Figure 2 for an example. After we have analyzed the user's shopping behaviors, we found that the user will need the food information in Taipei about 1:00 am in Taipei. The server will automatically search the RSS 2.0 seed generated from the store, and rebroadcast in personalized channel before 1:00. So, the agent combined RSS reader function will receive the seed, and notify the user of food information in Taipei before 1:00 am in Day1.

With this mechanism, users will not need to waste time searching for services in large databases, and we can let stores have more opportunities to publish their services.

6. Discussion

After integrating template cube and mining cube, we successfully retrieve the vertical habits. But, why we use the data cube-based structure? It is because that it has high extension. For example, we can change 24 segments to 48 or 12, and so do the location, category, and day. Moreover, not only the high extension, but it provides an efficient and effective structure for on-line analytical processing (OLAP) and on-line analytical mining.

Beside data cubed-based approach, we also use FP-Tree to find the horizontal behaviors. But we must remind that we use FP-Tree just because it is an efficient approach. If there is a new approach faster than it, we can adapt the new one to improve the prediction's efficiency. It is the most variable advantage of our high extensible structure.

7. Conclusion and Future Work

We proposed an intelligent agent for the lifestyle website. The intelligent agent will not only analyze the user's shopping habits, but automatically notify users of suitable services before the habits happen.

We also construct a personalized learning mechanism with high extension. In other words, we can easily modify and adapt to it.

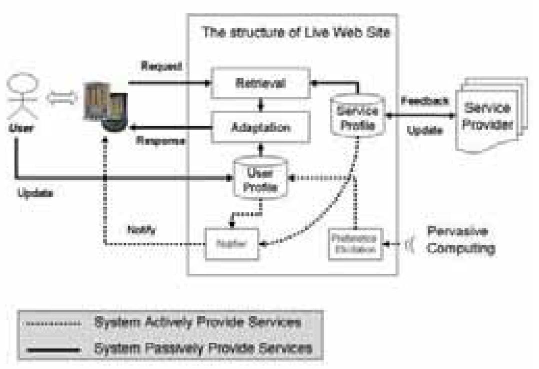


Figure 7: The structure of whole system in the future

As **Figure 7**, we hope to integrate pervasive computing to retrieve users' behaviors and construct a preference elicitation rule to elicit users' shopping habits.

8. References

[1] S. J. Soltysiak and I. B. Crabtree, "Automatic Learning of User Profiles — Towards the Personalisation of Agent Services," *BT Technology Journal*, Vol. 16, No. 3, pp.110-117, July 1998.

[2] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," *Proceedings of the 20th International Conference on Very Large Data Bases*, pp. 487–499, September 1994.

[3] R. Agrawal and R. Srikant, "Mining sequential patterns," *Proceedings of the 11th International Conference on Data Engineering*, pp. 3–14, March 1995.

[4] J Han, J Pei, and Y Yin, "Mining Frequent Patterns without Candidate Generation," *Proceedings of the 2000 ACM SIGMOD International Conference on Management of data*, pp. 1-12, 2000.

[5] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, 1998.

[6] R. Agrawal, C. Faloutsos, and A. Swami, "Efficient Similarity Search in Sequence Databases," *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms*, pp. 69-84, October1993.

[7] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast Subsequence Matching in Time-Series Databases," *ACM SIGMOD Record*, Vol. 23, No. 2, pp. 419-429, June 1994.

[8] K. Chan and A. Fu, "Efficient Time-Series Matching by Wavelets," *Proceeding of the 15th International Conference on Data Engineering*, pp.126-133, March 1999.

[9] H. Mannila, H. Toivonen, and A. Verkamo, "Discovering Frequent Episodes in Sequences," *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pp. 210-215, 1995.

[10] D. Rafiei, "On Similarity-Based Queries for Time-Series Data," *Proceeding of 15th International Conference on Data Engineering*, 1999.

[11] J. Han, G. Dong, and Y. Yin, "Efficient mining of partial periodic patterns in time series database," *Proceeding of 15th International Conference on Data Engineering*, Sydney, Australia, Mar. 1999.

[12] Hammond, T, Hannay, and B. Lund, "The Role of RSS in Science Publishing," *D-Lib Magazine*, Vol. 10, No. 12, December 2004.

[13] J. Han, W. Gong, and Y. Yin, "Mining segment-wise periodic patterns in time-related databases," *Proceeding of 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pp. 43-52, August 1998.

A Proposal for a Decentralized Multi-Agent Architecture for Virtual Enterprises

Andreas Grünert, Sven Kaffille, and Guido Wirtz
Distributed and Mobile Systems Group
University of Bamberg
Feldkirchenstraße 21, 96052 Bamberg, Germany
andreas.gruenert@atkearney.com,
{sven.kaffille|guido.wirtz}@wiai.uni-bamberg.de

Abstract

Today, enterprises need more flexibility than ever before, but classic companies often cannot meet this requirement. Therefore, a more flexible kind of enterprise evolved: The Virtual Enterprise. One reason why Virtual Enterprises are that flexible is their extensive use of IT. Yet, given the temporary character of Virtual Enterprises, existing software concepts are often not applicable. Multi-Agent Systems (MAS) may help to solve this problem. Actually, MAS based solutions have already been developed for diverse problems (mainly) of the Virtual Enterprise performance layer. However, a concept that allows cooperating agents to support the management of the complete life-cycle of a Virtual Enterprise, is still missing.
This paper attends to that gap. It defines an agent supported Virtual Enterprise life-cycle and proposes an abstract architecture for organizing a virtual enterprise based on a structured, multi-layered organization of agents. In doing so, the infra-structure for the next steps, i.e. elaborating the detailed ontologies, is provided.

Keywords: virtual enterprise, agents, architecture

1. Introduction

Today's enterprises are faced with a quickly changing global environment. They have to be highly flexible and adaptable to survive in a business world marked by trends like globalization, increasing customer orientation and product variety, and demand for shorter product life-cycles as well as shorter time-to-market. On the other hand surging product, organizational and environmental complexity hinder fexible and adaptive behavior as e.g. finding skilled employees or reducing the size of an enterprise quickly (cf. [21]). For these reasons so called Virtual Enterprises (VE) evolved, which are -for short- a temporary cooperation of independent enterprises with specific capablities and resources. What these enterprises require most, are tools that support their quick formation and dissolution, and ensure that the different enterprises' heterogeneous business and information technology is integrated in an acceptable way for the time the VE exists. While the business world has to become more flexible and adaptable the demand for flexible and adaptable software architectures increased as well. This has been addressed in information and computer science with help of new concepts and technologies. One of these is the concept of Multi-Agent Systems (MAS), in which autonomous and heterogeneous entities -the agents- interact. For MAS, theories for flexible cooperation (e.g. [20, 22]) as well as technologies and standards [5] have been developed. The goal of our work is to develop a MAS-based infrastructure to support VEs during their whole life cycle. For this purpose we propose an abstract architecture for VEs developed from the requirements the life cycle of a VE imposes on such an architecture.
This paper first develops the conceptual and technological foundations of VEs including a concise model of their life cycle as well as the corresponding structures and processes in MAS in section 2. In section 3, the abstract architecture for VEs is presented. Afterwards, our approach is evaluated against existing approaches in section 4. The paper concludes with some remarks on future work.

2. Conceptual and Technological Foundations

The foundations of our work stem from three main fields: economics, MAS theory, and MAS technology. Therefore this section describes the basic concepts from these fields, which have been incorporated into our work.

2.1. Virtual Enterprises

There is no uniform definition for VEs and there is no uniform approach to define them. Davidow and Malone define Virtual Enterprises via their organizational structure as *"a nearly contourless formation with permeable and permanently changing dividing lines between enterprise, suppliers and customers"* (Translation from [2, p. 15]). Others define Virtual Enterprises via the properties and preconditions of the emerging cooperation (e.g. [6]). Yet, looking at the individual definitions, as diverse as they may be, there are several aspects many authors agree on. Therefore, this paper works with an "assembled" definition of Virtual Enterprises with attributes and characteristics of VEs as follows:

- Cooperation network of legally and economically independent enterprises
- Federation based on a shared business understanding
- Motivated by customer needs and market opportunities
- Focus on core competencies for each partner
- One face to the customer
- Renunciation of an extensive legal framework
- Aimed at temporary collaboration
- Ad hoc formation
- No institutionalized central management functions
- Cooperation in horizontal and vertical structure
- Enabled by information/communication technology

As a VE is a temporary organization to exploit market opportunities, the VE vanishes when the market opportunity has been fully exploited or does not exist anymore. Therefore a VE has a life cycle like the products and services it produces. In literature [6, 13, 14], VEs with three, four and five phases have been proposed. Our work is based on a life cycle with five phases, which are related in an unambiguous conditional and chronological way and are executed sequentially with each phase establishing the preconditions for its successor. These phases are:

1. *Identification*: In this phase a market opportunity is identified by one company (or many companies in parallel). This company specifies a value chain and the activities that have to be performed. Then it performs a cost-benefit analysis and a competency analysis. Afterwards it decides whether the market opportunity should be exploited and a VE should be formed.
2. *Formation*: At the beginnnig of the formation phase requirements profiles are derived from the competency analysis. Based on this profiles potential partners are selected. Between the best fitting ones and the initiating company the formation of a VE is then negotiated.
3. *Design*: After a suitable group of enterprises has been identified and the individual companies have agreed to form a VE, a common strategy has to be agreed on. Then performance measures, the basic legal and financial framework, and operating structure and infrastructure are established and created. These tasks comprise e.g. protection of intellectual property of companies as well as assignment of value chain activities and roles to the individual companies. A plan for the dissolution has to be elaborated to determine the preconditions for the VE's dissolution and to prevent from legal or technological issues afterwards. After an agreement is reached the single enterprises commit to the VE and start operation.
4. *Operation*: During this phase the VE performs business according to the plans and the framework elaborated in the preceding phases. In this phase the VE operates similar to a single traditional enterprise but the coordination of the individual enterprises is more complex, as it crosses the boundaries of single enterprises. Therefore the VE has to emphasize on flexibility and adaptability in its coordination mechanisms. Another important issue is to emphasize on customer needs and to provide one face to the customer to successfully exploit the market opportunity.
5. *Dissolution*: When the preconditions agreed on for the dissolution of the VE are fulfilled it dissolves. In this phase shared know-how and data has to be distributed among the partners. Legal issues as e.g. how future warranty claims of customers are handled must be addressed. These issues should have been sorted out in the dissolution plan developed during *Design*.

This process has not necessarily to be performed in a simple sequence as the partners may decide to go back to an earlier phase. Thus, the VE life cycle may become iterative.

2.2. Cooperation and MAS

A Multi-Agent System (MAS) is according to [11] „a loosely coupled network of problem solvers that work together to solve problems that are beyond the individual capabilities or knowledge of each problem solver". A MAS can be constructed (i) with help of a top-down/divide-and-conquer approach or by (ii) a bottom-up approach by composing individual agents or groups of agents with different capabilities and knowledge. For (i) all problem solving and communication strategies become a hard-wired integral part of a MAS. Whereas approach (ii) requires coordination and negotiation between agents to achieve a solution for their problem(s). A MAS can be described by the following characteristics [19]:

- Each agent has limited capabilities and knowledge to solve problem(s) for itself
- There exists *no global control*.
- Data storage is *decentral*.
- Computation is *asynchronous*.
- The configuration of a MAS is *heterogeneous* or *homogeneous* regarding the agents.
- A MAS can be *closed or open*.
- A MAS is expected to operate in a highly *dynamic environment*.

2.2.1. Teamwork in MAS

The limited capabilities and knowledge of agents to solve problems on their own makes cooperation necessary, which can be structured in MAS with help of the Cooperative Problem Solving (CPS) process ([22], p. 576). CPS is divided into four phases: recognition, team formation, plan formation, and team action.

1. *Recognition*: At least one agent must recognize that there is the need or potential for cooperation. Furthermore a group of agents, that can cooperate must be identified.
2. *Team Formation*: In this phase the agents try to let all other agents know that there is a cooperation possible. If this phase is successful a potential team has been found.
3. *Plan Formation*: Now the agents try to find a common plan to achieve their goal(s). If they find and agree upon a common plan they engage into a joint commitment. They promise each other to carry out their plan. If there is no agreement on a common plan the cooperation fails.
4. *Team Action*: In this last phase the agents carry out the plan agreed upon before.

As CPS is an iterative process, this phases have not to be processed sequentially. The benefit of agent cooperation according to CPS is the added flexibility that results from the definition of plans and assignment of tasks/roles. The joint effort of the team is monitored in a decentral manner by all agents. To facilitate this a shared mental state and joint commitments are established to flexibly react to anticipated and unanticipated plan failures [20]. The commitments can be parameterized by means of conventions [22], that specifiy what actions must be performed when anticipated and unanticipated plan failures occur.

2.2.2. The FIPA Standard

In order to facilitate the interaction between heterogeneous agents the Foundation of Intelligent Physical Agents (FIPA)[1] has developed a set of standards. These standards are mainly concerned with agent management and communication and define an abstract architecture for MAS platforms [5]. The abstract architecture requires a MAS platform to provide yellow pages (directory facilitator), white pages (agent management system) and communication services. Agent communication and these services are standardized in a platform independent manner. Agents communicate with help of messages formulated in an agent communication language (ACL), which has a 3-layer architecture. On the first layer, the content language resides, which defines the syntax of messages. The second layer defines the semantics of the terms used on the first layer with the help of an ontology. The third layer defines the interpretation of messages utilizing speech acts. The standard mainly defines standard interaction protocols based on speech acts, ontologies and content languages that must be implemented by the directory facilitator and agent management system. The basic content language, which all FIPA-compliant agents must support is FIPA-SL.

[1]`http://www.fipa.org/`

3. The Abstract Architecture

In this paper we propose an abstract architecture that has been designed to support the management of a virtual enterprise. This management is completely distributed between the individual enterprises, which must coordinate and cooperate to create a virtual management layer. The requirements which have been imposed on the abstract architecture are:

- The life-cycle of a VE corresponds to a CPS process, but just mapping them one to one is too simplisitic, as an enterprise consists of many multi-agent organizations (MAO).
- An enterprise has many interfaces and connections to other enterprises in the VE and has to perform many tasks. Therefore an enterprise cannot be represented by just one agent, as this view is to coarse grained.
- Not all tasks relevant for VEs can be executed by artificial agents but humans as well as robots have to be integrated.
- A problem is that a VE and its corresponding MAS are created bottom-up rather than top-down. Therefore many characteristics of a VE are specified at runtime (either by software agents or humans) as e.g. the plans of VE operation. For this reason the abstract architecture has to fix the structure and behavior for a MAS that facilitates negotiation and coordination between VEs.
- Standard coordination tasks should be performed automatically by software agents.
- The abstract architecture should not impose too narrow constraints on the internal structure and behavior of VEs and their MASs.

Our architecture is divided into two parts: The VE Management MAS (VEMMAS) and the VE Performance MAS (VEPMAS). The former is responsible for the management tasks of the VE and the latter for the production of services and products provided by a VE.
The main components of a MAS for Virtual Enterprise lifecycle support on the VEMMAS layer are the Partner Selection MAO, the Production Planning MAO, and the Data Distribution MAO. As a Partner Selection MAO only encompasses agents of one enterprise, each partner enterprise may actually posses its own. Under certain circumstances these MAOs may cooperate and form a Virtual Enterprise spanning Partner Selection MAO. The two remaining MAOs are

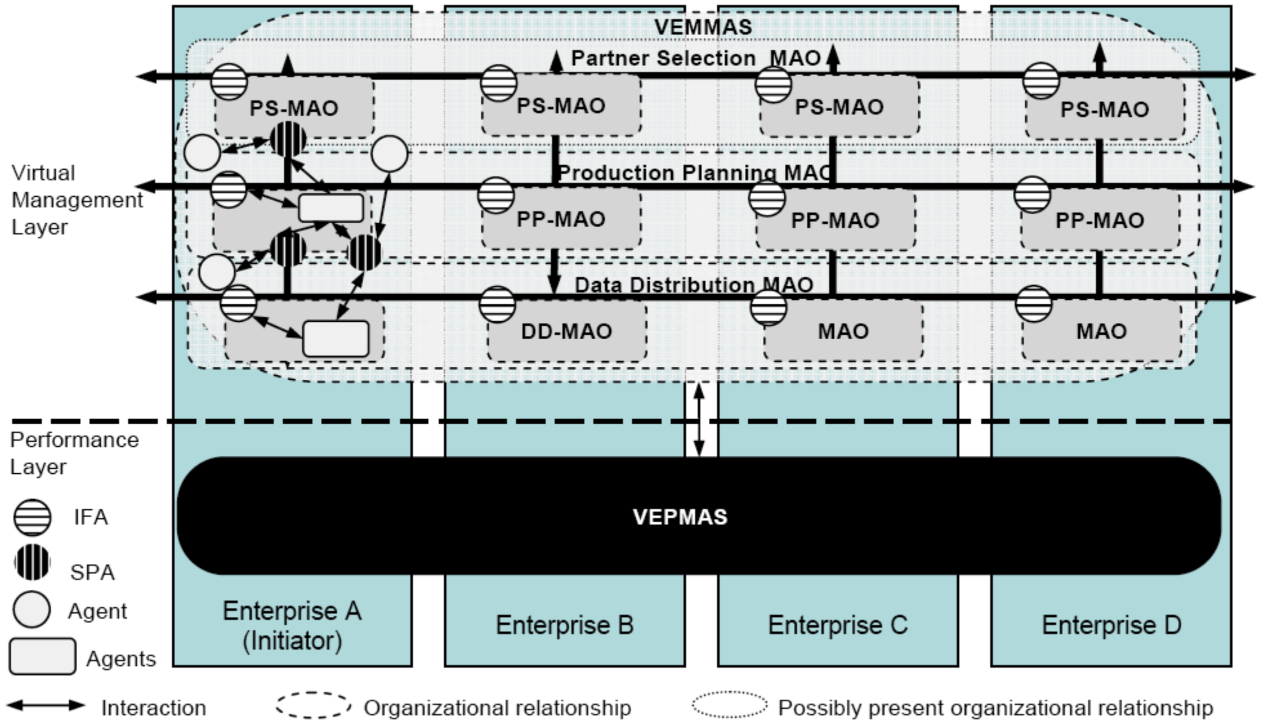


Figure 1. Proposed Architecture of VE*MAS

formed by agents of different enterprises. In the following, the Production Planning MAO, the Data Distribution MAO and the Virtual Enterprise spanning Partner Selection MAO will be called first-level MAOs.

Agents of individual partner enterprises are distinguished into second-level MAOs based on the first-level MAOs (e.g. the Production Planning MAO) they are part of. That means, for example, that agents of an enterprise that are part of the Production Planning MAO form a different second-level MAO than those agents of the same enterprise that are part of the Data Distribution MAO. Interactions between agents occur only along two dimensions, within individual enterprises and within the individual first-level MAOs. Thus, the number of interfaces between different enterprises and between different first-level MAOs is reduced.

The second-level MAOs encompass at least two types of agents. One Interface Agent (IFA) encapsulates a MAO from the point of view of a MAO of another enterprise. Second-level MAOs of different enterprises that form a first-level MAO together interact via these Interface Agents. Service Provider Agents (SPA) encapsulate a second-level MAO (and simultaneously the first-level MAO the second-level MAO is part of) from the point of view of other agents of the same enterprise. Via the SPAs it is possible for other agents to request the (re-)execution of certain roles.

Summarized, second-level MAOs of different enterprises that form a first-level MAO interact via IFAs, whereas agents of one enterprise that are part of different MAOs interact via SPAs. Agents that are not part of one of the mentioned MAOs can access these via the corresponding SPAs of their own enterprise. Thus, the demand that interactions may only happen along two dimensions is met.

Within these first-level MAOs CPS processes are carried out across enterprise boundaries. Within these processes the tasks (identified on a high level) described in section 2 can be assigned to different enterprises (repectively the second-level MAOs). Still missing are the agents that perform the tasks of a second-level MAO. Of what type these agents are, how many of them exist within the individual MAO, how they interact, and how they complete the tasks is of no interest for the abstract architecture but most likely they will engage in a CPS process, as well.

The encapsulation of a second-level MAO by its IFA and SPAs is particularly useful with regard to the Partner Selection MAOs. If an enterprise needs to request the selection of new potential partners it is completely sufficient if the enterprise has a SPA that receives the request from agents within the enterprise, and an IFA that can “execute” that request by delegating it to the Partner Selection MAO of another enterprise. There is no need for the enterprise to have a complete Partner Selection MAO established itself.

Each partner enterprise is responsible for its own agents and therefore needs at least one agent platform as described by the FIPA standard. An enterprise may also provide different platforms for agents that belong to different MAOs but this is of no concern for the abstract architecture. The same is true for how agents of an enterprise know what SPAs to use and how to contact them. This could be done via the directory service of an agent platform, but as already said, this is of no concern for the abstract architecture because each enterprise can use an individual solution as long as its agents know what SPAs to use and how to contact them.

In contrast to this, how IFAs of different enterprises can find each other so that the first-level MAOs can be formed is of great interest for the abstract architecture. The IFAs of an enterprise are registered at the local agent platform so that IFAs of other enterprises could ask that platform where they can find a certain IFA. Nonetheless, the abstract architecture proposes to use specialized agents to provide this information. All enterprises must have a so called Interconnection Agent (ICA) whose contact information is exchanged when the Virtual Enterprise is formed. These agents hold and exchange the contact information of the IFAs of an enterprise and of which first-level MAOs these are a part of.

Introducing ICAs is advantageous for two reasons. First, an enterprise may operate several agent platforms. When looking for a certain IFA of an enterprise, several platforms would have to be asked requiring that all these platforms are known. As each enterprise has exactly one ICA, only the contact information of one agent has to be exchanged when the Virtual Enterprise is formed. Second, using ICAs allows that changes concerning an IFA can be distributed actively to the IFAs of other enterprises. An agent platform could distribute those changes only passively. Together, the ICAs of all partner enterprises form a first-level MAO as well.

In order to implement this abstract architecture first of all, the components that are specified as mandatory by the FIPA standard have to be implemented. This includes for example an agent platform, an agent directory service and a message transport service. Next, second-level MAOs have to be implemented that will become part of the Production Planning MAO and the Data Distribution MAO of the Virtual Enterprise. The constituent parts of a second-level MAO are one IFA, one to several SPAs and the agents that actually perform the tasks of the MAO. The IFA and the SPAs have to be implemented. The other agents have to be present but might as well be human agents with a suitable interface to the MAO.

In addition, a second-level Partner Selection MAO has to be implemented. If the enterprise does not intend to function as an initiating enterprise, it is sufficient if this MAO encompasses one IFA and one to several SPAs. Agents that actually perform a partner selection are only necessary if the enterprise intends to initiate the formation of a Virtual Enterprise. Even then, these agents have not necessarily to be implemented because they could be human agents as well.

Finally, every enterprise needs an implemented ICA.

4. Related Work

The work presented here, proposes to use software agents in the context of virtual enterprises for partner selection, production planning and data distribution. Zheng and Zhang [23] propose an artificial marketplace, a negotiation protocol, and a bid selection algorithm that allow agents to form a virtual organization. Petersen and Divitini [15], e.g., propose agent-based mechanisms to form Virtual Enterprises, and Petersen and Matskin [16] developed agent interaction protocols for partner selection. Besides the seminal work of Jennings and Wooldridge on how to control Cooperative Problem Solving in industrial Multi-Agent Systems [22], lots of approaches use agents for dynamic and distributed process management [10, 17]. In terms of information sharing, Dutta et al. [4] have developed cooperative information sharing strategies to support distributed resource allocation, Decker et al. [3] have published considerations about how to design the behavior of information agents, and [1] propose a federated information management for cooperative virtual organizations.

The amount of related work justifies the basic decision to use agent technology for VEs but also demonstrates the need for an architecture in order to structure these efforts. Proposals for complete agent based architectures dedicated to virtual enterprises, however, are rarely to be found.

Although a well-developed model to describe team cooperation involving software agents as well as robots and humans using a proxy-architecture, the Machinetta framework[2] is not specifically taylored towards virtual enterprises [18]. In order to implement an architecture as presented in section 3, many adjustments on the intra- as well as inter-enterprise level are required. Additionally, the system is not FIPA-compliant and it cannot be assumed that each enterprise participating in such an architecture uses Machinetta.

The CONOISE (CONOISE-G) project is closely related to our approach in its efforts to employ agent-based models and techniques to automatically form and operate virtual organizations in general or in the specific context of grid computing, respectievely [12]. The system proposes a number of agents for, e.g., service providers, yellow pages, quality and clearing as well as quality of service, policy and reputation monitoring. The CONOISE-G architecture, however, is inherently different from the approach presented here by using a single managing agent. Thus, the system only supports a centralized broker architecture. The idea of our abstract architecture is to distribute the load of managing the VE among the participating enterprises, which first try to solve problems occuring during the life-cylce locally and if this is not possible, coordinate with the other enterprises according to their commitments and conventions.

[2]`http://teamcore.usc.edu/doc/Machinetta/`

For a more rigorous comparison of these approaches with our architecture, please refer to [7].

5. Conclusion and Future Work

In this paper a first step towards a uniform and flexible decentralized MAS architecture for supporting VEs and its internal structuring mechanisms has been proposed. The details about the single interaction protocols that are necessary to assign roles and negotiate commitments and conventions have been ommitted due to space limitations but can be found in [7]. Our idea regarding protocols is to propose meta-protocols to select interaction protocols e.g. for planning and negotiation of commitments and conventions. Some additional developments towards this are underway, e.g., in [9] a meta-protocol for service level agreements has been developed.

Closely related to the development of interaction protocols is the development of ontologies that define the important concepts for VEs in much more detail than the overall structure presented here. Required ontologies have to comprise ontologies for products and services as well as actions, commitments, and conventions of agents. One of the next steps will be the evaluation of existing ontologies whether they fit our requirements. Furthermore, the detailed structure of the performance layer, i.e., the VEPMAS and its interfaces to the management layer VEMMAS have to be investigated in more detail. Moreover, the abstract architecture will be extended by integrating a reputation management scheme as proposed in [8] much like the one of CONOISE [12] but taylored towards our decentralized approach.

References

[1] H. Afsarmanesh and L. M. Camarinha-Matos. Federated information management for cooperative virtual organizations. In *Proceedings of DEXA97 - International Conference On Data Bases and Expert Systems Applications*, 1997.

[2] W. H. Davidow and M. S. Malone. *Das virtuelle Unternehmen - Der Kunde als Co-Produzent*. Campus Verlag, Frankfurt/Main, New York, 1993.

[3] K. Decker, A. Pannu, K. P. Sycara, and M. Williamson. Designing behaviors for information agents. *Proceedings of the First International Conference on Autonomous Agents*, 1997.

[4] P. S. Dutta, N. R. Jennings, and L. Moreau. Adaptive distributed resource allocation and diagnostics using cooperative information sharing strategies. In *Proceedings of 5th International Conference on Autonomous Agents and Multi-Agent Systems*, 2006.

[5] FIPA TC Architecture. Fipa abstract architecture specification. Technical report, FIPA - Foundation for Intelligent and Physical Agents, 2002.

[6] H. T. Goranson. *The agile virtual enterprise: cases, metrics, tools*. Quorum Books, Westport, 1999.

[7] A. Grünert. Life-cycle assistance capabilities of cooperating Software Agents for Virtual Enterprises - evaluation and concept development for selected phases. Technical Report 71, Bamberg University, 2007.

[8] A. Grünert, S. Hudert, S. König, S. Kaffille, and G. Wirtz. Decentralized reputation management for cooperating software agents in open multi-agent systems. *International Transactions on Systems Science and Applications*, 1(4):363–368, 2006.

[9] S. Hudert, H. Ludwig, and G. Wirtz. A negotiation protocol framework for ws-agreement. In *KIVS 2007*, 2007.

[10] A. K. Jain, I. V. A. Manuel, and M. P. Singh. Using agents for process coherence in virtual enterprises. *Communications of the ACM*, 42(3):62–69, 1999.

[11] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[12] T. J. Norman, A. Preece, S. Chalmers, N. R. Jennings, M. Luck, V. D. Dang, T. D. Nguyen, V. Deora, J. Shao, A. Gray, and N. Fiddian. Agent-based formation of virtual organisations. *International Journal of Knowledge Based Systems*, 17(1-2):103–111, 2004.

[13] M. Oprea. Coordination in an agent-based virtual enterprise. *Studies in Informatics and Control*, 12(3):215–225, 2003.

[14] H. V. D. Parunak. Technologies for virtual enterprises. *Agility Journal*, 1997.

[15] S. A. Petersen and M. Divitini. Using agents to support the selection of virtual enterprise teams. In G. W. Paolo Giorgini, Yves Lesprance and E. S. K. Yu, editors, *Proceedings of the Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2002 at AAMAS*02)*, pages 98–112, Bologna, 2002.

[16] S. A. Petersen and M. Matskin. Agent interaction protocols for the selection of partners for virtual enterprises. In V. Marik, J. Mller, and M. Pechoucek, editors, *Multi-agent Systems and Applications III, 3rd International Central and Eastern European Conference on Multi-Agent Systems, CEEMAS 2003*, volume LNAI 2691, pages 606–615. Springer-Verlag, June 2003.

[17] N. R. J. S. Bussmann and M. Wooldridge. *Multiagent systems for manufacturing control: A design methodology*. Springer Verlag, 2004.

[18] P. Scerri, D. Pynadath, N. Schurr, A. Farinelli, S. Gandhe, and M. Tambe. Team oriented programming and proxy agents: The next generation. In *Proceedings of 1st International Workshop on Programming Multiagent Systems*, 2004.

[19] K. P. Sycara. Multiagent systems. *AI Magazine*, 19(2):79–92, 1998.

[20] M. Tambe and W. Zhang. Towards flexible teamwork in persistent teams: Extended report. *Autonomous Agents and Multi-Agent Systems*, 3:159–183, 2000.

[21] H. B. Walter Gora. *Virtuelle Organisationen im Zeitalter von E-Business und E-Government - Einblicke und Ausblicke*. Springer Verlag, Berlin, Heidelberg, New York, 2001.

[22] M. J. Wooldridge and N. R. Jennings. The cooperative problem solving process. *Journal of Logic and Computation*, 9(4):563–592, 1999.

[23] Q. Zheng and X. Zhang. Automatic formation and analysis of multi-agent virtual organization. *Journal of the Brazilian Computer Society (JCBS)*, 11(1):74–89, 2005.

Traceability for Agent-Oriented Design Models and Code

Gilberto Cysneiros
Dep. of Computer Science
City University, London
London, UK EC1V OHB
g.cysneiros@soi.city.ac.uk

Andrea Zisman
Dept. of Computer Science
City University, London
London, UK EC1V OHB
a.zisman@soi.city.ac.uk

ABSTRACT

In this paper we present a rule-based approach to support automatic generation of traceablity relations of design models and code specifications of agent-oriented systems. We define six different types of traceability relations between artefacts in Prometheus design models and JACK code. We assume the models and code specifications represented in XML and the rules specified in XQuery. A prototype tool has been developed in order to demonstrate and evaluate the work.

1. INTRODUCTION

Software traceability is considered an important activity when developing software systems [9][22][24][28] and it has been the focus of research for many years. Several approaches and techniques have been proposed to support software traceability. A survey of these approaches can be found in [28]. Traceability can be used to assist with evolution of software systems, reuse of parts of the system, validation that a system meets its requirements, understanding of the rationale for certain design decisions, identification of common aspects of the system, and change impact analysis of the system.

Despite its importance, existing support for traceability is not always adequate [22]. Some approaches assume that traceability relations between software artefacts should be established manually [7][13][24][25]. However, manual establishment of traceability relations is error-prone, difficult, time consuming, expensive, complex, and limited on expressiveness, causing traceability to be rarely established. Other approaches have been proposed to support semi- or fully-automatic generation of traceability relations [1][8][10][12][17][21][26][29], alleviating some of the above problems.

In recent years, we have been experiencing the development of Agent-Oriented Systems (AOS), in which software systems are composed of autonomous and flexible computational entities. A large number of systems have been developed by using agent-oriented paradigm. This paradigm has demonstrated success in many application areas, such as telecommunications, manufacturing, finance, air traffic management, aerospace, e-commerce, customer management, military simulation, decision support, and games. Different architectures and methodologies for AOS have been proposed [15].

Given the advances in the area of AOS and the importance of such systems, in this paper we propose a traceability approach to support the development of AOS. More specifically, we describe a rule-based approach to support automatic generation of traceability relations between agent-oriented design models and agent-oriented code. Our work concentrates on models generated when using Prometheus methodology [18] and code specified in JACK [30]. We have chosen Prometheus methodology as a basis of our work due to its large use in both academia and industrial settings and its support for the majority phases in the software engineering development life-cycle. The rationale for using code specified in JACK is also given by its large use in industrial settings. Moreover, JACK includes all components of Java programming language and offers extensions for implementing agent behaviour aspects. It is based on Belief, Desires and Intentions (BDI) architecture [23], which is considered one of the most successful architecture for developing AOS.

In our work, we assume the models in Prometheus and the code in JACK are represented in XML in order to tackle their heterogeneity. In addition, the rules to generate the traceability relations are represented in XQuery [31] to facilitate element identification in the XML documents.

The work described in this paper is part of a large project of research to support automatic generation of traceability relations based on rules between heterogeneous models generated during the development of software systems. The work here is built upon previous work to generate traceability relations between requirements artefact and analysis object models generated during the development of object-oriented systems [29], between documents created during the development of product line systems [12], and between i* models and Prometheus artefacts [3]. In this paper, we extend the work in [3], and focus on automatic generation of traceability relations for Prometheus design models and JACK code. We analyse Prometheus models and JACK code in order to specify traceability relations between their main elements and implement new traceability rules to generate these relations.

The remaining of this paper is structured as follows. In Section 2 we present an example of a Bookstore AOS with some Prometheus models and part of JACK code. In Section 3 we give an overview of the approach, describe the different types of traceability relations and rules, and illustrate the approach through some examples. In Section 4 we specified existing work in the area. Finally, in Section 5 we summarise our approach and suggest some future work.

2. EXAMPLE

This section describes an example of a Bookstore agent-oriented system. This system supports the main functionalities of selling books, validating the clients of the system, and managing catalogues of clients and books. The design models of the system are specified in Prometheus [18] while the code is implemented in JACK [30]. Due to space restriction, in this section we do not represent all the design models and the code of the system, but part of the models that will be used to illustrate our approach.

Prometheus methodology consists of several descriptors and diagrams to represent the design of AOS. The descriptors represent different types of artefacts such as agents, percepts, actions, goals, data, roles, plans, messages, and capabilities.

In Prometheus, (a) an agent represents an autonomous entity in an environment; (b) a percept represents data received by the environment; (c) an action represents how an agent affects the environment; (d) a goal describes the aim of an agent to be accomplished, some activities that an agent wants to achieve, or a state that should be either avoided or maintained once the goal is achieved; (e) a data represents external information that an agent needs to access or believes representing an agent's knowledge about the environment or itself; (f) a role specifies some functionality and groups together a set of related goals, percepts, actions, and data; (g) a plan is a sequence of actions that an agent can perform to achieve a goal; (h) a message represents communication between agents; and (i) a capability represents functionality of an agent by encapsulating percepts, data, actions, messages, plans, and internal capabilities.

Examples of diagrams in Prometheus are goal diagram, role diagram, use case scenario, system overview diagram, agent overview diagram, capability diagram, process diagram, and protocol diagram. Figures 1 and 2 present examples of a system overview diagram and agent overview diagram for the Bookstore AOS, respectively.

The system overview diagram specifies how an AOS interacts with the environment. The main elements in this diagram are agents (represented as rectangles with an image of an actor inside the diagram), percepts (represented as star elements) to which the agents respond, actions performed by the agents (represented as arrow rectangles), messages exchanged between agents (represented as a rectangle envelope), and external data accessed by the agents (not shown in Figure 1).

As shown in Figure 1, the Bookstore AOS is composed of agents Sales Assistant, Stock Manager, Credit Card Agent, and Security Manager. The Sales Assistant agent in Figure 1 responds to Book Details percept, which contains information about purchase of books, and to Keyword Search percept, which contains information about a search in the book catalogue. The Sales Assistant agent also sends Book Query and Book Purchase messages to Stock Manager agent requesting information about an item in the book catalogue and requesting a purchased item to be removed from the stock, respectively. The other elements in the diagram are represented similarly.

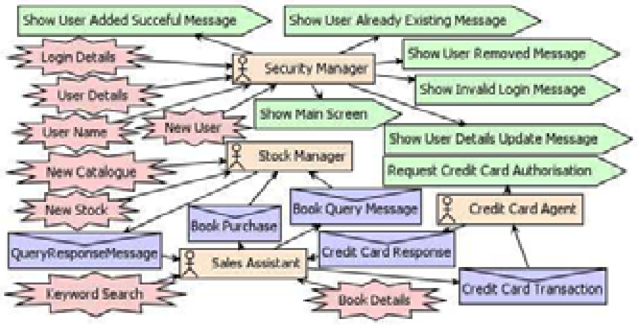


Figure 1: System Overview Diagram

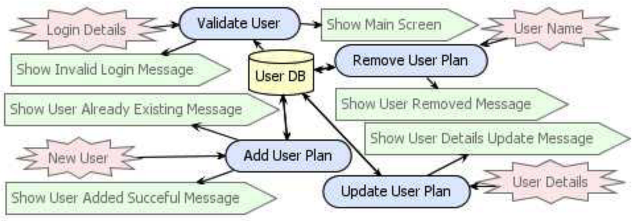

Login Details
Validate User
Show Main Screen
User Name
Show Invalid Login Message
User DB
Remove User Plan
Show User Already Existing Message
Show User Removed Message
Show User Details Update Message
New User
Add User Plan
Update User Plan
User Details
Show User Added Succeful Message

Figure 2: Security Manager Agent Overview Diagram

The agent overview diagram represents in details the design of an agent. The main elements in this diagram are actions (represented as arrow rectangles), plans (represented as ovals), data (represented as a data storage symbol), and percepts (represented as arrow rectangles). Figure 2 shows an example of an agent overview diagram for Security Manager agent in Figure 1. As shown in the figure, in the presence of Login Details percept, the Validate User plan is executed. This plan consists of checking if a user's login information matches a data in the User DB data storage. In positive case, action Show Main Screen is executed; otherwise action Show Invalid Login Message is performed. The other elements in the diagram are represented similarly. A detailed description of Prometheus is beyond the scope of this paper, but can be found in [18].

JACK language is based on Java programming language. It extends Java with agent-oriented constructs represented as classes, interfaces, and methods, and uses other valid Java declarations and statements such as package, import, attributes, and method definitions. An application in JACK is composed of several source code specifications representing agents, plans, events, capabilities, and belief sets.

In JACK, an agent specification is used to define the behaviour of a software agent. It includes the capabilities of an agent, the types of messages and events to which the agent responds, the events created by the agent, the belief sets (data) used by the agent to store information, and the plans the agent uses to achieve goals. Figure 3 shows an example of the implementation of agent SecurityManager in JACK. The SecurityManager agent inherits its core functionality from class Agent (*extends*). It contains declarations for events *(#posts* and *#handles*), plans, and data. A *#posts event* represents events that the agent can create, while a *#handles* event represents events to which the agent responds. A *#uses plan* specifies a plan executed by the agent, while a *#private data* identifies a belief set that the agent can use to store information. In the figure, the SecurityManager agent specification also contains an attribute bookstore (*private BookStore bookstore*) that provides an interface between the agent and the environment, and definitions of several methods to process percepts from the environment and create events.

```
public agent SecurityManager extends Agent {
   #posts event Login ev;
   #posts event CreateUser ev1;
   #posts event RemoveUser ev2;
   #posts event ChangePassword ev3;
   #handles event Login;
   #handles event CreateUser;
   #handles event RemoveUser;
   #handles event ChangePassword;
   #uses plan VerifyUser;
   #uses plan AddUser;
   #uses plan DeleteUser
   #uses plan ModifyPassword;
   #private data UserDB users();
   private BookStore bookStore;
   public SecurityManager(String name, BookStore bookStore){
      super(name); this.bookStore = bookStore;}
   public void login(String userName, String password) {
      postEventAndWait(ev.login(userName,password));}
   public void showMainMenu( ) {
      bookStore.showMainMenu();}
   public void modifyPassword(String password){
      postEventAndWait(ev3.changePassword(password));}
   public void showInvalidLogin(String message) {
      bookStore.showInvalidLogin(message);}}
```

Figure 3: Example of SecurityManager Agent in JACK

A plan specification describes a sequence of actions that an agent can execute when an event occurs. Figure 4 presents an example of the implementation of plan VerifyUser in JACK. This plan inherits its core functionality from class Plan. It contains declarations for events, interfaces, and data, and a *body()* method. The *body*() method describes the behavior of the agent (specified in the #uses interface declaration) when executing a plan. In this case, the *body()* method checks if the password and username properties of the Login event can be matched to a user recorded in the UserDB belief set.

In JACK, belief sets represent information that an agent has about its environment; events can be of type BDIGoalEvent, representing a goal that an agent wants to achieve, and BDIMessageEvent, representing events that one agent uses to communicate with another agent; and capabilities encapsulate plans, events, java code, and other functionalities. A detailed description of JACK is beyond the scope of this paper, but can be found at [30].

```
public plan VerifyUser extends Plan {
   #handles event Login ev;
   #uses interface SecurityManager self;
   #uses data UserDB users;
   …
   body() {
      logical String password;
      logical String name;
      logical String role;
      if (users.get(ev.userName,password,name, role)) {
         if (ev.password.equals(password.as_string())) {
            User user = new User(ev.userName,
                           name.as_string(), role.as_string());
            self.showMainScreen(user);
         } else { self.showInvalidLogin("Password Invalid");}
      } else {  self.showInvalidLogin("UserName Invalid");}}}
```

Figure 4: Example of VerifyUser Plan in JACK

3. OVERVIEW OF OUR APPROACH

We propose a rule-based approach to support automatic generation of traceability relations between Prometheus design models and JACK code. The use of rules are important to (i) automate and assist with decision making, (ii) allow standard ways of representing knowledge that can be used to infer data, (iii) facilitate the construction of traceability generators, and (iv) support representation of dependencies between elements in the documents.

In order to support the heterogeneity of models and tools used during the software development life cycle we assume the models to be represented in XML. We have chosen XML as the basis of our approach due to several reasons: (a) XML has become the de facto language to support data interchange among heterogeneous tools and applications, (b) the existence of large number of applications that use XML to represent information internally or as a standard export format, and (c) to allow the use of XQuery [31] as a standard way of expressing traceability rules. Moreover, our approach combines models in Prometheus and JACK code and, therefore, it requires a common representation of these models.

We propose to use an extended version of XQuery [31] to represent the rules. XQuery is an XML-based query language that has been widely used for manipulating, retrieving, and interpreting information from XML documents. Apart from the embedded functions offered by XQuery, it is possible to add new functions and commands. We have extended XQuery (a) to support representation of the consequence part of the rules, i.e. the actions to be taken when the conditions are satisfied, and (b) to support extra functions to cover some of the traceability relations being proposed. Examples of these functions are *isSynonym*, which verifies if the names of two elements are synonyms, and *isOverlap*, which verifies if an overlaps relation has been created between two elements.

In our approach, the models of our concern are generated using proprietary tools and represented in their native format (e.g. PDT[18], JACK[30]). These models are translated into XML format by using a *Model Translator* component based on XML Schemas proposed for the models, whenever the tools used to create the models do not generate them directly in XML. The XML-based models and rules are used as inputs to the *Traceability_Generator* component to generate traceability relations between the models. The component uses WordNet to support the identification of synonyms between the names of elements in the models. The traceability relations are represented in an XML document (*Traceability_Relations* document). The use of a separated document to represent the traceability relations is important to preserve the original models, to allow the use of these models by other applications and tools, and to allow the generated relations to be used to support the identification of other traceability relations that depend on the existence of previously identified relations (e.g. contributes, uses, creates, achieves, and depends on relations).

We call the traceability relations that do not depend on the existence of other relations as *primitive relations* and the ones that depend on the existence of other relations as *secondary relations*. The *Traceability_Relations* document is used as input to the *Traceability_Generator* component to support generation of secondary traceability relations. Examples of primitive and secondary relations are described in Subsection 3.1.

3.1 Traceability Relations

Based on the study of Prometheus methodology and analysis of the JACK language, our study and experience with software traceability [3][28][29][12], and types of traceability relations proposed in the literature [21][22], we have identified six different types of traceability relations between the various elements in the models used in our approach. These types of traceability relations have also been identified in our study of i* and Prometheus models [3]. Table 1 presents the different types of relations for the main types of elements in Prometheus and elements in JACK language. In Table 1, apart from overlaps relations that are bi-directional, the direction of a relation is represented from a row *[i]* to a column *[j]* (e.g. "Prometheus role is used by JACK agent"). In the following we describe the various types of traceability relations identified in our work.

Overlaps – In this type of relation, an element e1 *overlaps* with an element e2 (an element e2 *overlaps* with an element e1), if e1 and e2 refer to common aspects of AOS or its domain. For instance, an overlaps relation holds between Security Manager agent in Prometheus (Figure 1) and SecurityManager agent in JACK (Figure 3), since they refer to common aspects of the system. This is an example of a *primitive* relation.

Contributes (Contributed by) - In this type of relation, an element e1 *contribute*s *to* an element e2, if e1 assists with the achievement or accomplishment of another element e2. For instance, a contributes relation exists between Security Manger agent in Prometheus (Figure 1) and login method in JACK (Figure 3). This *secondary* relation is given by the fact that login method in JACK creates an event Login, which holds an overlaps relation with goal Login in Prometheus that is achieved by Security Manager agent (goal diagram is not shown in Section 2).

Uses (**Used by**) - In this type of relation, an element e1 *uses* an element e2, if e1 requires the existence of e2 in order to achieve its objective. For instance, a uses relation holds between Validate User plan in Prometheus (Figure 2) and Security Manager agent in JACK (Figure 3). In this case, SecurityManager JACK agent uses VerifyUser JACK plan that overlaps with Validate User plan in Prometheus (Figure 2). Therefore, we can infer that Security Manager agent in JACK uses Validate User plan in Prometheus. This is an example of a *secondary* relation due to the existence of the overlaps relation.

Creates (Created by) - In this type of relation an element e1 *creates* an element e2, if e1 generates element e2. For instance a *secondary* create relation exists between SecurityManager agent in JACK (Figure 3) and Show Invalid Login Message action in Prometheus (Figure 2), since SecurityManager agent contains showInvalidLogin method which holds an overlaps relation with Show Invalid Login Message action in Prometheus (Figure 2).

Achieves (Achieves by) - In this type of relation an element e1 achieves an element e2, if e1 meets the expectations and needs of e2. Plans in JACK describe a sequence of actions that an agent can take when an event occurs. In JACK, goals are implicitly represented as events that can trigger plans. Therefore, if there is a goal in Prometheus that has an overlaps relation with an event in JACK and a plan in JACK responds to that event, then we can say that there is an achieves relation between the plan in JACK and the goal in Prometheus. For instance, a *secondary* achieves relation exists between VerifyUser plan in JACK (Figure. 4) and Login goal in Prometheus (not shown in Section 2), since the VerifyUser plan responds to event Login that holds an overlaps relation with goal Login.

Depends on (**Is Dependent**) - In this type of relation an element e1 *depends on* an element e2, if the existence of e1 relies on the existence of e2, or if changes in e2 have to be reflected in e1. For instance, a depends on relation holds between Validate User plan in Prometheus (Figure 2) and showInvalidLogin method in JACK (Figure 3). This *secondary* relation is given by the fact that showInvalidLogin method holds an overlaps relation with Show Invalid Login Message action in Prometheus, and since a plan is defined as a sequence of actions, it depends on the existence of the methods that overlaps with these actions.

JACK Prometheus	Method	Agent		Plan		BeliefSet		Capability	BDIGoalEvent	BDIMessgeE vent	
Goal	Contributed by	Achieved by		Achieved by		Uses	Creates	Achieved by	Overlaps	---	
Role	Contributed by	Used by		Uses		Uses		Contributed by	Achieves	---	
Agent	Contributed by	Overlaps		Uses		Uses		Uses	Achieves	Uses	Creates
Capability	Contributed by	Used by		Uses		Uses		Overlaps	Achieves	---	
Plan	Depends on	Used by		Overlaps		Uses		Used by	Achieves	Uses	Creates
Percept	Is Dependent	Used by		Used by		---		Used by	Is Dependent	---	
Action	Overlaps	Created by		Created by		---		Used by	---	---	
Message	---	Used by	Created by	Used by	Created by	Uses		---	---	Overlaps	
Data	---	Used by	Created by	Used by	Created by	Overlaps		Used by	---	Used by	

Table 1: Different Types of Traceability Relations

3.2 Traceability Rules

The traceability rules used in our work to support automatic generation of traceability relations are composed of three main parts. Figure 5 shows a general template representing a pseudo-code for the rules. In the template, elements between square brackets ("[", "]") are optional, and $f_i(f_{i+1}\ldots(f_{i+j}(\bullet))\ldots)$ represents a composition of functions and if statements used in our rules. These functions are XQuery functions or the extra functions that we have implemented to support our work. Figure 6 presents an example of a rule represented in XQuery for generating *dependency* relation between methods in JACK and plans in Prometheus. We explain below the parts in a rule.

Part 1: It consists of the rule identification and contains a unique identifier (*RuleID*), a priority of the rule (*RulePriority*) indicating if this is a primitive rule (priority 1) or dependent rule (priority 2), the type of the rule (*RuleType*), the type of the source element to be traced (*ElemTypeA*), the type of the target element to be traced (*ElemTypeB*), and a brief description of the rule (*Description*). The priority of the rule is used to identify if the rule is primitive or dependent and to assist with the execution of the rules; i.e., rules with priority 1 are executed before rules with priority 2, since these latter rules depend on the existence of the relations generated by rules with priority 1. The type of the rule is based on the type of the traceability relation generated by the rule. Figure 6 shows examples of this first part of the rule.

Part 2: It consists of XQuery statements and is formed by other subparts. The first subpart (DECLARE) contains declarations of namespaces, documents, and sequence of elements used by the rule. The declaration of the documents and sequence of elements are described as XPath expressions. For the example in Figure 6, there are declarations of (a) *Similar* java classes, (b) JACK and Prometheus models (JACK.xml and BookShop.pd), and (c) sequences of elements of JACK methods and Prometheus plans to be compared ($methods and $plans).

The second subpart (*FOR*) iterates elements of the sequences and bind these elements to variables ($\$elem_a$ $\$elem_b$ $\$elem_c$). As shown in Figure 6, the first *for* statement binds plans in Prometheus and methods in JACK to variables $plan and $method, respectively.

The third subpart (CONDITION) defines the *condition* part of the rule that should be satisfied. Conditions can be defined by the *where-expression* clause of a *for* clause in XQuery or by the *test-expression* part of an *if-then-else* expression in XQuery. The condition part of the rule uses XQuery built-in functions and expressions, and the Java extra functions that we have developed.

In the example in Figure 6, a where expression in the condition part of the rule checks if a plan in Prometheus contains at least one action that has an overlaps relation with a method in JACK. For instance, showInvalidLogin JACK method in Figure 3 has an overlaps relation with ShowInvalidLoginMessage Prometheus action in Figure 2. In addition, Validate User Prometheus plan uses ShowInvalidLoginMessage Prometheus action in Figure 2. Thus, the condition in the where part of the rule holds.

The fourth subpart (ACTION) specifies the *consequence* part of the rule when the conditions are satisfied. It describes traceability relations (RELATION). The evaluation of the consequence part consists of writing the traceability relations in the XML *Traceability_Relation* document. Figure 7 presents part of this document with the result of executing RulePJ5b in Figure 6 for Validate User Prometheus plan (Figure 2) and showInvalidLogin JACK method (Figure 3). A traceability relation contains information about the respective rule (*RuleID*), its type (*RelType),* and related elements (element <Element>).

```
TRACE_RULE
  RuleID = R_ID
  RulePriority = Priority_Number
  RuleType = Rule_Type
  ElemTypeA = ElementTypeName
  ElemTypeB = ElementTypeName
  Description = DescriptionText
  XQUERY
    [DECLARE Namespace]
    [DECLARE Documents]
    [DECLARE Sequences]
    for $elem_a in $seq_a,
        $elem_b in $seq_b,…
        $elem_n in $seq_n
        CONDITION fi(fi+1…(fi+j(•))…)
        ACTION
         RELATION
            RuleID = R_ID
            RelType = Relation_Type
            [ELEMENT
              Document = DocumentPath
                ElemType = ElementType
              ElemName = ElementName
              ElemID = ElementID]
           ACTION_END
   XQUERY_END
TRACE_RULE
```

Figure 5: Traceability Rule Template

```
<Rule id="rulePJ5b" priority="2"   type="dependency"
        elementTypeA="Method" elementTypeB="Plan"
        description="Dependency between a method in
                JACK and a plan in Prometheus">
  <XQuery> <![CDATA[
         declare namespace sim = "java:xquery.Similar";
         let $prometheus := doc("file:///c:/retratos/BookShop.pd")
         let $jack := doc("file:///c:/retratos/JACK.xml")
         let $plans := $prometheus//object[@type='Plan']
         let $methods := $jack//method
    for $plan in $plans, $method in $methods
      where (some $actionID in
            ($plan//field[@name='actions']/list/object/@ref  |
             $plan//field[@name='actions']/list/object/@id)
            satisfies  sim:isOverlap($actionID,$method/@id))
      return
        <TraceabilityRelation type="dependency"  ruleID="rulePJ5b" >
          <Element doc="c:/retratos/BookShop.pd" type="Plan"
            name="{$plan/base/field[@name='name']/text()}"
            id="{$plan/@id}"/>
          <Element doc="c:/retratos//JACK.xml" type="Method"
             name="{$method/@name}" id="{$method/@id}"/>
      </TraceabilityRelation> ]]>  </XQuery>  </Rule>
```

Figure 6: Example of a Dependency Traceability Rule

```
<Traceability>…
<TraceabilityRelation ruleID="PJ5b" type="dependency">
   <Element  doc=" c:/retratos/BookShop.pd "  type="Plan"
       name="Validate User"  id="59 "/>
   <Element doc="c:/retratos/JACK.xml" type="Method"
       name="showInvalidLogin" id="m6"/>
</TraceabilityRelation> …
```

Figure 7: Example of a Traceability Relation

4. RELATED WORK

In [16], the authors state that the lack of tools to support AOS development is one of the challenges that needs to be overcome before agent-oriented paradigm can be widely adopted by industry. More recently, in [11] and [19] the authors proposed tools to support automatic mapping between agent-oriented models applying a Model Driven Architecture approach. In [17] the authors suggest debugging approaches to AOS. In [5], an approach to support automatic change propagation in agent software evolution is presented. Although some work has been proposed to support AOS development, an approach to assist with automatic traceability generation of AOS design models and code specification has not been proposed.

Approaches for traceability generation can be classified as manual, semi-automatic, and automatic [28]. Most of the commercial tools adopt the manual approach in which the user has to select the source and target objects to be traced and offer sophisticated visualisation, display, and navigability components [7][24]. The task of creating traceability relations manually is costly, labour-intensive, and error-prone. As a consequence, the cost of establishing traceability relations can overcome their benefits. Semi-automatic approaches are concerned with the generation of traceability relations based on a set of previously established relations [8], or when traceability relations are generated as a by-product of the software development process [21]. Several techniques have been proposed to support automatic generation of traceability relations ranging from information retrieval [1][10][17], rule-based [12][29], inference axioms [20], and hypermedia and information integration [26]. However, these approaches have not been used in the scope of AOS models and code. The work presented in this paper extends the work in [12][29] to support AOS models.

Examples of approaches to support traceability involving software code have been proposed in [1][8][17][26]. In [1] the authors apply probabilistic and vector space information retrieval techniques to find traceability relations between C++ code and manual documents, and Java code and functional requirements. They assume that programmers use meaningful names for items such as functions, variables, types, and methods. The work in [17] uses another information retrieval technique, latent semantic analysis, to identify traceability relations between code and system documentation, expressed in natural language. The scenario-based approach in [8] uses a set of hypothesized trace relations between software artifacts and scenarios and generates other traceability relations between scenarios and related code. The work in [26] uses open hypermedia and information integration techniques to enable the discovery, creation, maintenance, and visualisation of traceability relations between requirements and code. However, none of these approaches are based on the use of rules and have been tested in the scope of AOS and JACK code.

Various classifications for different types of traceability relations are presented in reference models and framework [4][6][9][21][22][28]. However, despite the reference models and classifications there is still a lack of standard semantic definition for the various types of relations [28]. The need to capture the semantic of traceability relations is fundamental to provide their effective use. Many existing tools support the representation of different types of relations, but the interpretation of these relations depends on the stakeholders, which causes confusion when interpreting relations and difficulties to develop tools for automatic generation of traceability relations. In this paper, we based upon the work in [3][12][29] and contribute to fulfil the lack of a classification for traceability relations for models and code specifications generated during the development of AOS. The work in this paper complements the work in [3], providing a traceability approach for different phases of the development of AOS; i.e. requirements, design, and implementation phases.

5. CONCLUSION AND FUTURE WORK

In this paper we present a rule-based approach to support automatic generation of traceability relations between Prometheus design models and JACK code. We presented six different types of traceability relations between different artefacts in Prometheus design models and JACK code. In the work, we assume design models and code specified in XML and define rules in an extension of XQuery. We have implemented 52 traceability rules. A prototype tool to parse the rules and create traceability relations has been implemented in Java and uses Saxon [27] to evaluate XQuery parts of the rules.

Currently, we are evaluating the work through some real case studies in terms of recall and precision in order to verify the effectiveness of the approach and to verify the completeness of the set of traceability rules created. Initial experiments of our work for automatic generation of traceability relations between i* and Prometheus models were promising. We are also extending the rules in order to support automatic identification of missing elements between Prometheus design models and JACK code.

6. REFERENCES

[1] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia and E. Merlo, "Recovering Traceability Links between Code and Documentation," IEEE Transaction on Software Engineering, v. 28, 2002.

[2] J. Cleland-Huang, C. Chang, G. Sethi, K. Javvaji , H. Hu, J. Xia, "Automating Speculative Queries through Event-based Requirements Traceability", IEEE Joint Int. Req. Eng. Conference, Essen, Germany, 2002.

[3] G. Cysneiros and A. Zisman. “Tracing Agent-Oriented Systems”. In Proc. of the Grand Challenge Traceability Symposium, USA, March 2007.

[4] A. Davis. "The analysis and specification of systems and software requirements," Systems and Software Requirements Engineering, 1990.

[5] K. Dam, M. Winikoff, and L. Padgham. "An agent-oriented approach to change propagation in software evolution", ASEC, 2006.

[6] J. Dick. "Rich Traceability," TEFSE, UK, 2002.

[7] DOORS., www.telelogic.com/products/doors.

[8] A. Egyed, "A Scenario-Driven Approach to Trace Dependency Analysis," IEEE Trans. on Software Engineering, vol. 29, 2003.

[9] O. Gotel, and A. Finkelstein "An Analysis of the Requirements Traceability Problem", International Conference on Requirements Engineering, USA, 1994.

[10] J.H. Hayes, A. Dekhtyar , S.K. Sundaram, “Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods”, IEEE Transaction on Software Engineering, V. 32, No. 1, 2006.

[11] G. Jayatilleke, L. Padgham, and M. Winikoff, "A model driven development toolkit for domain experts to modify agent based systems," AOSE, 2006.

[12] W. Jirapanthong and A. Zisman. “Supporting Product Line Develeopment through Traceability”, APSEC, Taiwan, 2005.

[13] H. Kaindl. "The Missing Link in Requirements Engineering", Software Engineering Notes, June 1992.

[14] L. Lavazza and G. Valetto, "Requirements-based Estimation of Change Costs", Empirical Software Engineering - An International Journal, 5(3), November 2000.

[15] M.Luck, “From Definition to Deployment: What Next for Agent-based Systems?”, The Knowledge Engineering Review, Vol. 14:2, pp.119-124, 1999.

[16] M.Luck, R. Ashri, and M. D’Inverno, “Agent-based Software Development”, ISBN 1-58053-605-0, 2004.

[17] A. Marcus and J.I Maletic, "Recovering Documentation-to-Source-Code Traceability Links using Latent Semantic Indexing", ICSE, 2003.

[18] L. Padgham and W.Winikoff. "Developing Intelligent Agent Systems–A Practical Guide", John Wiley & Sons, 2004.

[19] A. Perini and A. Susi, "Automating Model Transformations in Agent-Oriented Modelling," AOSE, 2005.

[20] F. Pinheiro and J.Goguen, "An object-oriented tool for tracing requirements," IEEE Software, vol. 13, 1996.

[21] K. Pohl, "Process-Centered Requirements Engineering," John Wiley & Sons, 1996.

[22] B. Ramesh and M. Jarke, "Towards Reference Models for Requirements Traceability", IEEE Transactions on Software Engineering, vol. 37, 2001.

[23] A. Rao and M. Georgeff. “BDI Agents: from theory to practice”, International Conference on Multi-Agent Systems, The MIT Press. Cambridge, MA, USA. 1995.

[24] RDT, http://www.igatech.com/rdt/index.html

[25] RTM. Integrated Chipware. www.chipware.com.

[26] S. Sherba, "Towards Automating Traceability: An Incremental and Scalable Approach," PhD thesis, Dep. of Computer Science, University of Colorado, 2005.

[27] Sourceforge; Saxon: http://saxon.sourceforge.net/

[28] G. Spanoudakis and A. Zisman, "Software Traceability: A Roadmap," in S. K. Chang, ed., Handbook of Software Engineering and Knowledge Engineering, August, 2005.

[29] G. Spanoudakis, A. Zisman, E. Pérez-Miñana, and P. Krause, "Rule-based Generation of Requirements Traceability Relations", Journal of Systems and Software, v. 72, 2004

[30] M. Winikoff, JACK^{TM} Intelligent Agents: An Industrial Strength Platform, Multi-Agent Programming, 2005.

[31] XQuery. http://www.w3.org/TR/xquery/.

ONTOMADEM: An Ontology-driven Tool for Multi-Agent Domain Engineering

Rosario Girardi and Adriana Leite
Federal University of Maranhão
Av. dos Portugueses, s/n, Campus do Bacanga,
CEP 65080-040, São Luís–MA, Brazil
{rgirardi@deinf.ufma.br, adri07lc@gmail.com}

Abstract

A significant contribution of Knowledge Engineering to Software Engineering comes from techniques and formalisms for knowledge representation and ontology development. Ontologies provide an unambiguous terminology that can be shared by all involved in a software development process. They can also be as generic as needed allowing its reuse and easy extension. These features turn ontologies useful for representing the knowledge of software engineering techniques and methodologies, and an appropriate abstraction mechanism for the specification of high-level reusable software artifacts like domain models, frameworks and software patterns. This work describes ONTOMADEM, a conceptualization of MADEM ("Multi-Agent Domain Engineering Methodology"), a software development methodology for Multi-Agent Domain Engineering. The ontology is used as a knowledge-based tool for capturing and representing the products of a Multi-agent Domain Engineering process, created through the instantiation of its hierarchy of classes.

1 Introduction

A significant contribution of the Knowledge Engineering discipline to the Software Engineering one comes from techniques and formalisms for knowledge representation and ontology development.

An ontology is usually defined as the specification of a conceptualization [20], a simplified, abstract way of perceiving a segment of the real world as a set of objects and their relationships, as well as the terms used to refer to them and their agreed meanings and properties. This specification is formal, i.e. can be processed by a computer system; it is explicit, i.e. concepts and constraints are previously and explicitly defined; it is sharable, i.e. it relates to consensual knowledge accepted by a group and is used by more than one individual. Ontologies are frequently formalized with knowledge representation structures, e.g. frame-based systems where concepts are represented in frames and relationships between concepts as frame slots. Thus, they can be available in knowledge bases where concepts are semantically related allowing effective searches and inferences thus facilitating their understanding and reuse.

Ontologies provide an unambiguous terminology that can be shared by all involved in a software development process. They can also be as generic as needed allowing its reuse and easy extension. These features turn ontologies useful for representing the knowledge of software engineering techniques and methodologies, and an appropriate abstraction mechanism for the specification of high-level reusable software artifacts like domain models, frameworks and software patterns [13][14][19].

This work describes ONTOMADEM, a conceptualization of MADEM ("Multi-Agent Domain Engineering Methodology"), a software development methodology for Multi-Agent Domain Engineering [14]. The MADEM methodology integrates techniques for domain analysis, domain design and domain implementation: GRAMO ("Generic Requirement Analysis Method based on Ontologies"), DDEMAS ("Domain DEsign technique of Multi-Agent Systems") and DIMAS ("Domain Implementation technique of Multi-Agent Systems"), respectively. Earlier work on GRAMO and DDEMAS has been already published [12][17].

The ontology is used as a knowledge-based tool for capturing and representing the products of a Multi-agent Domain Engineering process, created through the instantiation of its hierarchy of classes. It integrates ONTOPATTERN [13], an ontology we have previously developed for the representation and reuse of software patterns [16].

The paper is organized as follows. Section 2 describes the conceptualization of the MADEM methodology in the ONTOMADEM ontology. Section 3 introduces how domain models and multi-agent frameworks are developed with ONTOMADEM. The semantic relationships between the modeling concepts in these products are illustrated with examples extracted from cases studies conducted to evaluate both MADEM and ONTOMADEM. Section 4 discusses the capabilities provided by ONTOMADEM for using and reusing modeling products. Section 5 concludes the paper and discusses further work being conducted.

2 The ONTOMADEM tool

ONTOMADEM was developed in a two phase development process: the specification and the design of the ontology. In the specification phase, a conceptualization of MADEM was represented in a semantic network. In the design phase, concepts and relationships in the semantic network were mapped to a frame-based ontology in Protégé [11].

Three main classes representing the MADEM concepts drive the design of ONTOMADEM: *Modeling concepts*, *Modeling tasks* and *Modeling Products*. These concepts are described in the following sections.

2.1 MADEM modeling concepts

Main modeling concepts and tasks of MADEM are based both on techniques for Domain Engineering [2][8][21] and for development of multi-agent systems [5][6][9][24].

2.1.1 Domain Engineering

Domain Engineering and Application Engineering are two complementary software processes. Domain Engineering, also known as Development FOR Reuse, is a process for creating software abstractions reusable on the development of a family of software applications in a domain, and Application Engineering or Development WITH Reuse, the one for constructing a specific application using reusable software abstractions available in the approached domain(s).

A family of systems is defined as a set of existing software systems sharing some commonalities but also particular features [8].

The process of Domain Engineering is composed of the phases of analysis, design and implementation of a domain. Domain analysis activities identify reuse opportunities and determine the common and variable requirements of a family of applications. The product of this phase is a domain model. Domain design activities look for a documented solution to the problem specified in a domain model. The product of this phase is composed of one or more frameworks and, possibly, a collection of design patterns, documenting good solutions in that domain. Reusable components integrating the framework are constructed during the phase of domain implementation. This is the compositional approach of Domain Engineering. In a generative approach, Domain Engineering produces Domain Specific Languages (DSLs), and application generators that are used to construct a family of applications in a domain automating the reuse activities of selection, adaptation and composition [8].

A main concern of MADEM and Domain Engineering techniques is variability modeling for capturing and representing mandatory, alternative or optional features of domain models. Mandatory features should be present in all systems of the family while alternative and optional ones form the variable part of a model providing alternative or particular realizations of the model through its reuse in the Application Engineering process.

In ONTOMADEM, common and variable features are represented in the variability slot of the modeling concepts sub-classes, by instantiating the *Variability* class associated to that slot, according to the type of variability (mandatory, alternative or optional).

2.1.2 Agent-oriented modeling

For the specification of the problem domain to be solved, MADEM focuses on modeling the context, concepts, goals, roles and interactions of entities of an organization.

Entities have knowledge and use it to exhibit autonomous behavior. An organization is composed of entities with general and specific goals that establish what the organization intends to reach. The achievement of specific goals allows reaching the general goal of the organization (*reached from* relationship of Figure 1). For instance, an information system can have the general goal "satisfying the information needs of an organization" and the specific goals of "satisfying dynamic or long term information needs". Specific goals are reached through the performance of responsibilities (*achieves* relationship of Figure 1) in charge of particular roles (*in charge of* relationship of Figure 1) with a certain degree of autonomy. In the example of Figure 1, the *Retriever* role is in charge of the *Matching and similarity analysis* responsibility.

Responsibilities are exercised through the execution of activities (*exercised through* relationship of Figure 1). The set of activities associated with a responsibility are a functional decomposition of it.

Roles have skills on one or a set of techniques that support the execution of responsibilities and activities in an effective way (*requires* relationship of Figure 1). Pre-conditions and post-conditions may need to be satisfied for/after the execution of an activity (*is satisfied* and *satisfies* relationships of Figure 1). Knowledge can be consumed and produced through the execution of an activity (*uses* and *produces* relationships of Figure 1). For instance, an entity can play the role of "retriever" with the responsibility of executing activities to satisfy the dynamic information needs of an organization. Another entity can play the role of "filter" in charge of the responsibility of executing activities to satisfy the long-term information needs of the organization. Skills can be, for instance, the rules of the organization that entities know to access and structure its information sources.

Sometimes, entities have to communicate with other internal or external entities to cooperate in the execution of an activity (*participates* relationship of Figure 1). For instance, the entity playing the role of "filter" may need to interact with a user (external entity) to observe his/her behavior in order to infer his/her profile of information interests.

For the specification of a design solution, roles are assigned to agents (*plays* relationship of Figure 1) structured and organized into a particular multi-agent architectural solution according to non-functional requirements.

.

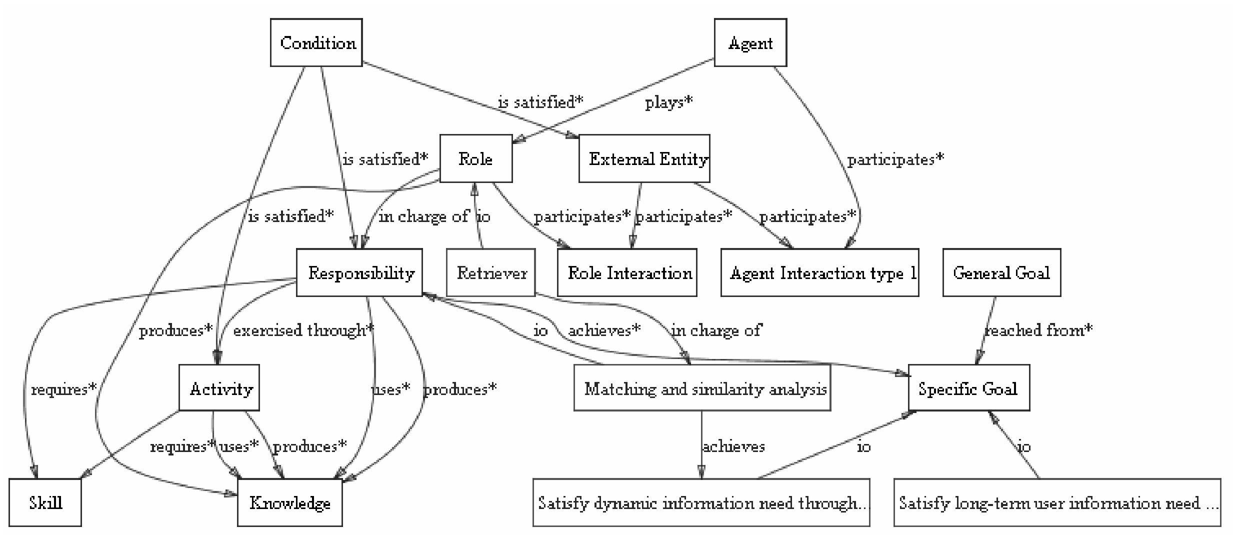


Figure 1 Some relationships between modeling concepts and their instances in the ONTOMADEM ontology

MADEM modeling tasks and products

The MADEM methodology supports the domain analysis, domain design and domain implementation phases of a multi-agent domain engineering process (Table 1).

Domain Analysis supported by the GRAMO technique approaches the construction of a domain model which specifies the current and future requirements of a family of applications in a domain by considering domain knowledge and development experiences extracted from domain specialists and applications already developed in the domain. Existing analysis patterns can also be reused in this modeling task.

Domain Analysis is performed through the following modeling tasks: *Context Modeling*, *Modeling of domain concepts*, *Goal Modeling*, *Role Modeling*, *Variability Modeling, Role Interaction Modeling* and *User Interface Prototyping*. The product of this phase, a *Domain Model*, is obtained through the composition of the products constructed through these tasks: a *Context Model*, a *Concept Model,* a *Goal Model, a Role Model,* a set of *Role Interaction Models* and a prototype of the user interface.

The *Context Modeling* task looks for representing in a *Context Model* a general view of the system environment and the external entities with which it interacts.

The *Modeling of domain concepts* task aims at performing a brainstorming of concepts of the domain and their relationships, representing them in a *Concept Model.* These concepts are refined in the subsequent modeling tasks.

The purpose of the *Goal Modeling* task is to identify the goals of the family of systems, the external entities with which it cooperates and the responsibilities needed to achieve them. Its product is a *Goal Model*, specifying the general and specific goals of the system family along with the external entities and responsibilities.

The *Role Modeling* task associates the responsibilities identified in the *Goal Modeling* task to the roles that will be in charge of them. The skills required for exercising a responsibility, the pre- and post-conditions that must be satisfied before and after the execution of a responsibility are also identified. Finally, the knowledge required from other entities (roles or external entities) for the execution of activity or responsibility and the knowledge produced from their execution is recognized. This task produces a *Role Model*, specifying roles, responsibilities; activities, skills, pre- and post-conditions, knowledge and relationships between these concepts.

Table 1- Modeling Phases, Tasks and Products of MADEM Methodology

Phases	Tasks		Products		
Domain Analysis	Concept Modeling		Concept Model		Domain Model
	Variability Modeling	Goal Modeling	Goal Model		
		Role Modeling	Role Model		
	Role Interactions Modeling		Role Interactions Models		
	User Interface Prototyping		User Interface Prototype		
Domain Design	Multi-agent Society Knowledge Modeling		Multi-agent Society Knowledge Model		Multi-agent Framework Model
	Architectural Design	Multi-agent Society Modeling	Multi-agent Society Model	Architectural Model	
		Agent Interaction Modeling	Agent Interaction Model		
		Activity Modeling	Activity Model		
		Coordination and Cooperation Modeling	Coordination and Cooperation Model		
	Agent Design	Agent Knowledge Modeling	Agent Knowledge Models	Agent Models	
		Agent State Modeling	Agent State Models		
Domain Implementation	Mapping from Design to Implementation Agents and Behaviors		Model of Agents and Behaviors		Implementation Model of the Multi-agent Society
	Mapping from Agent Interactions to Communication Acts		Model of Agent Communication Acts		
	Agent Implementation		Executable software agents		
Pattern Extraction and Representation			Software Patterns and Patterns Systems		

The *Variability Modeling* task is performed simultaneously with the *Goal and Role Modeling* ones.

The purpose of this task is to classify goals, roles, responsibilities and skills in *Goal* and *Role* models as common or variable features.

The *Role Interaction Modeling* task aims at identifying how external and internal entities should cooperate to achieve a specific goal. For that, responsibilities of roles are analyzed along with their required and produced knowledge specified in the *Role Model*. A set of *Role Interaction Models* specifying the interactions between roles and external entities needed to achieve a specific goal is constructed as a product of this task.

The *Goal* and *Role Models* provide a static view of the organization; the set of *Interactions Models*, a dynamic one.

The goal of the *User Interface Prototyping* task is to identify the interactions of the users with the system and simulate them in a prototype.

Domain design supported by the DDEMAS technique approaches the architectural and detailed design of multi-agent frameworks providing a solution to the requirements of a family of multi-agent software systems specified in a domain model. It consists of three sub-phases: the *Multi-agent society Knowledge Modeling* sub-phase, which identifies and semantically represents the concepts shared by all agents in their communication; the *Architectural Design* sub-phase, that establishes an architectural model of the multi-agent society including its coordination and cooperation mechanisms; and the *Agent Design* sub-phase, that defines the internal design of each agent, modeling its structure and behavior.

Domain implementation supported by the DIMAS technique approaches the mapping of design models to agents, behaviors and communication acts, concepts involved in the JADE framework [11], which is the adopted implementation platform. An *Implementation Model of the Multi-agent Society* is constructed as a product of this phase of MADEM, composed of a *Model of agents and behaviors* and a *Model of communication acts*.

The conceptualization of some MADEM modeling tasks and products in the ONTOMADEM ontology is illustrated in Figure 2 and Figure 3. Figure 2 shows the classes and relationships between modeling tasks of the domain design sub-phase in the ONTOMADEM ontology. Figure 3 illustrates the classes and relationships between modeling products of the domain model.

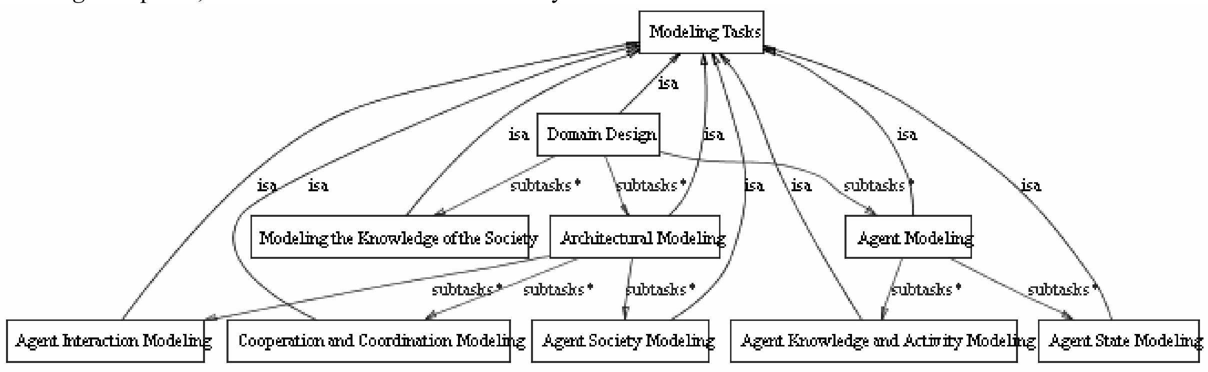

Modeling Tasks
isa
Domain Design
subtasks*
Modeling the Knowledge of the Society
Architectural Modeling
Agent Modeling
Agent Interaction Modeling
Cooperation and Coordination Modeling
Agent Society Modeling
Agent Knowledge and Activity Modeling
Agent State Modeling

Figure 2 Classes and relationships between modeling tasks of the domain design sub-phase in the ONTOMADEM ontology

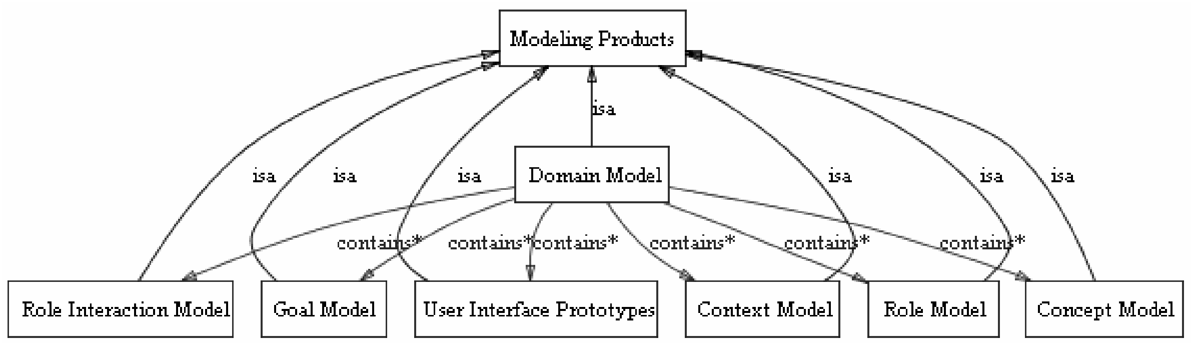


Figure 3 Classes and relationships between modeling products of the domain model in the ONTOMADEM ontology

3 Constructing domain models and multi-agent frameworks with ONTOMADEM

In ONTOMADEM, modeling products are generated through the instantiation of the corresponding *Modeling Tasks*, *Modeling Products*, and *Modeling Concepts* classes. For instance, the construction of a *Domain Model* requires the subsequent instantiation of the subclasses *Domain Analysis* of the *Modeling Tasks* class and *Domain Model* of the *Modeling Products* class;

then, the instantiation of each class representing a subtask of the *Domain Analysis* class (*Context Modeling , Concept Modeling*, *Goal Modeling*, *Role Modeling Role Interaction Modeling* and *User Interface Prototyping*); finally, the instantiation of each class representing a product composing a *Domain Model* (*Context Model, Concept Model, Goal Model, Role Mode, Role Interaction Models* and *Prototype of the User Interface*).

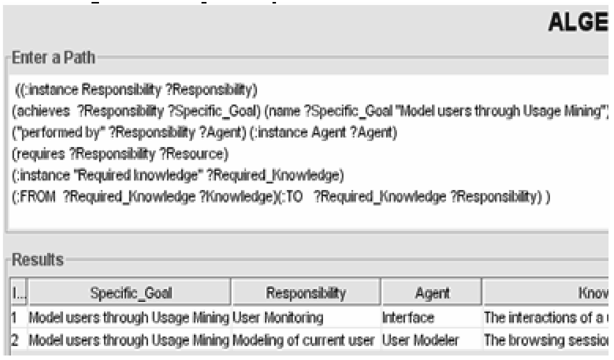
A graphical notation has been defined in MADEM for the representation of each modeling product [14]. This facilitates not only the instantiation process but also contributes for reducing the complexity of the modeling tasks allowing the visualization, decomposition and refinement of the modeling products.

Several domain and design models have been constructed in order to evaluate both MADEM and ONTOMADEM. Recently, we have developed two families of multi-agent systems: one that approaches the problem of providing personalized Web services through Usage Mining [14] and another one supporting the development of applications for Web Information Retrieval and Filtering.

ONTOWUM-DM, a domain model describing the common and variable requirements of a family of multi-agent applications for providing personalized Web services through Usage Mining is described in [14]. ONTOWUM-DD, a multi-agent design solution to the requirements specified in ONTOWUM-DM, is introduced in [23]. In [16], a system of software patterns extracted from this development experience is described.

4 Using and reusing modeling products from ONTOMADEM

Since retrieval is based on semantics, results from searching on modeling products and concepts in the ONTOMADEM knowledge base are more effective than the ones that could be obtained through simple keyword retrieval on instance texts.



```
((:instance Responsibility ?Responsibility)
(achieves  ?Responsibility ?Specific_Goal) (name ?Specific_Goal "Model users through Usage Mining");
("performed by" ?Responsibility ?Agent) (:instance Agent ?Agent)
(requires ?Responsibility ?Resource)
(:instance "Required knowledge" ?Required_Knowledge)
(:FROM  ?Required_Knowledge ?Knowledge)(:TO   ?Required_Knowledge ?Responsibility) )
```

I...	Specific_Goal	Responsibility	Agent	Know
1	Model users through Usage Mining	User Monitoring	Interface	The interactions of a
2	Model users through Usage Mining	Modeling of current user	User Modeler	The browsing sessio

Figure 4 Results on semantic searching for modeling concepts and products in the ONTOMADEM knowledge base

For instance, the following *information need* could be expressed in the query of Figure 4, using the Algernon query language supported by the Algernon plug-in of Protégé:

"Show the responsibilities that allow reaching the specific goal of "Model users through Usage Mining" along with the agents in charge of, required knowledge, and resources required to perform them".

High precision is exhibited in the results of the query. This facilitates the understanding, validation and reuse of modeling products from the ONTOMADEM knowledge based repository.

5 Related work

Some prototypes of knowledge-based tools and environments, like ODYSSEY [4] and ODE [10], have been already developed to increase the productivity of the software development process, the reusability of generated products, and the effectiveness of project management. One main characteristic distinguishing ONTOMADEM from these approaches is its reuse support for agent-oriented software development.

Most available development tools for the construction of multi-agent systems, like JADE [3] provide support only to the implementation, debugging and deployment phases of a software development cycle.

Some tools supporting the earlier phases, like PTK, begin to appear. PTK (Passi Toolkit) [7] is an add-in for the commercial UML-based CASE tool Rational Rose. It enables the user to follow the PASSI process of analysis and design, providing a set of functionalities that are specific for each phase of the process by means of sub and pop-up menus that appear after having selected some UML elements (classes, use cases and so on). This tool also allows the designers to perform checking operations, which are based on the correctness of single diagrams and consistency between related steps and models.

Two main differences between ONTOMADEM and PTK are the support that the first one provides for the construction of the reusable products of a Multi-agent Domain Engineering process and the ontological approach in which it is based.

6 Concluding remarks and further work

This work introduced ONTOMADEM, a knowledge-based tool for Domain Engineering of multi-agent systems. ONTOMADEM supports the application of the MADEM methodology for the construction of families of multi-agent applications in a problem domain. Domain models and multi-agent frameworks are represented as semantically related instances of the ONTOMADEM ontology, turning it a knowledge-based repository where precise searches and logical inferences can be done thus facilitating the validation, understanding and reuse of the available modeling products.

Two main case studies have been developed to evaluate both MADEM and ONTOMADEM in the problems domains of information retrieval and filtering, recommendation systems and user modeling based on usage mining [14][23].

From the point of view of Knowledge Engineering for Software Engineering [1], ONTOMADEM contributes with an example of conceptualization of a software development methodology.

The ONTOMADEM evolution is guided by the improvements we are still introducing in the MADEM methodology. Since we have adopted the JADE/JESS framework [3] for our implementation activities, we are taking advantage of the integration capabilities of the JADE and Protégé development environments to the partial generation of code.

We have also worked on a methodology for Multi-agent Application Engineering, a process for the construction of specific multi-agent applications by reusing the products of the Multi-agent Domain Engineering process. A conceptualization of this methodology is being used to extend ONTOMADEM to support a Multi-agent Application Engineering process.

A Protégé plug-in is being developed for a partial automation of the MADEM modeling tasks in ONTOMADEM based on a set of inference rules.

Using an approach similar to the one described in [18] we are also working on the mapping of queries in natural language to a logic-based formalism in order to turn more intuitive and user-friendly the searches for modeling concepts and products in ONTOMADEM.

References

1. A. Abran, J. Moore, P. Bourque, R.L. Dupuis, L. Tripp, Guide to the Software Engineering Body of Knowledge – SWEBOK, Trial Version 1.0, IEEE-Computer Society Press, May 2001, URL: http://www.swebok.org
2. Arango, G.: Domain Engineering for Software Reuse. Ph.D. Thesis. Department of Information and Computer Science, University of California, Irvine, 1988.
3. Bellifemine, F., Caire, G., Poggi, A., Rimassa, G.. JADE A White Paper. Exp v. 3 n. 3, Sept 2003. http://jade.tilab.com/
4. Braga, R.; Werner, C.; Mattoso, M. "Odyssey: A Reuse Environment based on Domain Models", IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET'99), pp.50-57, Texas, Mar 1999.
5. Bresciani, P., Giorgini, P., Giunchiglia, F., and Mylopoulos, J., and Perini, A.: TROPOS: An Agent-Oriented Software Development Methodology. In Journal of Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers Volume 8, Issue 3, May (2004) 203 – 236.
6. Cossentino, M., Sabatucci, L., Sorace, S. and Chella, A.: Patterns reuse in the PASSI methodology. In: Proceedings of the Fourth International Workshop Engineering Societies in the Agents World (ESAW'03), pp. 29-31. Imperial College London, UK. October 2003.
7. Cossentino, M. and Potts, C., 2002. A CASE tool supported methodology for the design of multi-agent systems. In: The 2002 International Conference on Software Engineering Research and Practice (SERP'02). June 24-27, Las Vegas (NV), USA.
8. Czarnecki, K., Eisenecker, U. W.: Generative Programming: Methods, Tools, and Applications. ACM Press/Addison-Wesley Publishing Co., New York, NY, 2000.
9. Dileo, J., Jacobs, T. and Deloach, S.: Integrating Ontologies into Multi-Agent Systems Engineering. Proceedings of 4th International Bi-Conference Workshop on Agent Oriented Information Systems (AOIS 2002), pp. 15-16, Bologna (Italy), July 2002.
10. Falbo, R. A., G. Guizzardi, and Duarte, K. C.: An Ontological Approach to Domain Engineering. In Proceedings of the XIV International Conference on Software Engineering and Knowledge Engineering (SEKE 2002), Ischia, Italy, ACM Press, pp. 351-358, 2002.
11. Gennari, J., Musen, M. A., Fergerson, R. W. et al.: The Evolution of Protégé: An Environment for Knowledge-Based Systems Development. Technical Report SMI-2002-0943. 2002.
12. Girardi, R.; Lindoso, A. "DDEMAS: A Domain Design Technique for Multi-agent Domain Engineering". In: The Seventh International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2005) At The 24th International Conference on Conceptual Modeling (ER 2005), 2005, Klagenfurt. Proceedings of ER Workshops, LNCS 3770. Berlin Heidelberg: Springer-Verlag, 2005. p. 141-150.
13. Girardi, R. and Lindoso, A. "An Ontology-based Knowledge Base for the Representation and Reuse of Software Patterns". ACM Software Engineering Notes, New York, v. 31, n. 1, 2006.
14. Girardi, R. and Balby, L.. A Domain Model of Web Recommender Systems based on Usage Mining and Collaborative Filtering. Requirements Engineering Journal, Vol. 12, N. 7, Jan. 2007.
15. Girardi, R. and Lindoso, A., A Multi-agent Architectural Model for Web Recommender Systems based on Usage Mining and Collaborative Filtering, submitted paper, 2006.
16. Girardi, R., Balby, L. and Oliveira, I. "A System of Agent-based Patterns for User Modeling based on Usage Mining", Interacting with Computers, v. 17, n.5, pp. 567-591. Sept. 2005.
17. Girardi, R., and Faria, C.: An Ontology-Based Technique for the Specification of Domain and User Models in Multi-Agent Domain Engineering. CLEI Electronic Journal, V. 7, N. 1, Pap. 7, June, 2004.
18. Girardi, R.; Ibrahim, B. Using English to Retrieve Software. The Journal of Systems and Software, v. 30, n. 3, p. 249-270, 1995.
19. Girardi, R.; and Lindoso, A., An Ontology-driven Technique for the Architectural and Detailed Design of Multi-Agent Frameworks, In: Kolp, M.; Bresciani, P.; Henderson-Sellers, B.; Winikoff, M. (Org.). Agent-Oriented Information Systems III, Lecture Notes in Artificial Intelligence., Ed. Springer-Verlag, pp. 124-139. Berlin. 2006.
20. Gruber, T. R "Toward Principles for the Design of Ontologies used for Knowledge Sharing", International Journal of Human-Computer Studies. Nº 43, pp. 907-928. 1995.
21. Harsu, M.: A Survey of Domain Engineering. Report 31, Institute of Software Systems, Tampere University of Technology, December 2002.
22. Lindoso, A., Girardi, R. The SRAMO Technique for Analysis and Reuse of Requirements in Multi-agent Application Engineering. IX Workshop on Requirements Engineering, Cadernos do IME, UERJ Press, v. 20, 41-50. Rio de Janeiro. 2006.
23. Marinho, Leandro B.: A Multi-Agent Framework for Usage Mining and User Modeling-based Web Personalization. Master dissertation, Federal University of Maranhão - UFMA - CPGEE, 2005. (In Portuguese)
24. Odell, J., Parunak, H.V.D. and Bauer, B.: Extending UML for Agents. Proc. of the Agent-Oriented Information Systems Workshop at the 17th National Conference on Artificial Intelligence, accepted role, AOIS Workshop at AAAI pp. 3-17, 2000.

A Three Level Multi-agent Architecture to Foster Knowledge Exchange

Juan Pablo Soto, Aurora Vizcaíno, Javier Portillo-Rodríguez, Mario Piattini
Alarcos Research Group, Information Systems and Technologies Department, UCLM-Soluziona Research and Development Institute, University of Castilla – La Mancha
Ciudad Real, Spain
jpsoto@proyectos.inf-cr.uclm.es, {aurora.vizcaino, mario.piattini}@uclm.es, javier.portillo@alu.uclm.es

Abstract

This paper proposes a multi-agent architecture based on the concepts of communities of practice and reputation to manage knowledge management systems. The main goal of this proposal is to emulate the behavior of communities of practice where people exchange information and in this way attempt to foster the reuse of information in organizations which use knowledge base or knowledge management systems.

1. Introduction

The need to support knowledge processes in organizations has always existed. However, its importance has definitely increased in the last few years. Recently, the concept of knowledge management suggests a paradox since compared with traditional production factors knowledge is so complex, scattered and hidden that it is rather complicated to manage it.

On the other hand, traditional Knowledge Management Systems (KMS) have received certain criticism as they are often implanted in companies overloading employees with extra work; for instance, employees have to introduce information into the KMS and worry about updating this information. As a result of this, these systems are sometimes not greatly used by the employees since the knowledge that these systems have is often not valuable or on other occasions the knowledge sources do not provide the confidence necessary for employees to reuse the information. For this purpose, companies create both social and technical networks in order to stimulate knowledge exchange. An essential ingredient of knowledge sharing information in organizations is that of "community of practice", by which we mean groups of people with a common interest where each member contributes knowledge about a common domain [12]. The ability of a community of practice to create a friendly environment for individuals with similar interests and problems in which they can discuss a common subject matter encourages the transfer and creation of new knowledge. Many companies report that such communities help reduce problems caused by lack of communication, and save time by "working smarter"[13]. For these reasons, we consider the modelling of communities of practice into KMS as an adequate method by which to provide these systems with a certain degree of control to measure the confidence and quality of information provided by each member of the community.

In order to carry this out, we have designed a multi-agent architecture in which agents try to emulate human behaviour in communities of practice with the goal of fostering the use and exchange of information where intelligent agents suggest "trustworthy knowledge" to the employees and foster the knowledge flow between them.

The remainder of this work is organized as follows. The next section presents two important concepts that exist in the development of our work (agents and trust). In Section Three the multi-agent architecture proposed to manage trustworthy KMS is presented..In Section Four a prototype developed to evaluate our architecture is explained in order to illustrate how it could be used. Finally, conclusions are presented in Section Five.

2. Agents and trust

Because of the importance of knowledge management, tools to support some of the tasks related to knowledge management have been developed. Different techniques are used to implement these tools. One of them, which is proving to be quite useful, is that of intelligent agents [10]. Software agent technology can monitor and coordinate events, meetings and disseminate information [1]. Furthermore, agents are proactive; this means they act automatically when it is necessary. The autonomous behavior of the agents is critical to the goal of this research since agents help to reduce the amount of work that employees have to perform. On the other hand one of the main advantages of the agent paradigm is that it constitutes a natural metaphor for systems with purposeful interacting agents, and this abstraction is close to the human way of thinking about their own activities [14]. This foundation has led to an increasing interest in social aspects such as motivation, leadership, culture or trust [3]. Our research is related to this last concept of "trust" since artificial agents can be made more robust, resilient and effective by providing them with trust reasoning capabilities.

For agents to function effectively in a community, they must ensure that their interactions with the other agents are trustworthy. For this reason it is important that each agent is able to identify trustworthy partners with which they should interact and untrustworthy correspondents with which they should avoid interaction. The stability of a community depends on the right balance of trust and distrust.

3. Our proposal

The goal of this work is to provide a reputation model for communities of practice using a multi-agent architecture that:

- Assists employees in identifying trustworthy entities.
- Gives artificial agents the ability to reason about the trustworthiness of other agents or of a knowledge source.
- Encourages knowledge exchange between the community members.
- Provides the confidence necessary to foster the usage of information and knowledge of the KMS.

To do this, we first need to define a conceptual model for the agent that permits it to obtain the level of confidence of an information source or of a provider of knowledge.

The conceptual model of the agent is based on two related concepts: trust and reputation. The former can be defined as confidence in the ability and intention of an information source to deliver correct information [2] and the latter as the amount of trust an agent has in an information source, created through interactions with information sources. There are other definitions for these concepts [4, 6]. However, we have presented the most appropriate for our research since the level of confidence in a source is based on, in our case, previous experience of this.

The reputation of an information source not only serves as a means of belief revision in a situation of uncertainty, but also serves as a social law that obliges us to remain trustworthy to other people. Therefore, people, in real life in general and in companies in particular, prefer to exchange knowledge with "trustworthy people" by which we mean people they trust. People with a consistently low reputation will eventually be isolated from the community since others will rarely accept their justifications or arguments and will limit their interaction with them. It is for this reason that the remainder of this paper deals mainly with reputation.

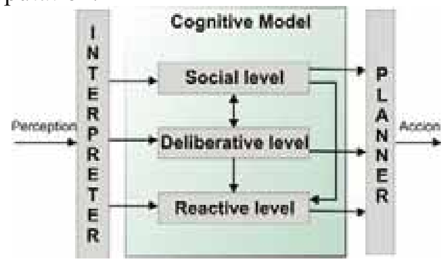


Figure 1. General architecture

Taking the concepts reputation and communities of practice into account we designed a multi-agent architecture which is composed of three levels (see Figure 1): reactive, deliberative and social. The reactive and deliberative levels are considered by other authors as typical levels that a multi-agent system must have [9]. On the other hand, the social level is not frequently considered in an explicit way, despite the fact that these systems (multi-agent systems) are composed of several individuals, interactions between them and plans constructed by them. The social level is only considered in those systems that try to simulate social behaviour or those that represent a more generic architecture prepared to represent this or other behaviour. Since we wish to emulate human feelings such as trust, reputation and even intuition we have added a social level that considers the social aspects of a community which takes into account the opinions and behaviour of each of the members of the community. Other previous works have also added a social level, for instance in [5] the author tries to emulate human emotions such as fear, thirst, bravery and also uses an architecture of three levels: reactive, deliberative and social.

In the following paragraphs we will explain each of these levels in detail.

Reactive level: This is the agent's capacity to perceive changes in its environment and to respond to these changes at the precise moment at which they happen. It is in this level when an agent will execute the request of another agent without any type of reasoning. That is to say, the agent must act quickly in the face of critical situations.

Deliberative level: The agent may also have a behaviour which is oriented towards objectives, that is, it takes the initiative in order to plan its performance with the purpose of attaining its goals. In this level the agent would use the information that it receives from the environment, and from its beliefs and intuitions, to decide which is the best plan of action to follow in order to fulfill its objectives.

Social level: This level is very important as our agents are within communities and they exchange information with other agents. Thanks to this level they can cooperate with other agents by using an expressive language. This language analyzes the present situation, considering the goals and interests of the agent and structure solutions in the form of plans.

Two further important components of our architecture are the *Interpreter* and the *Planner*. The former is used to perceive the changes that take place. The planner indicates how the actions should be executed.

In this paper only the deliberative architecture is described due to space restrictions.

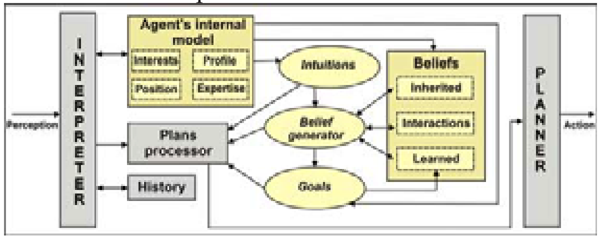


Figure 2. Deliberative architecture.

The components of the Deliberative Architecture are (see Figure 2);

Agent's internal model: As an agent represents a person in a community this model stores the user's features. Therefore, this module stores the following parts:

- The *interests*. This part is included in the internal model in order to make the process of distributing knowledge as fast as possible. That is, the agents are able to exchange knowledge automatically, checking whether their stored knowledge matches with the interests of other agents. This behaviour fosters knowledge sharing and reduces the amount of work employees have to do because they receive knowledge without making searches.
- *Expertise*. This term can be briefly defined as the skill or knowledge of a person who knows a great deal about a specific thing. This is an important factor since people often trust in experts more than in novice employees.
- *Position*. Employees often consider information that comes from a boss as being more reliable than that which comes from another employee in the same (or a lower) position as him/her [11]. In an enterprise this position can be established in different ways, for instance by using an organizational diagram or classifying the employees according to the knowledge that a person has.

Such different positions inevitably influence the way in which knowledge is acquired, diffused and eventually transformed in the local area. Because of this these factor will be calculated in our research by taking into account a weight that can strengthen this factor to a greater or to a lesser degree.

History: This component stores the interactions of the agents with the environment.

Belief generation: This component is one of the most important of the cognitive model because it is in charge of creating and storing the agent's knowledge. Moreover, it defines the agent's beliefs.

Beliefs: The beliefs module is composed of the inherited beliefs of the organization, lessons learned, and agents' interactions. Inherited beliefs are the organization's beliefs that the agent receives. For instance: an organizational diagram of the enterprise, the expertise of each employee, the philosophy of the company or community. Lessons learned are the lessons that the agent obtains while it interacts with the environment This interaction can be used to establish parameters in order to know what the agent can trust (agents or knowledge sources).

Intuitions: The intuitions are beliefs that have not been verified but which it thinks may be true. According to [7] intuition has not yet been modelled by agent systems. In this work we have tried to adapt this concept by comparing the agents' profiles to obtain an initial value of intuition that can be used to form a belief about an agent.

Goals: The goals are formed by the objectives of the agent. For instance, one of the goals of each member of a community of practice is knowledge exchange. The goals are defined in accordance with the community or group in which the agent interacts

4. Prototype

In order to test our architecture we have developed a prototype system into which people can introduce documents and where these documents can also be consulted by other people. The goal of this prototype is to allow software agents to help employees to discover the information that may be useful to them thus decreasing the overload of information that employees often have and strengthening the use of knowledge bases in enterprises. In addition, we try to avoid the situation of employees storing valueless information in the knowledge base.

The main feature of this system is that when a person searches for knowledge in a community, and after having used the knowledge obtained, that person then has to evaluate the knowledge in order to indicate whether:

- The knowledge was useful.
- How it was related to the topic of the search (for instance a lot, not too much, not at all).

To design this prototype we have designed a *User Agent* and a *Manager Agent*. The former is used to represent each person that may consult or introduce knowledge in a knowledge base. Therefore, the *User Agent* can assume three types of behavior or roles similar to the tasks that a person may carry out in a knowledge base. The User Agent plays one role or another depending upon whether the person that it represents carries out one of the following actions:

- The person contributes new knowledge to the communities in which s/he is registered. In this case the User Agent plays the role of **Provider**.
- The person uses knowledge previously stored in the community. Then, the User Agent will be considered as a **Customer**.
- The person helps other users to achieve their goals, for instance by giving an evaluation of certain knowledge. In this case the role is of a **Partner**. So, Figure 3 shows that in community 1 there are two User Agents playing the role of Partner, one User Agent playing the role of Consumer and another being a Provider.

The second type of agent within a community is called the *Manager Agent* (represented in black in Figure 3) which must manage and control its community.

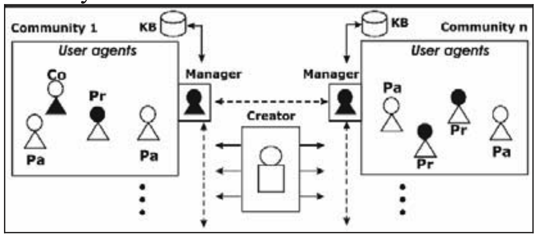


Figure 3. Communities of agents

The prototype provides the options of using community documents and updating reputation values, proposing new topics in the community, etc. Due to space limitations, we shall now describe only the following situation:

Using community documents and updating reputation values. People can search for documents in every community in which they are registered. When a person searches for a document relating to a topic his/her User Agent consults the Manager Agent about which documents are related to their search. Then, the Manager Agent answers with a list of documents. The User Agent sorts this list according to the reputation value of the authors, which is to say that the contributions with the best reputations for this Agent are listed first. On the other hand, when the user does not know the contributor then the User Agent consults the Manager Agent about which members of the community know the contributors. Thus, the User Agent can consult the opinions that other agents have about these contributors, taking advantage of other agents' experience. To do this the Manager consults its interaction table and responds with a list of the members who know the User Agent. Then, this User Agent contacts each of them. If nobody knows the contributors then the information is listed, taking their authors' expertise and positions into account. In this way the User Agent can detect how worthy a document is, thus saving employees' time, since they do not need to review all the documents related to a topic but only those considered most relevant by the members of the community or by him/herself according to previous experience with the document or its authors.

Once the person has chosen a document, his/her User Agent adds this document to its own document list (list of consulted documents), and if the author of the document is not known by the person because it is the first time that s/he has worked with him/her, then the Community Manager adds this relation to the interaction table. This step is very important since when the person evaluates the document consulted, his/her User Agent will be able to assign a trustworthy value to that document. The formulas used to approach this have been explained in [8] (they have been omitted due to space constrains).

5. Conclusions

Communities of practice have the potential to improve organizational performance and facilitate community work. Because of this we consider it important to model people's behavior within communities with the purpose of imitating the exchange of information in companies that are produced in those communities. Therefore, we are attempting to encourage the sharing of information in organizations by using knowledge bases. To do this we have designed a multi-agent three-layer architecture where the artificial agents use similar parameters to those of humans in order to evaluate knowledge and knowledge sources. These factors are: reputation, expertise, position, previous experience and even intuitions.

This approach implies several advantages for organizations as it permits them to identify the expertise of their employees and to measure the quality of their contributions. Therefore, it is expected that a greater flow of communication will exist between them which will consequently produce an increase in their knowledge.

6. Acknowledgement

This work is partially supported by the ENIGMAS (PIB-05-058), and MECENAS (PBI06-0024) project, Junta de Comunidades de Castilla-La Mancha, Consejería de Educación y Ciencia, both in Spain. It is also supported by the ESFINGE project (TIN2006-15175-C05-05) Ministerio de Educación y Ciencia (Dirección General de Investigación)/ Fondos Europeos de Desarrollo Regional (FEDER) in Spain.

7. References

[1] Balasubramanian, S., Brennan, R. and Norrie, D., An Architecture for Metamorphic Control of Holonic Manufacturing Systems. *Computers in Industry*, Vol. 46, No. 1, 2001, pp. 13-31.

[2] Barber, K. and Kim, J., Belief Revision Process Based on Trust: Simulation Experiments. In *4th Workshop on Deception, Fraud and Trust in Agent Societies*, Montreal Canada, 2004.

[3] Fuentes, R., Gómez-Sanz, J. and Pavón, J., A Social Framework for Multi-agent Systems Validation and Verification. Wang, S. et al Eds. ER Workshops 2004, Springer-Verlag, LNCS 3289, 2004, pp. 458-469.

[4] Gambetta, D., Can We Trust Trust? In *Gambeta, D., ed.: Trust: Making and Breaking Cooperative Relations*, Basil Blackwell, New York, 1990, pp. 213-237.

[5] Imbert, R. and de Antonio, A., When Emotion Does not Mean Loss of Control. In *Lecture Notes in Computer Science*, T. Panayiotopoulos, J. Gratch, R. Aylett, D. Ballin, P. Olivier, and T. Rist, Eds. Springer-Verlag, London, 2005, pp. 152-165.

[6] Marsh, S., Formalising Trust as a Computational Concept. PhD Thesis, University of Stirling, 1994.

[7] Mui, L., Halberstadt, A. and Mohtashemi, M., Notions of Reputation in Multi-Agents Systems: A Review. *International Conference on Autonomous Agents and Multi-Agents Systems* (AAMAS'02), 2002, pp. 280-287.

[8] Soto, J.P., Vizcaíno, A., Portillo, J. and Piattini, M., Knowledge Management Systems with Reputation and Intuition: What for?, *Accepted to be published in International Conference on Enterprise Information Systems* (ICEIS'07), 2007.

[9] Ushida, H., Hirayama, Y. and Nakajima, H., Emotion Model for Life like Agent and its Evaluation. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference (AAAI'98 / IAAI'98)*. 1998. Madison, Wisconsin, USA, 1998, pp. 62-69.

[10]van-Elst, L., Dignum, V. and Abecker, A., Agent-Mediated Knowledge Management. In *International Symposium AMKM 2003*, Stanford, CA, USA, Springer, 2003.

[11]Wasserman, S. and Glaskiewics, J., Advances in Social Networks Analysis. *Sage Publications*, 1994.

[12]Wenger, E., Communities of Practice: Learning Meaning, and Identity, *Cambridge U.K.*: Cambridge University Press, 1998.

[13]Wenger, E., McDermott, R., and Snyder, W., Cultivating Communities of Practice, *Boston: Harvard Business School Press*, 2002.

[14]Wooldridge, M. and Ciancarini, P., Agent-Oriented Software Engineering: The State of the Art. In *Wooldridge M., Ciancarini, P. (Eds.), Agent Oriented Software Engineering*, Springer-Verlag, LNAI 1975, 2001.

An Agent Based System for Search in Distributed Environments

LI SA, YONG-SHENG DING
College of Information Sciences and Technology, Shanghai University, Shanghai
201620, P. R. China
sali@ustc.edu, ysding@dhu.edu.cn

Abstract: The obvious problems that have infected the popularity of peer-to-peer (P2P) systems are effective information search and traffic caused by the blind flooding-based search. In this paper, we have concentrated on developing an agent-based model for controlling query messages that are represented as agent; the Ecologically Inspired Distributed Search (EIDS), which derives its inspiration from natural ecosystem, is presented. As an experiment result, we compare its performance against the well-known k-random walker approach.

Key-Words: Agent-based System, Peer-to-Peer, Distributed Search, Modeling and Simulation, Algorithm

1 Introduction

There has been a growing interest in peer-to-peer networks since the initial success of some very popular file-sharing applications such as Napster and Gnutella. A peer-to-peer (P2P) network is distributed systems based on the concept of resource sharing by direct exchange between peer nodes (i.e., nodes having same role and responsibility). Exchanged resources include content, as in popular P2P file s are end systems in the Internet and maintain information about a set of other nodes (called neighbors) in the P2P layer. These nodes form a virtual overlay network on top of the Internet. Each link in a P2P overlay corresponds to a sequence of physical links in the underlying network.

Early search mechanisms primarily used flooding or k-random walk [1] algorithms. In the flooding approach, each node propagates the query to all its neighbors. On a receipt of a query, the node searches in its local repository. If the object is found, it informs the query originator and further search in that path terminate. If not, the node further forwards the query to all its neighbors.

The flooding method generates a large amount of network traffic. To overcome this problem, random walk algorithms are often used. In Random Walks, the requesting node sends out k query messages to an equal number of randomly chosen neighbors. Each of these messages follows its own path, having intermediate nodes forward it to a randomly chosen neighbor at each step. These queries are also known as walkers. Random Walkers cannot learn anything from its previous successes or failures, displaying high variability in all ranges of requests.

The environments in which P2P applications are deployed exhibit extreme dynamism in structure and load. In order to deal with the scale and dynamism that characterize P2P systems, a paradigm shift is required that includes self-organization, adaptation and resilience as fundamental properties. Complex adaptive systems (CAS) commonly used to explain the behavior of certain ecological [7] and social systems can be the basis of a new programming paradigm for P2P applications. Here we are concerned with the development of an ecosystem-inspired approach to the design of agent systems for searching in Unstructured Peer-to-Peer Networks.

Ecosystems have been a source of inspiration to a number of previous developers of agent systems [2,3,4]. Moukas [3] employs ecosystem inspired ideas in the Amalthaea architecture for information filtering. Paul Marrow and colleagues [2] have advocated an “information ecosystems” approach to support a variety of information management applications. The interaction of many simple agents was used to solve problems, rather than sophisticated processing at the individual agent level. Doran [4] provides a contrasting view of ecosystem-inspired agent systems, using agent models to assist in the management of natural ecosystems.

This paper developed an ecology-based model for managing a number of search agents on the P2P networks that can provide decentralized distributed robust control of agents in the dynamic P2P network environments. This kind of agents does provide a viable means of performing network resource discovery, which makes P2P more practicable. This paper is organized as follows: Sec.2 defines the model and states the attributions of agents; a detailed description of all aspects related to the EIDS algorithm. Sec.3, next to showing the performance of the algorithm in comparison to other approaches, presents experimental results and analysis. Sec.4 provides a discussion of related work on agent based model used for searching in peer-to-peer networks.

2 System Model

This section is divided into two parts. In the first part (Section 2.1) we describe the framework chosen to model the P2P environment. In the second part (Section2.2) we describe the ecologically inspired search algorithm.

2.1 Environment Definition

The model framework involves three concepts: peers, neighbors and search agents. The peers are typically computing devices that can maintain some state, or perform computations. A peer has a set of neighbors and is able to send search agents to its neighbors only.

The factors which are important for simulating P2P environments are the overlay topology. The overlay topology consists of a two-dimensional grid responsible for maintaining the neighborhood connections between the peers in the P2P network. Due to the grid structure, each peer residing in a particular node has a fixed set of eight neighbors. Each grid is conceived to be containing a heterogeneous distribution of peers of the P2P network with one or more distinct resources. We assume that there are 1024 unique resources; each of them can be represented by a 10-bit binary token as the resource information (RI). All agents take with also a 10-bit binary token as their searching information (SI) when they move across the environment. Similarity between a RI and a SI is measured by the number of bits that are identical. That is, sim(RI, SI) = d –HD(RI, SI), where HD is the Hamming distance between RI and SI. Zipf's distribution [5] is chosen to distribute each of the 1024 unique resources in the network. These resources are gathered and consumed by the agent to survive. Each peer may be visited by at most one agent at any time. At any time during the evolution of the model, each agent has a distinct location on the environment, characterized by the peer the agent visits, and a distinct field of view, measured in grids. Each agent has perfect knowledge of resource levels within its field of view; an agent has no knowledge or memory of resource levels outside its field of view. All the agents have a fixed time to live. In addition, an agent has a characteristic metabolic rate for the resource it finds and consumes. Any resources found by an agent can be retained as energy without constraint. If an agent's energy diminishes to zero, the agent dies from starvation.

2.2 Environment Definition

The ecologically inspired search is a distributed algorithm in which queries are represented as agents. The agents are created at the peer who issued the query and travel over the P2P network in which peers are arranged in a grid-like topology, as in the Swarm simulator [6]. At random times, each agent makes a random number of hops along the P2P network.

The search in our P2P network is initiated from the user peer. The user emanates search agents (message packets) to its neighbors-the packets are thereby forwarded to the surroundings. The method of spreading search agents forms the basis of the algorithm.

Ecosystems usually have several attractive qualities (such as dynamic decentralized control, self regulation, no single point of failure, robustness, and stability) that we require for P2P system. We propose a solution to the problem of controlling the number of agents appropriate for a search which is inspired by large ecosystems (Figure.1):

1. Each kind of resources (Resources could be files in a file-sharing system or CPU cycles in a computational grid.) in P2P system will be associated with energy.

2. Agents finding a resource successfully will collect the energy associated with the resource.

3. Agents consume energy over time to sustain their existence.

4. Agents that exceed the time to live or exhaust their supply of energy die.

5. Abundance of energy can cause a new agent to spawn.

All search operations are controlled by a set of micro-scale rules. The spatial distribution of resources are searched, the energy associated with the resources are gathered and consumed by the agents to survive.

An agent has two dynamic attributes behavioral attributes:

- The current energy level (EL): The current energy level of the agent. If this falls below zero the agent will die.
- The agent's age (AA): Measured in number of hops.

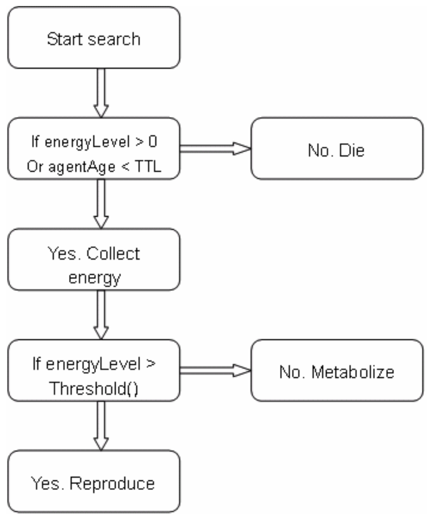


Figure.1 Flowchart of agent behavior

In addition, an agent has a number of static attributes that do not change during their lifetime:

- Search Profile (SP): Built from the informational interest of the peer by which the agent is made.
- Metabolic Rate (MR): The amount of energy it consumes during each hop.
- Energy received at Birth (EB): The energy the agent is born with.
- Energy need for reproduce (ER): The energy the agent needs to attain before it can reproduce.
- Time to Live (TTL): The agent's maximum possible age. It represents the maximum hop-distance a search agent can reach before it gets discarded.

As time evolves, four micro-scale behavioral rules control the search agents-*Query Start Rule (QSR), Resource Search Rule (RSR), Reproduce Rule (RR)* and *Die Rule (DR)*. These rules are explained as following:

Query Rule: A query is initiated by a randomly selected peer who requests for some kinds of resources. To obtain an answer to the request, agents are generated by the peer and flooded in its neighboring peers. The **Query Start Rule** is elaborated below.

Rule 1 *QR: Generate Search Agent (*SA)
/*Agents are generated by the peer in response to user requests. */

The peer emanates search agent (SA) to its neighbors.

Search Rule: Once the search agents are emanated, they hop from peer to peer in subsequent time steps. Whenever a search agent moves to a peer, it checks whether the peer has earlier been visited or not. If not, then the agent moves to the peer. In this connection, each peer maintains a field named visit (V), a field named resource profile (RP) and a field named new energy (NE). A successful search is reported if the required resource can be found in this peer. A flag will be set to true to indicate a successful search. An algorithmic form for the resource search rule (RPR) is presented as follow.

```
Rule 2 SR: If (Search agent (SA)) Start
        AA++;
        V++;
         If ((V = = 1) AND (SP = RP)
                /*Report a match, V = 1 indicates first
                time visit by an agent.*/
        flag = true;
        EL = EL + NE;
        Update;
```

Reproduce Rule: Once the current energy level of the search agent exceeds a threshold, the agent will spawn a new one and splitting its current energy in half.

```
Rule 3 RR: If (Search agent (SA)) Start
          If (EL >= ER) /*The agent get enough energy
                              for reproduce.*/
         Produce a new agent;
         EL = EL/2;
```

Die Rule: When the agent's age reach the time to live (TTL) value, or the current energy level of the agent falls below zero, the agent will die.

```
Rule 4 DR: If (Search agent (SA)) Start
        If ((AA >= TTL) OR (EL <= 0))
          The agent die;
```

2.3 Performance Metrics

In this paper we focus on efficiency aspects of the processes solely, and use the following simple metrics in our abstract p2p networks.

- Hit rate is defined as the number of resources found for each agent within a given period of time.
- Average number of hops per successful query. This parameter depends on the topology of P2P network as well as on how effectively search mechanism uses it. The less hops is required in average to find requested data, the less traffic is generated and the less time is required for search.
- Population of agents. The number of agents needs to find the requested resource according to the distribution of resource in the uncertain network environments.

Based upon the above mentioned model and metric definition, we now present the experimental results. Simulation runs on Pentium 2.3 GHz with 1GB RAM under windows XP.

3 Simulation Result and Analysis

The experimental results illustrate the efficiency of the algorithms and the effect of controlling the number of agents dynamically based on ecologically inspired control mechanism. This mechanism is completely distributed, executed locally and uses only locally available information. Thus, no globally available information is required. The emergent behavior resulting from the individual localized control decisions will yield an optimal, or sufficiently optimal, solution at the global level. For comparison, we also simulate experiments with k-random walk. The time-step experiment is elaborated next.

3.1 Search Efficiency

The search is initiated by a randomly selected node and the number of resource found each time-step from the commencement of the search is calculated. The value of resource hit rate provides the indication of search efficiency. Figure. 2 shows the result of running the time-step experiment for 20 generations (1 generation is 100 time-steps). The time-to-live parameter is set to 25, and k is set to 12, grid size is 100×100.

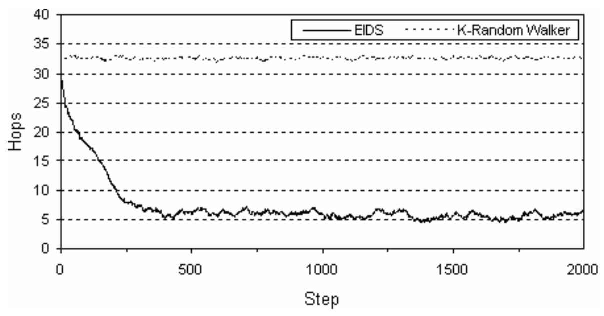


Figure.2 Average number of hops per successful query

We expect that if an agent employs spawn strategy, the total number of agents would increase, the hops required to complete a search would decrease. When EIDS and K-Random Walk are compared, EIDS requires much fewer hops (6 hops) than K-Random Walk (32 hops). In fact, K-Random Walk constantly requires a large number of hops.

The decision for agents spawn strategy is based on a parameter $\rho \in [0, 1]$. For example, if parameter is set to 0.8, each agent will employ the spawn strategy in 80% of the cases. If the parameter is set to 1, then the agent will spawn a new agent whenever its energy level is above a threshold; when the parameter is set to 0, the agent will never spawn any new one, in the fact, it employs the k-random walk strategy.

The hit rate is dependent on parameter not only in the start-up phase, but also in the converged phase (Figure.3). The more search agents in the network, the higher the hit rate. The best result can be reached when setting $\rho= 1$. After ten generation, 2.8 resources on average are found in this case. All the five curves employing spawn strategy converge to the same limit over time. The worst result is obtained when setting $\rho= 0$, which is the k-random walk case, the performance stays constant with on average 0.6 resources are found. We see that search efficiency of spawn agents is almost 3-4 times higher than that of k-random walkers.

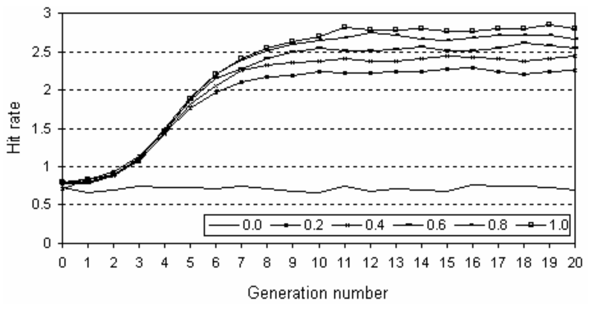


Figure.3 Hit rate per generation

3.2 Ecosystem Inspired Control of Agent's Population

The system of search agents and the environment they inhabit, i.e., the P2P network, consist of an information ecosystem. All peers can be seen as both information producers and consumers; as consumers, peers send agents (queries) for searching information resources they required, these living agents survive in the context of limited information resources they can find in the network environment. Agent population is determined by the resources of P2P network, the "carry capacity" of the networked information environment, that is, by the size of the relevant set for the given query (Figure.4).

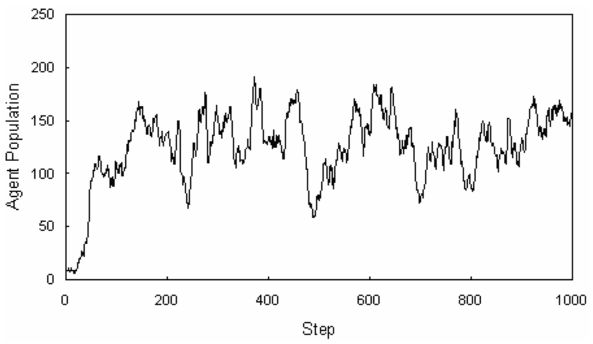


Figure.4 Agent population over time (TTL=6, Size= 30×30)

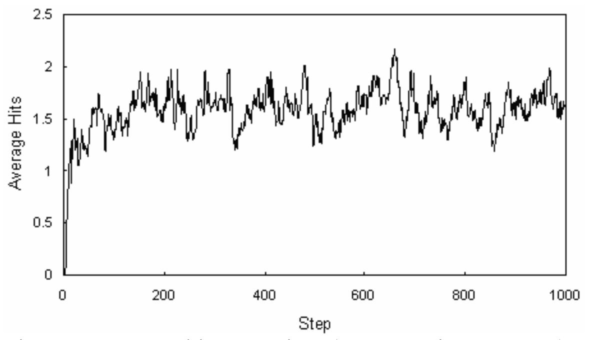


Figure.5 Average hits over time (TTL=6, Size= 30×30)

As the number of agents in the system grows at the beginning of simulation, the resource they find increase rapidly; when the resource the agents can find reach the max mount of resource, agent population fluctuates within a definite scope, that's to say the population reach the carry capacity of network environment, and the hit rate maintain a steady average change (Figure.5).

4 Conclusions

The most important functionality of P2P network is search. In this paper, we have concentrated on developing an agent-based model for controlling query messages that are represented as agent; a search algorithm which derives its inspiration from natural ecosystem is presented. Experiment results above show that this ecologically inspired algorithm is much more efficient search method than k-walker random walk. Each additional step in the search increases the number of nodes visited by only a constant. So exponentially increased over load on each visited node by flooding can be avoided. The basic strengths displayed by the EIDS algorithm need to be further explored and developed, by applying it in more realistic circumstances in the near future.

References:

[1] Q. Lv, P. Cao, E. Cohen, and S. Shenker. Search and replication in unstructured peer-to-peer networks. In *Proceedings of the 16th ACM Conference on Supercomputing*, 2002.

[2] C. Hoile, F. Wang, E. Bonsma, and P. Marrow. Core specification and experiments in diet: a decentralised ecosystem-inspired mobile agent system. In *Proc. of AAMAS 2002*, pp. 623-630.

[3] Moukas, A. Amalthaea: Information Discover and Filtering using a Multiagent Evolving Ecosystem. *Proc. Conf. Practical Applications of Agents and MultiAgent Technology*, 1997.

[4] G. K. Zipf. *Psycho-Biology of Languages.* Houghton-Mfflin, 1935.

[5] The SwarmWiki environment, Center for the Study of Complex Systems, the University of Michigan, http://www.swarm.org/wiki.

[6] Pack Kaelbling, L., Littman, M.L., & Moore, A.W. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 1996, pp. 237-285.

[7] N. Minar, M. Gray, O. Roup, R. Krikorian, and P. Maes. Hive: distributed agents for networking things. In: *Proceedings of ASA/MA '99*, 1999.

Tree Hash Under Concurrency Control

Kyosuke YASUDA Takao MIURA
Dept.of Elect. and Elect. Engineering, Hosei University
Kajinocho 3-7-2, Koganei, Tokyo, Japan

Abstract

Dynamic Hash allows us to adjust the size of hash space dynamically, i.e., we can change the space size dynamically according to volume of data to maintain efficiency for retrieval and insert. We have proposed *Tree Hash* (Tree Hash,TH) as a new hash structure so far. By the technique, the hash space grows smoothly and hierarchically while the space avoids severe damage to efficiency caused by bucket split during batch (consecutive) insertion. Also we can relieve excessive I/O caused by overflow splitting. In this investigation, we discuss concurrency control mechanism for tree hash technique to improve total throughput. Here we examine two kinds of parallelism, one by key level and another bucket level. We show how to implement the mechanism based on a framework of *compensation transactions*.
Keywords: Tree Hash, Concurrency Control, Compensation Transaction

1 Introduction

Compared to single user execution, it is hard tremendously for us to control concurrent access: multiple updates may arise at the same time, one should undo some change while other might retrieve them. In *theory of transaction*, several important aspects have been focused on atomicity, consistency, isolation, and durability, called *ACID*, as well as deadlock issues.

For the purpose of highly efficient data management, many kinds of *hash techniques* have been proposed so far. In fact, the technique allows us to retrieve and update data in $O(1)$. This is especially useful for online real-time environment. *Dynamic Hash* allows us to adjust the size of hash space dynamically, i.e., we can change the space size dynamically according to volume of data to maintain efficiency for retrieval and insert while keeping constant *density*[1] We could see many implementation based on this approach such as *Linear Hash*(LH)[1].

[1]Note "density" means the ratio of the practical number of data to the space size.

To obtain efficient data management under parallel environment, we should examine theory of transaction to the hash technique. In [2], they have discussed linear hash under parallel environment. They introduce 3 kinds of locks, *read-lock*, *selective-lock*, and *exclusive-lock* and 5 kinds of operations on a lock compatibility matrix for retrieval and update to buckets. One of their problems is that deadlock arises very often in the case of small hash space. By using multi-level lock [3], we can improve the problem. The main idea comes from using both key lock and page lock. Also, by compensation transaction, we can manage recovery process efficiently.

It is well-known that there exist severe problems in LH to insert buckets. In fact, when the space grows linearly, the bucket of just overflowed remains unchanged but the space must contain longer overflow-chains. We have proposed *Tree Hash* by which we can relieve the situation of interest[4].

In this investigation we propose a concurrency control suitable for tree hash. Here we introduce three kinds locks and two levels of locks to manage concurrency control similar to the one for linear hash. We show, by some experimental results, there happen fewer deadlocks but better efficiency.

In section 2 we quickly review basic ideas of Tree Hash, and in section 3 we define concurrency control suitable for hash structures. Section 4 contains some experiments, analysis and the comparison with other approach. We conclude our investigation in section 5.

2 Tree Hash

We have proposed *Tree Hash* (TH) to improve bucket-bias issues caused in LH. The basic idea is that we split the *overflowed* bucket hierarchically.

The main difference of LH and TH comes from the two points, hashed value calculation and bucket splitting. In TH, we use L that corresponds to level. L is defined as the maximum among the level values. In TH, L are called as *root variables*. We manage bucket address through a *bucket address table*.

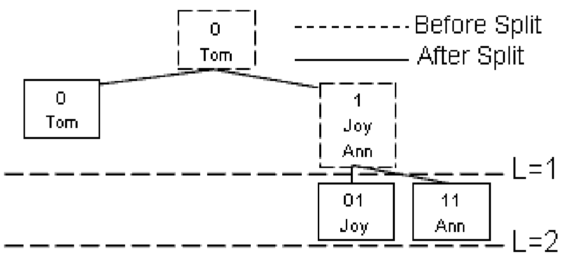


Figure 1. Bucket Split

Let $j = L$ initially. Then we obtain a hashed value a through an algorithm: $a \leftarrow h_j(C) = C \, mod \, 2^j$; If there exists no bucket address a in the bucket address table, we put $j \leftarrow j - 1$ and calculate a again. We repeat the process until we see a in the bucket address table. That is, once we examine a bucket with the longest bit-length but we don't find the data, we cut down the length and apply the process again.

Because we don't have growth position in TH for bucket splitting, we can split any bucket at any time. Since we can split any bucket according to the first bit, we may have bucket splitting as a tree structure in Figure 1. Note that the tree means *history* of bucket-splitting, but *not* a collection of many intermediate bucket-nodes organized into a tree.

Because buckets in TH are independent with each other, we can apply arbitrary conditions for splitting. In this investigation, we assume following conditions:
(1) There exist overflows in a bucket and the density is greater than a given load factor.
(2) There exist overflows in a bucket and the level of the bucket is less than or equal to a threshold value.
(3) The ratio of data in the bucket just after splitting is greater than a threshold value. When we split a bucket, we get the following values using the level m of a bucket at a: $a' \leftarrow a + 2^m; m = m + 1$; We split a bucket at a into two buckets at a and a', move appropriate data to a' and we add a' to the bucket address table.

EXAMPLE 1 Let us show an example of inserting key `Ann` when there exist two buckets at 0 and 1. Let us assume that we get hashed value 1 and insert the key in a bucket at 1, and that we see a bucket at 1 is divided into two buckets. A bucket 1 is divided into a bucket 01 and a bucket 11 (Figure 1). Since a bucket address has bit-length 2, the level L becomes 2.

We can implement Tree Hash structure under distributed environment. For more detail see [4].

	read-lock	selective-lock	exclusive-lock
read-lock	Yes	Yes	No
selective-lock	Yes	No	No
exclusive-lock	No	No	No

Table 1. Lock Type

3 Concurrency Control of Hash Structure

3.1 Concurrency Control

As described before, TH is different from LH only at hash value calculation and splitting-control, it is possible to apply concurrency control techniques of LH directly to TH. In LH, we should have 5 kinds of data manipulations, *retrieve, insert, delete, split* and *merge*. Since we can describe *merge* and *delete* by the combination of the basic operations in TH, we put our focus on *insert, split* and *retrieve*.

Generally there arises some conflict against multiple updates to a same key. In both of LH and TH, it is possible to retrieve, insert and split buckets that are currently retrieved without any control. However, a problem arises when we split a bucket to which we are retrieving with some key data, and we need some sophisticated mechanism such as *lock* to block splitting. Similarly we can retrieve, insert and split buckets to which we are now inserting some key data. However, when we apply *undo* data to recover from abort of insertion, we can't retrieve and split the buckets that we are now inserting key data.

When there happens overflow, we should block inserting key data until we complete linking overflow-chain correctly. And finally it is impossible to retrieve, insert and split buckets that we are now splitting.

When some operation terminates abnormally (or an operation *aborts*), the recovery process should be invoked because many update operations make the data space inconsistent. We apply *compensation* process to recovery from abort. By compensation process we can undo all the updates in the aborted process by applying reverse-updates in a reverse order, i.e., "delete a" for "insert a", "update a with b" for "update b with a" and so on. By the process, we can manage recovery without any special control mechanism but we can update buckets directly for recovery independently of other (usual) processes, if there is no intervention among them.

3.2 Lock Types and Compatibility Matrix

In our TH concurrency control we introduce three types of lock control, *read-lock*, *selective-lock* and *exclusive-lock*. Basically retrieval requires read-lock, insertion selective-lock and splitting exclusive-lock for the bucket of interest.

To see how well lock mechanism works, let us illustrate Lock Compatibility matrix in a table 1.

Let us note that, in this investigation, there are 2 levels of lock control, *key-lock* and *page (bucket) lock*. To get a lock on key value, we set a lock on a bucket containing the key first, then we set a lock on any part in the bucket or the overflow area. Once we lock the key, we release page-lock so that multiple processes could insert key data into a same bucket. We get read-lock on a key for retrieval, and exclusive-lock on an empty place in a bucket for inserting key. We get an exclusive-lock on an overflow part even if exclusive-lock or read-lock have already been applied to the overflow. Note that we permit read-lock to retrieval of key data that the process has inserted. ”Split” operation permits neither retrieval nor insertion for a bucket that has been already processed, and we keep holding exclusive-lock for the bucket. Once we commit the process, we release all the locks.

3.3 Re-Hash Scheme

During TH process given a key, we go to some bucket through a hash function with some root variables, but we may see the bucket has different level value because of splitting before, and we should maintain a correct hash value to get to the key value. Then we should go to another bucket from the current bucket position. Here is an algorithm for the re-hash scheme with the level j which the bucket keeps: $if\,(m > j)\,then\,j = j+1; a \leftarrow h_j(C) = C\,mod\,2^j$; As shown in the algorithm, we get some lock on a bucket, but if this is not the one at a, we release the page-lock and lock a correct bucket at a.

3.4 Operation Sequence in a Transaction

Let us discuss the whole configration of TH processes organized into a transaction and how to work. In a figure 2 let us show a typical execution sequence in the environment.

First a user sends a request (a collection of operations which constitute a logical unit of works) to his/her transaction manager(TM). The request corresponds to a *transaction*. After logging the request, TM initiates a set of retrieve or insertion processes according to the request. Each process requires locks to the page and the key at first. TM watches waits for clearing all the lock requests for a while, otherwise there exist some lock conflicts and TM restart all the processes from scratch. Once the processes get several locks on buckets, they starts with inserting or retrieving the key values. TM waits for completion of all the processes. When some retrieve or insert process may abort, the process gets restarted by TM after the compensation process completes the effort. TM generates the log output, commits the work, and tells the result to the user. Note there are two types of *synchronization by TM*, one for clearing all the lock requests and another for completion of all the processes.

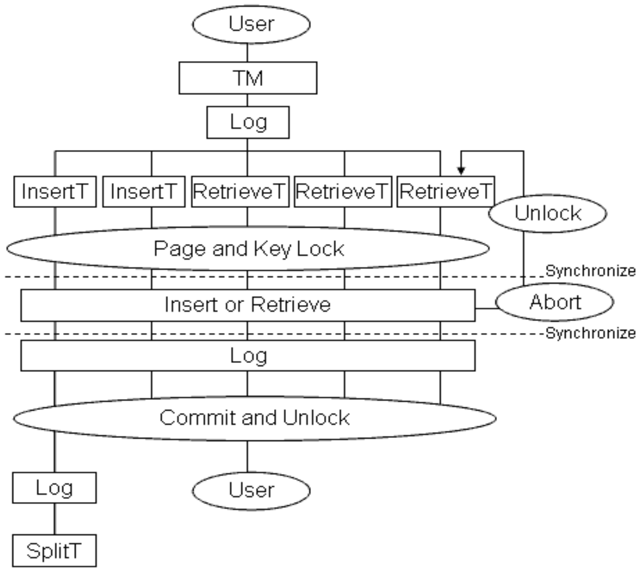


Figure 2. Operation Sequence

We may have the situation to start with splitting buckets when an insertion process terminates. Then TM initiates ”split” process which gets locks on buckets and split. The process generates the log message and commits as usual.

3.5 Deadlock and Compensation Process

When terminating all the processes by force, we may have the situation of *deadlock*. In this work, we detect deadlock by means of *timeout* protocol, i.e., we believe there arises a deadlock after *some* interval and we kill one of the processes waiting for lock-clearing.

The compensation process keeps holding the locks on the key and the bucket during recovery. Thus we can keep the hash space consistent from other access. The compensation process recovers the space by applying reverse operations in a reverse order. After completing the work, the compensation process releases all the locks.

EXAMPLE 2 Let us insert the key ”`Ann`” into the TH space. Once TM puts the log message, an insert process (IT) gets lock on root variables, and obtains hash value of the key. Assume IT gets answer 1 and releases the lock on the variables. Then IT gets selective-lock on a bucket 1. IT applies re-hash scheme to the key because the bucket may be split during IT. Assume IT gets answer 1 again, which means IT still keeps the desired bucket and keeps holding the lock on the key.

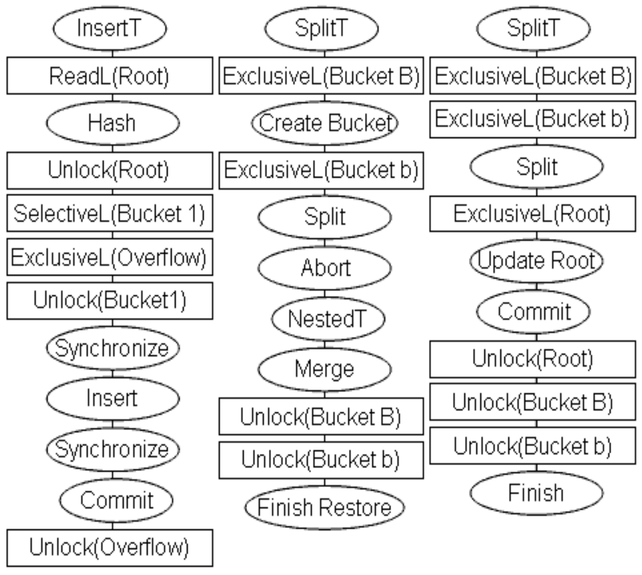


Figure 3. Control of Lock and Unlock

Since there exists an overflow, IT gets exclusive-lock on overflows and does some works. IT waits for the completion of all other processes. After taking synchronization, IT inserts the data and takes synchronization again. TM writes the log message, IT commits the work and unlocks the key.

If we split a bucket during inserting a key ”`Ann`”, ”split” process (ST) writes the log message and gets exclusive-lock on a bucket B, in our case, a bucket 1. ST generates a new bucket b and gets a lock on it, in our case, a bucket 11 for TH. Then we decompose the bucket B into the two buckets. ST waits for releasing the lock on the key.

When the process aborts during splitting a bucket, a compensation process begins. The process reads the log and start recovering. The process examines whether splitting happens at a bucket b, and may combine two buckets for recovery. Finally the process releases the locks.

ST restarts splitting after geting exclusive-locks on the bucket B. ST gets exclusive-lock on root variables and adjusts them. ST writes the log message, commits work and releases the locks on root variables and on the bucket.

We illustrate he whole sequences in a figure 3 containing lock and unlock.

4 Experimental Results

In this section we show some experimental results to see the effectiveness of concurrency control for TH. We also discuss some comparison with LH.

Experiment Number	1	2	3	4	5	6
InsertedRecords	X	25000	25000	25000	25000	25000
Transaction	20	X	20	20	20	20
Multiplicity	200	200	X	200	200	200
MemoryCapacity	30	30	30	30	X	30
AbortProbability	2	2	2	2	2	X

Table 2. Parameters in Experiment

InsertedRecords	LH(I)	TH(I)	LH(R)	TH(R)
10000	3.075	2.702	1.280	1.044
25000	3.138	2.736	1.718	1.414
50000	3.203	2.712	1.524	1.343
100000	3.255	2.618	1.431	1.309

Table 3. InsertedRecords.I/O

In this experiment, we use a collection of 120,000 coordinates records of non-uniformly distribution, which correspond to *postal addresses* area of New York, Boston and Philadelphia[2]. We examine the collection by simulating multiple transaction environment through multi-threading in Java. We evaluate the results by counting bucket I/O per data.

Here we examine 6 experiments described below. In this work, we assume three conditions over bucket splitting[4].

(1) 0.90 as Load factor
(2) $L - 2$ as a splitting level
(3) 0.80 as the ratio of data in a bucket just after splitting

Here we adjust L value only if level values in more than 3 buckets goes beyond L.

There are 5 parameters for our experiments. ”InsertedRecords” means how many data we insert to TH space, ”Transaction” means how many processes a transaction consists of, ”Multiplicity” means the total number of concurrent processes, ”MemoryCapacity” shows teh size of cache memory and ”Abort Probability” means probability assumption how many processes abort. Let us illustrate all the combination of the parameters in a table 2. We examine a value X as a variable in each experiment.

We insert data successively and retrieve them successively afterwards. Let us note that ”(I)” in the table means the insertion case, ”(R)” the retrieval case, ”(A)” the counts of abort and ”(D)” the counts of deadlock.

4.1 Inserting Records

First of all, let us examine how many I/O we need depending on the number of insertion in an experiment 1. We discuss the efficiency with 10000, 25000, 50000 and

[2]`www.rtreeportal.org`

InsertedRecords	LH(I)(A)	TH(I)(A)	LH(R)(A)	TH(R)(A)
10000	2.047	2.010	2.380	1.770
25000	1.987	2.040	2.144	1.996
50000	2.105	2.160	2.098	2.024
100000	2.156	2.233	2.011	2.545

Table 4. InsertedRecords.Abort

Transaction	LH(I)	TH(I)	LH(I)(D)	TH (I)(D)
10	3.249	2.753	1.667	1.333
20	3.138	2.736	1.000	0.667
30	3.120	2.666	5.000	2.333
50	3.124	2.634	21.333	25.667

Table 5. Transaction

100000 data. We show the results in tables 3 and 4 where the value tells the average number of bucket I/O per record.

Clearly the more data we insert, the more the number of I/O we need. Compared to LH, TH shows slightly less I/O but the number increases by about 0.1 to 100,000 from 10,000. Especially all the TH(I) cases preserve about 2.7 while LH cases needs 0.4 additionally. Thus we can say TH keeps efficiency independently of the number of data.

As for retrieval cases, both LH and TH show efficient performance especially in more than 50,000 cases. TH(R) cases are superior to LH, 0.2 less I/O. The main reason is that we need less access to overflow in TH, say 2/3.

Through this experiment we see 2 % as ”abort” probability, the value is almost same to both cases of insertion and retrieval and to the initial probability parameter. This means there happens less process abort caused by deadlock.

4.2 Transactions

In an experiment 2, we examine how TH performs well under many processes given a multiplicity (the total number of processes). Note that one request (a transaction by a user) can be divided into several processes and executed in a concurrent manner. Here we discuss the efficiency under the conditions how many processes we have within a transaction under the condition of the multiplicity 200. Let us show the results in tables 5.

In this experiment we see there is no difference of I/O counts in a ”retrieve” case because of no deadlock. We show only insertion case.

The I/O counts decrease by 0.1 when the number of concurrent processes increases. In our approach TM generate log output and the I/O count decreases when the number of transactions of each request becomes bigger. In a case of the number 10, each process needs 0.2 I/O for logging, but the number of processes is 50, each process requires 0.04 I/O for logging.

In this experiment there arises deadlock very often when the number of processes is more than 30. With bigger numbers of processes in a request, deadlock arises more often since there exist more interfere among processes with each other. However, according to the experiment, we have less I/O caused by by deadlock than I/O by logging.

Multiplicity	LH(I)	TH (I)	LH(I)(D)	TH(I)(D)
200	3.138	2.736	1.000	0.667
400	3.153	2.739	1.667	0.667
600	3.168	2.757	28.667	37.333
800	3.200	2.789	97.333	72.333

Table 6. Multiplicity

4.3 Multiplicity

In experiment 3, we examine TH efficiency with various multiplicity. Remember ”multiplicity” means the total number of processes. If a transaction consists of 20 processes with the multiplicity 200, there can be at most 10 users concurrently. Let us show the results in tables 6.

In this experiment we see there is no difference of I/O count in a ”retrieve” case, and we discuss only insertion case here.

The more transactions there are the more often we see deadlock. This is because we may get locks on same buckets as in other transactions if we have more and more multiplicity. The number of deadlock grows dramatically when multiplicity is more than 600, and the number of deadlock can’t be proportional to multiplicity.

In both TH and LH, the number of I/O increases by 2 % for insertion with multiplicity 800. When a process aborts because of deadlock, we don’t need to delete the dirty data. Thus the number of I/O is 2, one for accessing log and another for accessing a bucket. We need 1.7 I/O for recovery which is theoretical minimum.

4.4 Memory Capacity

In TH processing we assume cache memory management to reduce I/O counts based on Least Recently Used (LRU) policy. Here we examine how about the relationship between memory size and TH efficiency. Let us show the results in tables 7.

Basically the more memory we have the less I/O happens. In our process, we examine whether the bucket is correctly located in memory after getting page-lock. Otherwise we go to the bucket after getting lock on the key. In our experiment 5, we have 90 percent of I/O count with the memory capacity 90 compared to 10.

MemoryCapacity	LH(I)	TH(I)	LH(R)	TH(R)
10	3.238	2.812	1.754	1.418
30	3.138	2.736	1.718	1.414
50	3.063	2.664	1.682	1.307
70	2.972	2.584	1.643	1.330
90	2.883	2.509	1.607	1.296

Table 7. MemoryCapacity

Probability	LH(I)	TH(I)	LH(R)	TH(R)
0	3.078	2.673	1.691	1.344
2	3.138	2.736	1.718	1.380
4	3.215	2.814	1.740	1.409
6	3.289	2.868	1.772	1.435

Table 8. AbortProbability.I/O

4.5 Abort Probability

Finally we examine actual probability of process abort. Let us show the results in tables 8 and 9.

When a TH process aborts, we should erase *dirty* data in a reverse order. Then we have special I/O for log reading, the bucket retrieval and deleting. We need 1.7 I/O for log reading, appropriate number of I/O for retrieval and 1 for deleting. Through our experiment, we need 4.42 I/O for LH and 4.10 for TH in total. If no deletion is necessary, we need 1.7 I/O. Average I/O for recovery becomes 3.06 for LH and 2.90 for TH when 50 percent of aborts is no deletion case.

As a result, we need 3.40 I/O in LH and 3.03 I/O in TH for recovery. However, there exists relatively large variation of 0.2 because of deadlock.

We see the abort probability given as a parameter and the actual probability are almost the same. Looking at deadlock 0-4, we see abort has no effect on deadlock.

4.6 Discussions

By our experiments, we can say that LH and TH work in a similar manner.

To reduce I/O count, we must have (1) small number of data, (2) large memory, (3) lower multiplicity, and (4) large number of concurrent processes within a transaction. In TH, the condition (1) is straightforward. We must have more deadlock without (2). However, we have both less I/O caused by large memory capacity and more I/O count by deadlock, and by our experiment, we can say we'd better have enough memory.

The conditions to avoid deadlock are (1) lower multiplicity and (2) large number of concurrent processes within a transaction. When the multiplicity exceeds the thresholds at the conditions (1) and (2), deadlock arises more often and we should give the appropriate threshold values.

Probability	LH(I)(A)	TH(I)(A)	LH(R)(A)	TH(R)(A)
0	0.152	0.027	0.000	0.000
2	1.987	2.040	2.144	1.996
4	4.047	4.272	4.044	4.228
6	6.167	6.480	6.520	6.316

Table 9. AbortProbability.Abort

TH is not really different from LH from the viewpoint of abort probability and deadlock. However, we have less I/O for recovery for abort and as well as all the aspects pointed in the experiemnts, TH is superior to LH. Moreover, the efficiency in TH is independent of data size, and is more useful under concurrency control.

Remember that we can manage splitting conditions arbitrarily, and that dynamically we can adjust efficiency caused by splitting under heavy traffic situation.

5 Conclusion

In this investigation, we have examined the concurrency aspects for Tree Hash. By several experimental results, we have shown the superior properties. It is possible to manage TH space efficiently without excessive deadlocks by giving parameters in a suitable manner, and we can recover the space with less I/O independently of other processes by means of compensation approach. We can adjust efficiency caused by splitting under heavy traffic situation.

Let us point out that there can be TH under the distributed environment[4]. There remains how to do that.

References

[1] Litwin, W.: "LINEAR HASHING : A NEW TOOL FOR FILE AND TABLE ADDRESSING", In *Proc. of VLDB*, 1980.

[2] Ellis, C. S.: "CONCURRENCY IN LINEAR HASHING", In *ACM Transactions on Database Systems (TODS)*, 1987

[3] Madria, S.K., Tubaishat, M.A.: "AN OVERVIEW OF SEMANTIC CONCURRENCY CONTROL IN LINEAR HASH STRUCTURES", In *Intn'l Symposium on Computer and Information Systems (ISCIS98)*, 1998

[4] Yasuda,K. Miura, T. and Shioya, I.: "DISTRIBUTED PROCESSES ON TREE HASH", In *Computer Software and Applications Conference (COMPSAC '06)*, 2006

Query Processing in Paraconsistent Databases in the Presence of Integrity Constraints

Navin Viswanath Rajshekhar Sunderraman
Georgia State University, Atlanta, GA 30302
E-mail: nviswanath1@student.gsu.edu,raj@cs.gsu.edu

Abstract

In this paper, we present an approach to query processing for paraconsistent databases in the presence of integrity constraints. Paraconsistent databases are capable of representing positive as well as negative facts and typically operate under the open world assumption. It is easily observed that integrity constraints are usually statements about negative facts and as a result paraconsistent databases are suitable as a representation mechanism for such information. We use set-valued attributes to code large number of regular tuples into one extended tuple (with set-valued components). We define an extended relational model and algebra capable of representing and querying paraconsistent databases in the presence of integrity constraints. The extended algebra is used as the basis for query processing in such databases.

1 Introduction

The relational data model introduced by Codd ([6]) has been a major success story in computer science and is the basis for the database systems widely in use. Relations allow only atomic values in the domains of attributes (First Normal Form) and negative information is inferred using the Closed World Assumption (CWA). These assumptions are quite reasonable in traditional applications, but are questionable in newer application domains such as scientific and life science data management. Scientists usually do not prefer to use default negation rules to infer negative data and rely on the open world assumption. Several researchers ([5, 7, 9, 12]) have looked into the problem of representing and manipulating negative data explicitly in databases. Such databases have been referred to as paraconsistent databases as they are based on 4-valued paraconsistent logic ([4]). In [2], Bagai and Sunderraman introduced paraconsistent relations, a framework for relational databases to represent and query explicit negative facts. Paraconsistent relations represent both positive as well as negative facts explicitly and hence employ the open world assumption for negative data. Paraconsistent relations and their associated algebra were used in [1] to compute the well-founded semantics of logic programs. In this paper, we propose to further extend the notion of paraconsistent relations to include set-valued attributes. We show that this extension is suitable for query processing in the presence of integrity constraints.

Relations that contain relations as tuple components are called non-first normal form relations [8, 10, 11]. In this paper, we restrict our attention to relations that allow only sets as tuple components and thus is a special case of non-first normal form relations [8, 10]. If the domain of an attribute is a subset of the powerset of the atomic-valued domain, we call it a set-valued attribute. The extended relational model requires that the domain of attributes be set-valued. Atomic values are represented as singleton sets. Every row in such an extended relation shall henceforth be called an **s-tuple** to differentiate it from the term **tuple** that we normally associate with regular relations and the relations will themselves be called **s-relations**. Since tuple components in s-relations are set-valued, we extend the notation to allow the "complement" operation from set theory. It will be shown that this notation, apart from increasing the clarity and simplicity of representation, also increases the power of the algebra, specifically when applied to paraconsistent relations. Thus tuple components can also contain $\overline{\{a\}}$ which represents a set containing all elements in the domain of the attribute except a, and ϕ, the empty set. Null values in s-relations are thus represented as ϕ. $\bar{\phi}$ will now represent the entire domain. We will also make the assumption that all attributes have the same domain $\bar{\phi}$, without any loss of generality.

Consider `student(ssn,name,phone)`, a simple predicate, which describes students. In our set-valued paraconsistent relational model, we will be able to express facts such as "Student John has ssn 1234 and has two phones 1111 and 1112" using the s-tuple notation $< 1234, John, \{1111, 1112\} >$ to denote a positive fact. The functional dependency constraint $ssn \rightarrow name$ would allow us to infer negative facts in the form of the s-tuple

$< 1234, \overline{\{John\}}, \bar{\phi} >$.

The rest of the paper is organized as follows: Section 2 presents the set-valued extension to the relational model. Section 3 presents the set-valued extension to the paraconsistent relational model. Section 4 presents query processing approaches for paraconsistent data in the presence of integrity constraints. Section 5 concludes with a discussion on future work.

2 A Set-Valued Extension to the Relational Model

Let a relation scheme Σ be a finite set of attribute names, where for any attribute name $A \in \Sigma$, $dom(A)$ is a non-empty domain of values for A. A tuple on Σ is any map $t : \Sigma \rightarrow \cup_{A \in \Sigma} dom(A)$, such that $t(A) \in dom(A)$ for each $A \in \Sigma$. Let $t(\Sigma)$ denote the set of all tuples on Σ [2].
An ordinary relation on scheme Σ is thus any subset of $t(\Sigma)$.
Now we define the notion of s-tuples and s-relations:
s-tuple.An s-tuple on Σ is any map $t : \Sigma \rightarrow \cup_{A \in \Sigma} dom(A)$, such that $t(A) \subseteq dom(A)$ for each $A \in \Sigma$. Let $\tau(\Sigma)$ denote the set of all s-tuples on Σ. Then, $\tau(\Sigma) =< \bar{\phi}, \bar{\phi}, \ldots, \bar{\phi} >$.
s-relation. An s-relation on scheme Σ is a set of s-tuples on Σ.
Figure 1 shows an s-relation in which each s-tuple has set-valued components. The ϕ in the last s-tuple indicates that there is no value (NULL) under the column Ph for that s-tuple.

STUDENT		
SSN	Name	Ph
{111}	{Tom}	{4046514633, 4046654321}
{888}	{Jennifer}	ϕ

Figure 1. An s-relation, STUDENT

Given an s-relation e which contains k s-tuples where the i^{th} tuple is denoted by $< t_{i1}, \ldots, t_{in} >$ where all $t_{ij}, 1 \leq i \leq k, 1 \leq j \leq n$, are set-valued, there is a one-to-one correspondence between the s-relation e and the ordinary relation corresponding to e. We denote by e_{ord} the ordinary relation corresponding to the s-relation e defined as follows:

$$e_{ord} = \cup_{i=1}^{k} t_{i1} \times \ldots \times t_{in}$$

We introduce two new operators **REDUCE** and **COMPACT**, which will be used in all operations in the model.

2.1 REDUCE

We define an operator, $REDUCE$, which takes an s-relation e on scheme Σ as input and returns another s-relation on scheme Σ after eliminating redundant s-tuples. Below is a formal description of $REDUCE$.

$$REDUCE(e) = \{t_1 \in e | \neg(\exists(t_2 \in e)) \wedge \forall_{x \in \Sigma}(t_2[x] \supset t_1[x])\}$$

e		
A	B	C
{1,2,3}	$\bar{\phi}$	$\overline{\{1\}}$
{3}	{6,7}	$\overline{\{1,3\}}$
{6}	{8}	{7,9}

$REDUCE(e)$		
A	B	C
{1,2,3}	$\bar{\phi}$	$\overline{\{1\}}$
{6}	{8}	{7,9}

Figure 2. Example of $REDUCE$

2.2 COMPACT

We introduce a new operator $COMPACT$ that takes an s-relation e as input, and produces another s-relation e'.The new s-relation e' will have atmost the number of s-tuples in e or fewer. The algorithm below will take as input an s-relation e and the associated functional dependencies and produce another s-relation e' as output.

```
Algorithm $COMPACT(e, R, F)$
Input:An s-relation $e$ under the scheme $R$ and a set of associated functional dependencies $F$.
Output:A compacted s-relation $e'$.
Method:The s-relation $e'$ is obtained as follows:
1. Let $C = \{k_1, k_2, .., k_n\}$ be the candidate keys computed from $F$.
2. if($\exists i, j | (k_i \cap k_j = \phi) \vee ((|C| = 1) \wedge (|k_1| \neq 1)))$
   return $e$
   else{
      if($\exists k_c \in C | (|k_c| = 1) \vee (\forall_{i,j}(k_i \cap k_j = k_c) \wedge$
         $(|k_c| = 1))$
         for every set of tuples $T$ for which $\pi_{<R-k_c>}(T)$
         is singleton,
         replace them with a new s-tuple $t$ such that
         $t[R - k_c] = \pi_{<R-k_c>}(T)$ and $t[k_c] = \cup \pi_{k_c}(T)$
         return the new compacted relation $e'$
      }
```

The intuition behind the $COMPACT$ algorithm is that *the column picked to be set-valued would be one that belongs to all keys of the s-relation*. When the keys of the s-relation are computed, a number of scenarios can occur:
Case 1: The s-relation has only one key attribute.
Case 2: The s-relation has only one key but the key consists of more than one attribute.
Case 3: The s-relation has multiple keys and all of them

have exactly one attribute in common.
Case 4: The s-relation has multiple keys and NOT all of them have exactly one attribute in common.
Let us consider the operation of $COMPACT$ on the s-relation e below. Let A be the only key attribute of the s-relation. In both cases 2 and 4 the algorithm just

e		
A	B	C
{1}	{6,7}	{9}
{2}	{3}	{8}
{3}	{6,7}	{9}

$COMPACT(e)$		
A	B	C
{1,3}	{6,7}	{9}
{2}	{3}	{8}

Figure 3. An example of $COMPACT$

returns the original s-relation and does not attempt to $COMPACT$ the s-relation e any further. This is because the time complexity of performing such an operation becomes exponential when there is more than one attribute to $COMPACT$ on.
Let us now examine cases 1 and 3 which ARE candidates for $COMPACT$. The simplest is Case 1 where the s-relation has just one key attribute. Since that attribute is the ONLY key of the s-relation, it very likely that we will have two or more tuples that have identical values under all other attributes (it should be noted here that this would not have been possible if there WAS another key for the s-relation). These tuples can then be combined into a single s-tuple with the key attribute set-valued.
Case 3 is where there are multiple keys in the s-relation and all have exactly one attribute in common. Since we are looking for exactly one attribute to perform $COMPACT$, an attribute that appears in all keys of the s-relation seems an ideal candidate going by the intuition that $COMPACT$ was based on.

2.3 Algebraic Operators

Here we define the algebraic operators for s-relations. We also define an operator $REP^{\{\}}$ which takes an ordinary relation under any scheme Σ as input and produces an s-relation under the same scheme as follows:
$REP^{\{\}}(R) = \{s|(\forall t \in R)(\forall A \in \Sigma)s[A] = \{t[A]\}\}$
This operator $REP^{\{\}}$ is used in the definition of the difference operator.

Set theoretic operators

Union. The union of s-relations e_1 and e_2, under scheme Σ and with functional dependencies F_1 and F_2 repectively, denoted by $\cup^s$, is defined as follows:
$e_1 \cup^s e_2 = COMPACT(REDUCE(\{t|t \in e_1 \text{ or } t \in e_2\}), \Sigma, F_1 \cup F_2)$

Difference. The difference between two s-relations, e_1 and e_2, under scheme Σ and with functional dependencies F_1 and F_2 repectively, denoted by $-^s$, is defined as follows:
$e_1 -^s e_2 = COMPACT(REDUCE(REP^{\{\}}(e_{1ord}) \cup e_2) - e_2, \Sigma, F_1)$
Intersection. We use the identity $e_1 \cap^s e_2 = e_1 -^s (e_1 -^s e_2)$ with the algorithm above to compute $\cap^s$ on s-relations e_1 and e_2.

Relation theoretic operators

Selection. The selection of e by F, where e is an s-relation on scheme Σ, denoted by $\sigma^s_F(e)$ where e is an s-relation on scheme Σ as follows:
Let $\theta = \{<, \leq, =, >, \geq, \neq\}$. Let c, c_1, c_2 be constants and $X, Y \in \Sigma$ The formula F can be classified into one of four cases:
Case 1: $c_1 \theta c_2$
Case 2: $X \theta c$
Case 3: $c \theta Y$
Case 4: $X \theta Y$.
Case 1 is trivial and returns either TRUE or FALSE and hence the query returns either e or the empty relation.
Case 2: Without loss of generality, assume X is the first column in the s-relation e. Let $t =< u_1, u_2 \ldots u_n >$ be any s-tuple in e and let $u_1 = \{a_1, a_2 \ldots a_m\}$. Then, $u_1' = \{a_i | 1 \leq i \leq m \text{ and } a_i \theta c \text{ is true }\}$. If $u_i\prime = \phi$, then drop t. Else return $t' =< u_1', u_2 \ldots u_n >$
Case 3 is similar to Case 2.
Case 4: Without loss of generality, let X be the first column and Y be the 2^{nd} column in s-relation e. Let $t =< u_1, u_2 \ldots u_n >$ be any s-tuple in e and let $u_1 = \{a_1, a_2 \ldots a_m\}$ and $u_2 = \{b_1, b_2 \ldots b_m\}$. Let $c = b_1$ and repeat Case 2 to generate t_1'. Let $c = b_2$, $c = b_3$ and so on to generate a new s-tuple t_i' for each b_i. Thus atmost n new s-tuples are generated for each $b_i, 1 \leq i \leq n$. This can be reduced to $min(m, n)$ s-tuples by choosing $Y \theta' X$ instead of $X \theta Y$ where θ' is the complementary operation to θ.
Projection. The projection of e onto Δ, denoted by $\pi^s_\Delta(e)$ where e is an s-relation on scheme Σ, and $\Delta \subseteq \Sigma$ and F is the set of functional dependencies, is defined as follows:
$\pi^s_\Delta = COMPACT(REDUCE(\{t[\Delta]|t \in e\}), \Delta, F)$.
Cartesian product. Let e_1 and e_2 be two s-relations on schemes Σ and Δ respectively. Then, the cartesian product, denoted by $e_1 \times^s e_2$ is an s-relation on scheme $\Sigma \circ \Delta$ defined as
$e_1 \times^s e_2 = REDUCE(\{t_1 \circ t_2 | t_1 \in e_1 \text{ and } t_2 \in e_2\})$
where $\circ$ denotes the concatenation operation.

3 Set-Valued Paraconsistent Relations

Unlike normal relations where we only retain information that is believed to be true of a particular predicate, the paraconsistent relational model is a step towards completing the database. In a paraconsistent relation, we also retain what is *believed to be false* of a particular predicate [2, 3]. We define paraconsistent relations formally as follows:
Paraconsistent relations. A paraconsistent relation on a scheme Σ is a pair $< R^+, R^- >$ where R^+ and R^- are ordinary relations on Σ.
Thus R^+ represents the set of tuples believed to be true of R and R^- represents the set of tuples believed to be false. We allow the paraconsistent relations to be set-valued and introduce the notion of **sp-relations**.
sp-relation. An sp-relation on a scheme Σ is a pair $< R^+, R^- >$ where R^+ and R^- are s-relations on Σ. Also,
$COMPACT(R) =< COMPACT(R^+), COMPACT(R^-) >$
$REDUCE(R) =< REDUCE(R^+), REDUCE(R^-) >$
Figure 4 is a instance of a paraconsistent employee database. We use this as a running example for all the queries in the paper. The database has a relation

$Employee$		
SSN	Name	Age
{111}	{Navin}	{24}
{222}	{James}	{23}
{333}	{Jennifer}	{25}
{555}	$\bar{\phi}$	$\bar{\phi}$
{666}	$\bar{\phi}$	$\bar{\phi}$

$Supervisor$	
SSN	SuperSSN
{111}	{333}
{222}	{111}
{111}	$\overline{\{333\}}$
{333}	$\bar{\phi}$

Figure 4. A set-valued paraconsistent employee database

$Employee =< Employee^+, Employee^- >$ which represents the employee entity and the relation $Supervisor =< Supervisor^+, Supervisor^- >$ which represents their supervisors(who are themselves employees). The tuples $< \{555\}, \bar{\phi}, \bar{\phi} >$ and $< \{666\}, \bar{\phi}, \bar{\phi} >$ in $Employee^-$ indicate that there are no employees with SSN='555' or SSN='666'.

3.1 Algebraic Operators

Here we define the algebraic operators for sp-relations.

Set theoretic operators

Let R and S be two sp-relations on scheme Σ.
Union. The union of R and S, denoted $R \cup^{sp} S$, is an sp-relation on scheme Σ, given by
$(R \cup^{sp} S)^+ = R^+ \cup^s S^+$, $(R \cup^{sp} S)^- = R^- \cap^s S^-$,
where $\cup^s$ denotes union over s-relations and $\cap^s$ denotes intersection over s-relations.
Complement. The complement of sp-relation R, denoted by $-^{sp}R$ is an sp-relation on scheme Σ,given by
$(-^{sp}R)^+ = R^-$, $(-^{sp}R)^- = R^+$
Intersection. The intersection of sp-relations R and S, denoted by $R \cap^{sp} S$, is an sp-relation on scheme Σ, given by,
$(R \cap^{sp} S)^+ = R^+ \cap^s S^+$, $(R \cap^{sp} S)^- = R^- \cup^s S^-$
Difference. The difference of sp-relations R and S, denoted by $R -^{sp} S$, is an sp-relation on scheme Σ, given by
$(R -^{sp} S)^+ = R^+ \cap^s S^-$, $(R -^{sp} S)^- = R^- \cup^s S^+$

Relation theoretic operators

Selection. Let R be an s-relation under scheme Σ and let F be a formula of the form $X\theta Y$ where $\theta = \{<, >, =, <=, >=, \neq\}$. Then, the selection of R by F, denoted by $\sigma_F^{sp}(R)$ is a sp-relation on Σ, given by $\sigma_F^{sp}(R)^+ = \sigma_F^s(R^+)$, $\sigma_F^{sp}(R)^- = R^- \cup^s \sigma_{\neg F}^s(\tau(\Sigma))$
where σ^s is the selection operation on s-relations.
The negative component, $R^- \cup \sigma_{\neg F}^s(\tau(\Sigma))$, is computed as follows. Since $\tau(\Sigma)$ represents the set of all tuples on Σ, it can be represented as the single $|\Sigma|$-tuple $< \bar{\phi}, \bar{\phi}, ..., \bar{\phi} >$. Selecting s-tuples that satisfy $\neg F$ from $\tau(\Sigma)$ will thus mean removing from each component $\bar{\phi}$ in $\tau(\Sigma)$, those values that satisfy $\neg\neg F$, or F. Notice that when F is of the form $X\theta Y$, and either X or Y is a constant, $\sigma_{\neg F}^s(\tau(\Sigma))$ will always contain only one s-tuple.
Projection. Let R be an sp-relation on scheme Σ and let $\Delta \subseteq \Sigma$. Then, the projection of R onto Δ, denoted by $\pi_\Delta^{sp}(R)$, is an sp-relation on Δ, given by,
$\pi_\Delta^{sp}(R)^+ = \pi_\Delta^s(R^+)$, $\pi_\Delta^{sp}(R)^- = \{t \in \tau(\Delta) | t^\Sigma \subseteq (R^-)^\Sigma\}$,
where π_Δ^s is the projection over Δ of s-relations.
The negative component of the projection denotes the set of all tuples in scheme Δ, $\tau(\Delta)$ such that all their extensions are present in $(R^-)^\Sigma$.
We define extensions of an s-tuple as follows:
If Σ and Δ are relation schemes such that $\Delta \subseteq \Sigma$, then for any s-tuple $t \in \tau(\Delta)$, we let t^Σ denote the set of $|\Sigma|$ -tuples $\{t' | t'(A) = t(A)$, for all $A \in \Delta$ and $t'(B) = \bar{\phi}$, for all $B \in \Sigma - \Delta\}$.
Join. Let R and S be sp-relations on schemes Σ and Δ respectively. Then the natural join of R and S, denoted by $R \bowtie^{sp} S$, is given by,
$(R \bowtie^{sp} S)^+ = R^+ \bowtie^s S^+$, $(R \bowtie^{sp} S)^- = (R^-)^{\Sigma\cup\Delta} \cup^s (S^-)^{\Sigma\cup\Delta}$
where $\bowtie^s$ can be defined in terms of $\times^s$ and σ^s.

4 Query Processing in the Presence of Constraints

An important application of sp-relations is the treatment of constraints in relations. In this paper, we restrict ourselves to two commmon constraints found in relational databases - functional dependencies and referential integrity constraints. Generally the number of tuples that do not belong in a relation will be much larger than then number of tuples in it. Hence in a paraconsistent relation $R =< R^+, R^- >$, set-valued attributes might be very useful while expressing the s-relation R^-. Sections 4.1 discusses a method of representing constraints in sp-relations.

4.1 Representing Constraints in sp-relations

A functional dependency of the form $A \rightarrow B$ in a relation R introduces a constraint that for any two tuples $t_1, t_2 \in R$, if $t_1[A] = t_2[A]$ then $t_1[B] = t_2[B]$. This results in an explosion in the information content when the relation is paraconsistent. Whenever a functional dependency is present in a relation, the constraint thus introduced implies that we can infer a number of facts to be false in R, or, in other words, we can conclude that those facts will belong to R^-. Let $R =< R^+, R^- >$ be a paraconsistent relation under the scheme Σ. Assume that there is a tuple $t \in R^+$ with $t[A] = a$ and $t[B] = b$ and a functional dependency $A \rightarrow B$ for some attributes $A, B \in \Sigma$. This implies that any tuple with $t[A] = a$ and $t[B] \neq b$ will be in R^-. Thus R^- will contain tuples of the form
$\{t \mid (t[A] = a) \wedge (t[B] = x) \wedge (x \neq b) \wedge$
$(t[\Sigma - \{A, B\}] \in \tau(\Sigma - \{A, B\}))\}$
for every functional dependency $A \rightarrow B$.
With sp-relations, it is much easier to represent the functional dependencies in the negative component. The notations that were introduced in Section 1 now simplify the process and it involves introducing just *one s-tuple of the form* $< a, \overline{\{b\}}, \bar{\phi}, \bar{\phi}... >$ *in* R^-.
Similarly, a referential integrity constraint on a database requires that each value in the foreign key in a relation matches the value in the primary key. In paraconsistent relations, when a value is stored as false in the primary key of a relation i.e. in the negative component in all possible combinations, then all foreign keys matching that primary key value will also become false. For example, in the employee database, the employee relation with primary key SSN has values '555' and '666' stored in the negative component in all possible combinations. This implies that no employee exists with either SSN '555' or '666'. The supervisor relation has SSN as a foreign key. Since SSN values '555' and '666' are false in the employee relation, all extensions of these values can be introduced in the negative component of the supervisor relations as $< 555, \bar{\phi} >$ and $< 666, \bar{\phi} >$.

4.2 Query Example

The database instance of Figure 4 modified to include the FD $SSN \rightarrow Name, Age$ and the attribute references SSN and $SuperSSN$ in $Supervisor$ to SSN in $Employee$ is shown in Figure 5.

$Employee$		
SSN	Name	Age
{111}	{Navin}	{24}
{222}	{James}	{23}
{333}	{Jennifer}	{25}
{555}	$\bar{\phi}$	$\bar{\phi}$
{666}	$\bar{\phi}$	$\bar{\phi}$
{111}	$\overline{\{Navin\}}$	$\overline{\{24\}}$
{222}	$\overline{\{James\}}$	$\overline{\{23\}}$
{333}	$\overline{\{Jennifer\}}$	$\overline{\{25\}}$

$Supervisor$	
SSN	SuperSSN
{111}	{333}
{222}	{111}
{111}	$\overline{\{333\}}$
{333,555,666}	$\bar{\phi}$
$\bar{\phi}$	{555,666}

Figure 5. The employee database instance after coding constraints

Consider the query: *Find the SSN's of all employees not supervised by the employee with SSN=333.*
The query can be expressed as shown below in terms of the algebra:
$-^{sp}(\pi^{sp}_{<SSN>}(\sigma^{sp}_{<SuperSSN=333>}(Supervisor)))$
Let $R = \sigma^{sp}_{<SuperSSN=333>}(Supervisor)$. Then the answer to the query is obtained by the expression $-^{sp}(\pi^{sp}_{<SSN>}(R))$. The answer to the query seems to be intuitively correct since we have information that the employee with SSN='111' is supervised by the employee with SSN='333' and no one else. Hence $\{111\}$ appears in the negative component of the answer. Similarly, we do not have complete information about the supervisors of employee with SSN=$\{222\}$ and hence this value does not

R	
SSN	SuperSSN
{111}	{333}
{111}	$\overline{\{333\}}$
{333,555,666}	$\bar{\phi}$
$\bar{\phi}$	{555,666}
$\bar{\phi}$	$\overline{\{333\}}$

$\pi^{sp}_{<SSN>}(R)$
SSN
{111}
{333,555,666}

$-^{sp}(\pi^{sp}_{<SSN>}(R))$
SSN
{333,555,666}
{111}

Figure 6. Answer to query

appear in the answer. SSN values $\{555{,}666\}$ are not employees at all and hence they appear in the positive component of the answer trivially. Notice that this query was formulated in a manner different from what would have been done in ordinary relational databases. There we would have used a difference operator instead of the complement used here. The answer to this query if the expression was written with the difference operator would have been $< \{333\} >$ in the positive component and $< \{111, 555, 666\} >$ in the negative component which would indicate that employees with SSN $\{555{,}666\}$ **are** supervised by the employee with SSN='333'. This is intuitively incorrect since the database indicates that they are not employees at all. Thus the complement operator defined in this model allows us to express queries with more precision than can be achieved with the minus operator in the relational model.

5 Conclusions and Future Work

In this paper, we present a relational model that is suited to query processing in the presence of constraints. The constraints imposed on a database result in the inference of negative facts. The paraconsistent relational model is an elegant way to process queries in the presence of constraints since negative facts are represented explicitly as tuples in this model. One of the drawbacks of the paraconsistent model was that the negative facts in a database, especially those introduced by constraints such as functional dependencies and referential integrity constraints, are very large and representing them as tuples is infeasible. This motivated the introduction of a set-valued relational model. The set-valued approach defined here is markedly different from earlier approaches in that we extend it to allow notations such as the complement and the empty set. The empty set notation ϕ seems to be particularly relevant here since it is consistent with the other domain values in the database as opposed to the NULL in ordinary relations, which is not very intuitive.

The paper also addresses two other issues introduced by set-valued notations. One is the issue of redundancy, since redundant information is not easy to eliminate when the tuples are set-valued. The $REDUCE$ operator addresses this issue and reduces redundancy. The other is to exploit the set-valued model in order to reduce the number of tuples, especially in the negative components of relations. The $COMPACT$ operator was defined to 'shrink' the table using the keys in the relation. The sp-relational model provides an elegant way for processing queries in the presence of constraints by explicitly representing this information in the database.

An interesting topic for future work is the analysis of the operators defined in the sp-relational model. This becomes particularly important since the algebra of this model allows for different expressions of queries and brings out issues of semantics when we go from expressing queries in the natural language to the algebra. We have highlighted this in the last section with an example query.

References

[1] Rajiv Bagai and Rajshekhar Sunderraman. An algebraic construction of the well-founded model. In Vangalur S. Alagar and Maurice Nivat, editors, *AMAST*, volume 936 of *Lecture Notes in Computer Science*, pages 518–530. Springer, 1995.

[2] Rajiv Bagai and Rajshekhar Sunderraman. A paraconsistent relational data model. *International Journal of Computer Mathematics*, 55(3), 1995.

[3] Rajiv Bagai and Rajshekhar Sunderraman. Bottom-up computation of the fitting model for general deductive databases. *Journal of Intelligent Information Systems*, 6(1):59–75, January 1996.

[4] N. D. Belnap. A useful four-valued logic. In G. Eppstein and J. M. Dunn, editors, *Modern Uses of Many-valued Logic*, pages 8–37. Reidel, Dordrecht, 1977.

[5] H. A. Blair and V. S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computer Science*, 68:135–154, 1989.

[6] E.F. Codd. A relational model for large shared data banks. *Comm. of the ACM*, 13(6):377–387, 1970.

[7] Hendrik Decker. A case for paraconsistent logic as foundation of future information systems. In Jaelson Castro and Ernest Teniente, editors, *CAiSE Workshops (2)*, pages 451–461. FEUP Edições, Porto, 2005.

[8] Jaeschke G. and Schek H. Remark on the algebra of non first normal form relation. In *Proceedings of ACM Symposium on Principles of Database Systems*, 1982.

[9] John Grant and V.S. Subrahmanian. Applications of paraconsistency in data and knowledge bases. *Synthese*, 125:121–132, 2000.

[10] G. Ozsoyoglu, Z. M. Ozsoyoglu, and V. Matos. Extending relational algebra and relational calculus with set-valued attributes and aggregate functions. *ACM Trans. on Database Syst.*, 12(4):566–592, 1987.

[11] Mark A. Roth, Henry F. Korth, and Abraham Silberschatz. Extended algebra and calculus for nested relational databases. *ACM Trans. Database Syst.*, 13(4):389–417, 1988.

[12] V. S. Subrahmanian. Paraconsistent disjunctive deductive databases. *Theoretical Computer Science*, 93:115–141, 1992.

An Object-Oriented Approach to Storage and Retrieval of RDF/XML Documents

Ching-Ming Chao
Department of Computer and Information Science, Soochow University

***Abstract*-The Resource Description Framework (RDF) is a foundation for processing metadata, which provides interoperability between applications exchanging machine-understandable information on the World Wide Web. Due to its highly flexible hierarchical structure and machine-independent characteristic, XML has become the formal way to store and transmit RDF models. We refer such documents as RDF/XML documents, or RDF documents for short. Efficient storage and retrieval of RDF documents in persistent data stores is an important issue in computer technology today. In this paper, therefore, we propose an object-oriented approach to this issue. Firstly, we propose an object-oriented data model, called the RDF Data Storage Model (RDSM), for storage of data extracted from RDF documents, as well as an RDF document decomposition algorithm for extraction of data from RDF documents. Secondly, we propose a generic RDF API that supports fundamental RDF data accessing and querying operations, and utilize the W3C's SPARQL language as the high-level query language for retrieval of RDF data. Finally, an experimental system is implemented to demonstrate the efficiency and effectiveness of the proposed approach.**

I. Introduction

The Resource Description Framework (RDF) is a W3C Recommendation for the notation of metadata on the World Wide Web (WWW) [1,2]. The RDF Schema (RDFS) extends this standard by providing developers with the means to specify vocabularies and model object structures [3]. These techniques enable enrichment of the Web with machine-understandable semantics, thus helping to construct the Semantic Web described by Tim Berners-Lee *et al.* in [4].

It should to be noticed that RDF is a model for expressing metadata, which implies that RDF users can use any possible syntax to encode data described in RDF. Due to its highly flexible hierarchical structure and machine-independent characteristic, XML has become the formal way to store and transmit RDF models. We refer such documents as RDF/XML documents, or RDF documents for short. How to efficiently store and retrieve RDF documents in persistent data stores is an important issue in computer technology today.

In this paper, therefore, we propose an object-oriented approach to this issue. Firstly, we propose an object-oriented data model for storage of data extracted from RDF documents, as well as an RDF document decomposition algorithm for extraction of data from RDF documents. Secondly, we propose a generic RDF API that supports fundamental RDF data accessing and querying operations, and utilize the emerging W3C's SPARQL language as the high-level query language for retrieval of RDF data. Finally, we implement an experimental system to demonstrate the efficiency and effectiveness of the proposed approach.

The rest of this paper is organized as follows. Section II reviews related work. Section III presents the data model and the decomposition algorithm. Section IV describes the generic RDF API and the query-answering algorithm. Section V illustrates the results of performance evaluation. Section VI concludes this paper and suggests directions for future research.

II. Related Work

Several methods for storing XML documents in relational databases have been proposed. These methods can be roughly classified into two categories: the structure-mapping approach and the model-mapping approach [5]. In the former approach, a database schema represents the logical structure of the target XML document, and the logical structure can be obtained by analyzing the Document Type Definition (DTD) or the XML Schema Definition (XSD) accompanying the document. In other words, a set of tables is defined for every single XML document structure or, more precisely, for the DTD or XSD of the XML document. Examples of this approach can be found in [6,7]. In the latter approach, a set of consolidated tables is created to store all XML documents. Examples of this approach can be found in [5,8]. The strength of the structure-mapping approach lies in that the precise information represented in the DTD or XSD can be preserved. Since this approach creates a set of tables for every DTD or XSD, however, management of these tables raises a challenging issue to the database administrator (DBA). On the other hand, the model-mapping approach only maintains one set of consolidated tables and therefore greatly reduces the burden of DBA.

Besides classification according to the mapping approach, these methods can also be distinguished into using schema information or using no schema information. By using schema information such as DTD or XSD, an XML document can be decomposed without losing its precise structure and data type information. By using no schema information, on the other hand, it is possible to lose the precise structure and data type information of a tree-like semistructured data set (to which an XML document conforms). Examples of the former case can be found in [5,6,7], and examples of the latter case can be found in [8].

The database management system (DBMS) chosen to store incoming XML documents is also an important factor affecting the development of these methods. Traditionally, most of the methods adopt the relational or object-relational DBMS (RDBMS/ORDBMS) as the underlying DBMS. Relational technology and its successor object-relational technology, however, have proven awkward for queries that require complex XML constructs in their results, and may be inefficient when fragmentation due to handling of set-valued attributes and sharing causes too many joins in the evaluation of simple queries [7].

One notable method proposed by Shanmugasundaram *et al.* can be found in [7]. In this method, a DTD is simplified with a series of transformation, and the related DTD graph is then created on the basis of the simplified DTD in order to generate relational schemas with the Shared or Hybrid techniques they proposed. Another notable method proposed by Florescu and Kossmann can be found in [8]. In this method, several approaches are proposed (Edge, Binary, Universal Table, Separate Value Tables, and Inlining) by mapping an XML instance to an ordered and labeled directed graph and storing the structure of the graph in several fixed-schema tables according to the scheme they choose.

III. Storing RDF/XML Documents

A. Data Model for Storing RDF Data

An RDF model usually contains several statements, each of which consists of a resource as the subject, a resource as the predicate and a resource/literal as the object. Furthermore, both of the resource and literal can be treated as a kind of generalized node–a node containing the parts and operations that all of the resources or literals have in common–so that they can be presented as a class hierarchy forming the data model used to store these nodes systematically. We adopt the model-mapping approach and propose an object-oriented data model, called the RDF Data Storage Model (RDSM), for storage of data extracted from RDF documents. The design of the RDSM model is complied with the OODB standard 2.0 of the Object Data Management Group (ODMG). Class definitions of the RDSM model are shown in Fig. 1.

A generalized node that forms the foundation of all building blocks in an RDF model is the *RDFNode* class. The actual building blocks, including subjects, predicates and objects, can all inherit from this class. The *URI* attribute is used to store the Uniform Resource Identifier (URI) when an instance of the *Resource* class or the *Literal* class is created. The *seqNum* attribute is useful for an instance acting as a predicate to record the position information where the predicate is placed in a container of a statement. The relationship *statement_of* and its inverse relationship *node_of* can be helpful to look up where this resource/literal resides.

By inheriting from the *RDFNode* class, the *Resource* and *Literal* classes can be created as physical instances of the subjects, predicates, and objects involved in RDF statements. For a *Resource* instance, we use the *ID* attribute to store the ID of this resource represented with the **rdf:ID** attribute to which other statements can refer while we are decomposing

```
class RDFNode
{
        attribute URI  string;
        attribute seqNum integer;
        relationship Statement statement_of
                inverse Statement::node_of;

        string getURI();
        void setURI(in string sURI);
        integer getseqNum();
        void setSeqNum(in integer iSeqNum);
        boolean addStatement(in Statement oStatement);
        Statement getStatement();
}

class Literal : RDFNode
{
        attribute literal string;

        string getLiteral();
        void setLiteral(in string sLiteral);
}

class Resource : RDFNode
{
        attribute ID string;

        string getID();
        void setID(in string sID);
}

class Statement
{
        attribute containerType string;
        relationship List <RDFNode> nodes_of
                inverse RDFNode::statement_of;

        string getContainerType();
        void setContainerType(in string sContainerType);
        Resource getSubject();
        boolean addResource(in Resource oResource);
        Resource getPredicate();
        boolean addPredicate(in Resource oPredicate);
        RDFNode getObject();
        boolean addObject(in RDFNode oObject);
}

class Model
{
        Model create();
        Model duplicate();
        Literal createLiteral(in string sURI,
                              in string sLiteral);
        Resource createResource(in string sURI,
                                in string sID);
        Statement createStatement(in Resource oSubject,
                                  in Resource oPredicate;
                                  in RDFNode oObject,
                                  in string sContainerID);
        void addStatement(in Statement oStatement);
        void removeStatement(in Statement oStatement);
}
```

Fig. 1. Class definitions of the RDSM model.

an RDF document. Note that the *URI* attribute in an instance of the *Literal* class is used to store the data type URI of this instance, and the actual value in string literal form is stored in the *literal* attribute of the *Literal* instance.

To present a statement that consists of a specific subject, predicate and object, we use the *Statement* class as the repository to store the information needed to construct an RDF statement. In an instance of the *Statement* class, we use a *List* object to keep track of the resources and literals that are used to form this statement. To record the type of a container that might be used in a statement, we simply store the information about the type of the container by putting one of the values "Bag," "Seq" and "Alt" in the *containerType* attribute.

The *Model* class can be used to create the model that would contain several interconnected statements. We use its operations to create a new model, duplicate an existing model, create a resource or literal, or create a new statement.

B. Decomposing RDF/XML Documents

While receiving an RDF document, we need to decompose it into components that our RDF data store can accept and store. Before introducing the process of decomposing an RDF document, we first consider the following cases:

■ *Typed Literals:* These specify type information of the corresponding literals. Fig. 2(a) shows an example of a typed literal. When we decompose an RDF document, type information will be preserved for the convenience of query processing. For a literal that is provided without type information, we will try to analyze its structure and determine a reasonable type for it. Since the XML Schema standard provides many built-in data types, we preload these data types as fundamental RDF resources for quotation purpose.

■ *Nesting Statements:* As the example shows in Fig. 2(b), the inner **rdf:about** attribute plays a double role. It is the value of the **s:Composer** property in the first statement, and is also the subject of a further statement. When we encounter this case, we first flatten nesting statements into several simple RDF triples, and then we internally group the sharing resources to improve storage performance.

■ *Containers:* A container structure can be used to indicate a group of things. Fig. 2(c) shows a typical **rdf:Bag** container structure. A container structure can be processed similarly to the way for processing nesting statements, except for the **rdf:Seq** structure. The order information of the involving members needs to be recorded to preserve the semantics of an **rdf:Seq** container instance.

Taking into account of the three cases listed above, we develop an algorithm for decomposing an RDF document. We adopt the Document Object Model (DOM) in [9] for processing RDF documents and assume that the namespace URIs used in a given document are all declared as attributes of the **rdf:RDF** element for performance consideration. To further improve the performance of the algorithm, we will not process nodes such as instruction and comment nodes. The algorithm, which is shown in Fig. 3, executes the following steps:

```
<rdf:Description rdf:about="http://www.imdb.com/name/nm0515908">
  <s:age rdf:datatype="http://www.w3.org/2001/XMLSchema#
    decimal">58</s:age>
</rdf:Description>
```

(a)

```
<rdf:Description rdf:about="http://www.imdb.com/name/nm0515908">
  <s:Composer>
    <rdf:Description rdf:about="http://www.imdb.com/title/tt0293508">
      <s:Name>The Phantom of the Opera</s:Name>
      <s:publishedIn>2004</s:publishedIn>
    </rdf:Description>
  </s:Composer>
</rdf:Description>
```

(b)

```
<rdf:Description rdf:about="http://www.imdb.com/name/nm0515908">
  <s:Composer>
    <rdf:Bag>
      <rdf:li rdf:resource="http://www.imdb.com/title/tt0293508"/>
      <rdf:li rdf:resource="http://www.imdb.com/title/tt0173714/"/>
      <rdf:li rdf:resource="http://www.imdb.com/title/tt0070239/"/>
    </rdf:Bag>
  </s:Composer>
</rdf:Description>
```

(c)

Fig. 2. Three special RDF/XML serializations.

1) Record the namespace prefixes and the associated URIs used in the whole RDF document.

2) Create a *Model* instance for organizing and grouping all of the resources and literals extracted from the document.

3) Extract a subject by retrieving the URI representing this resource; use the *URI_GENERATOR()* function to generate a URI for an anonymous resource; record the value of the **rdf:ID** attribute for future use.

4) Extract the localname of a found predicate and combine it with the corresponding URI prefix to form the full URI of the predicate.

5) Check whether the object of the predicate is a resource or not. If the object is a resource, acquire the URI of a named resource or call the *URI_GENERATOR()* function to generate a URI for an anonymous resource. If the object is a string literal, get the value in string form, and so does the corresponding data type if it is explicitly specified in the document (or analyzed by the decomposition process).

6) Use the *CONSTRUCT_STATEMENT()* function to create a new statement if no further object can be extracted, else continue to extract the objects belonging to a nesting statement or a container statement.

IV. Retrieving RDF/XML Documents

A. Generic RDF API

To provide the ability to access and query RDF data stored in our RDF data store, we introduce a generic RDF API for accessing and querying RDF data. The set of operations is generic RDF Model operations and can be implemented by

```
Algorithm RDF/XML Document Decomposition Algorithm
➢ Extract the namespace prefixes and the associated URIs used in the
  document.
➢ Construct a Model instance for organizing and grouping all building
  blocks extracted from the given document.
➢ For each first-level rdf:RDF node do
      ⇒ Generate a Resource instance as the subject of a triple with the
        corresponding URI or with URI_GENERATOR() function.
      ⇒ Get the ID indicated with the rdf:ID attribute, if any.
    For each second-level rdf:RDF node do
        ⇒ Combine the namespace URI corresponding to the prefix of
          the found predicate with its localname.
            ⇒ Get the object of current predicate as a named resource
              or an anonymous resource generated with
              URI_GENERATOR() function, if the object is a
              resource.
            ⇒ Get the type information and the string literal of the
              object, if the object is a literal.
            ⇒ Construct a RDF statement with
              CONSTRUCT_STATEMENT() function by passing the
              newly extracted subject, predicate, and object as the
              function's parameters.
        ⇒ If the current predicate contains subnode(s) then
            ⇒ Iteratively get every container member object or the
              nesting object, and then construct statement for every
              newly acquired member object.
End Algorithm
```

Fig. 3. RDF/XML document decomposition algorithm.

any service that wishes to expose RDF to client applications. Note that the *StatementSet* or any similar set collection can be presented in RDF/XML format. The operations involved in the API are described below.

■ *Get Object:* This operation allows extracting the object of a given statement. The full URI is returned if the object is a resource. The string literal with the associated data type URI is returned if the object is a literal.

LiteralAppendedURI **getObject (***Statement***);**

LiteralAppendedURI: Full URI or literal with the associated data type's full URI of the object.
Statement: Statement for the operation.

■ *Get Predicate:* This operation allows extracting the predicate of a given statement.

URI **getPredicate (***Statement***);**

URI: Full URI of the predicate.
Statement: Statement for the operation.

■ *Get Subject:* This operation allows extracting the subject of a given statement.

URI **getSubject (***Statement***);**

URI: Full URI of the subject.
Statement: Statement for the operation.

■ *Get Statements:* This operation matches the template (*Subject, Predicate, Object*) against the model, where the slots are either a fixed value (URI or literal) or a wildcard (meaning match anything in this position).

StatementSet **getStatements (***Subject, Predicate, Object***);**

Subject: URI or * (wildcard).
Predicate: URI or *.
Object: URI, literal or *.
StatementSet: Set of statements returned.

■ *Insert Statements:* This operation allows inserting multiple statements into the RDF data store. If a statement already exists within the model, then it is not duplicated nor is an exception thrown.

insertStatements (*StatementSet***);**

StatementSet: Set of statements for the operation.

■ *Query:* This operation allows implementing a query language to query data in the data store. It accepts the *Query* parameter in the syntax of a subset of the W3C's SPARQL language to query data in our RDF data store.

StatementSet **query (***Query***);**

Query: The query to be executed.
StatementSet: Set of statements returned.

■ *Remove Statements:* This operation allows removing all of the statements specified in the *StatementSet* parameter.

removeStatements (*StatementSet***);**

StatementSet: Set of statements for the operation.

■ *Update Statements:* This operation removes one set of statements (specified with the *RemoveSet* parameter) from the data store and inserts another set of statements (specified with the *InsertSet* parameter) into the data store.

updateStatements (*RemoveSet, InsertSet***);**

RemoveSet: Set of statements to be removed.
InsertSet: Set of statements to be inserted.

B. Querying RDF Data

The SPARQL language [10], which is under development by the W3C's RDF Data Access Working Group (DAWG), is a query language and data access protocol for the Semantic Web. We support the SELECT query to provide high-level querying capabilities to the clients.

SPARQL is built on the *triple pattern*, which is written as subject, predicate, and object and is terminated with a full stop (i.e., “.”). URIs are written inside angle brackets (i.e., “<” and “>”). String literals are denoted with either double quotes (i.e., “"”) or single quotes (i.e., “'”). The keyword PREFIX binds a prefix to a namespace URI. A prefix binding applies to any QNames in the query with that prefix. Variables are indicated by “?”; however, “?” does not form part of the variable. “$” is an alternative to “?”. In a query, $author and ?author denote the same variable. The SELECT clause is used to define the data items that will be returned by a query. The WHERE clause uses braces (i.e., “{” and “}”) to group a collection of triple patterns, and this collection is called a *graph pattern*. Each of the triple patterns must match for the graph pattern to match. Matching a triple pattern to a graph gives bindings between variables and building blocks in a RDF graph so that the triple pattern, with variables replaced by corresponding RDF data, is a triple of the graph being matched. The result of a query is a sequence of results that form a table or result set. Each row in the table corresponds to one query solution, and each column corresponds to a variable declared in the SELECT clause.

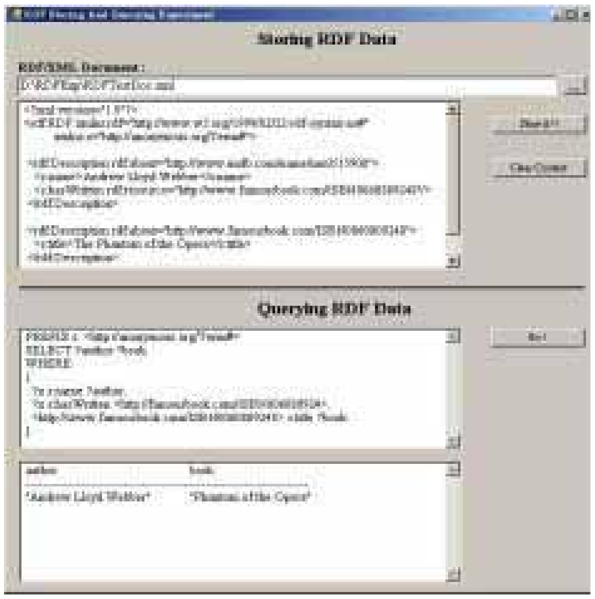
By combining with the API introduced in the previous subsection, we can create a system for querying RDF data stored in our data store. Firstly, we iteratively retrieve each of the triple solutions to a specified triple pattern and connect it to other triple solutions from other triple patterns to form a candidate graph. Then, we look up the bindings between the variables specified in the SELECT clause and the candidate graph to acquire the query result. The algorithm for acquiring query results is shown in Fig. 4.

V. Performance Evaluation

We have implemented an experimental system for storing and retrieving RDF documents. In the experimental system, databases are built on the Computer Associates' Jasmine II object-oriented database management system and programs are written in the C# object-oriented programming language under the Microsoft Visual Studio 2005 development environment. The hardware platform is with the equipments of Pentium III 800MHz CPU, 512Mbytes of RAM, and 80Gbytes of hard disk storage capacity. The graphical user interface (GUI) of the experimental system is shown in Fig. 5.

```
Algorithm RDF Query Answering Algorithm
➢ Query all of the possible triple pattern solutions for every triple
  pattern involved in a graph pattern.
➢ For each graph pattern solution constructed by picking up one of the
  triple pattern solutions from each triple pattern do
    ⇒ If the bindings between the variables in the SELECT clause and
      the graph pattern solution exist then
        ⇒ Acquire the bindings as one solution of the query result
          set.
End Algorithm
```

Fig. 4. RDF query answering algorithm.



```
[illegible]
```

author	book
"Andrew Lloyd Webber"	"Phantom of the Opera"

Fig. 5. GUI of the experimental system.

The application's window is divided into two parts. The upper part of the window is configured for loading the RDF document. The user can specify the document by entering the document's full path or invoking a file selection dialog box by clicking the “…” button. The specified document is then shown in the display area and the user can load it into the data store by clicking the “Store it!!” button. The lower part of the window is arranged for querying RDF data. The user issues a query statement in the entering area and clicks the “Go!” button to execute the query statement. The query result is then displayed in the display area in a tabular format.

To evaluate the performance of our storing process, we conducted the test by taking test files under the categories of “Examples From the RDF Model and Syntax Specification”, “Automatically Generated RDF Files”, and “Miscellaneous Examples” listed on the web page named “RDF Examples and Miscellaneous Tests” (http://www.w3.org/2000/10/rdf-tests/) as the input files of our experimental system. The results of the evaluation are shown in Table 1.

Because the original contents of several test files were not fully complied with the RDF/XML syntax defined in the RDF standard suite, we modified their original contents to make these files testable for our system. From the results shown in Table 1(a) and 1(c), we can see that most of the test files can be processed in less than 3 seconds. Even though our system gets the worst performance shown in Table 1(b), since the file size in this category is up to 5 kilo bytes, the average processing time is still less than 4 seconds. From the results of the evaluation, we can say that our storing process is acceptably efficient for decomposing and storing RDF documents we selected, and it is reasonable to presume that our storing process can process almost all of the RDF documents.

TABLE 1

EVALUATION RESULTS OF THE PROPOSED STORING PROCESS

Category Name	Examples From the RDF Model and Syntax Specification
Number of Files	14
Longest Time	5 sec.
Average Time	2.85 sec.

(a)

Category Name	Automatically Generated RDF Files
Number of Files	4
Longest Time	5 sec.
Average Time	4 sec.

(b)

Category Name	Miscellaneous Examples
Number of Files	534
Longest Time	4 sec.
Average Time	2.18 sec.

(c)

To evaluate the performance of our querying process, we prepared ten properly designed queries (Q1-Q10) over the contents of the test files previously stored. The estimated number of returned triples of these queries is well arranged at an incremental proportion. That is, the number of returned triples of the tenth query (Q10) is anticipated roughly being ten times of the number of returned triples of the first query (Q1). The results of the evaluation are shown in Fig. 6.

As shown in Fig. 6, the equation indicates that the elapsed time of these queries represents a liner relationship, and the variance (R^2) indicates that the elapsed time of a query with a known number of returned triples can be precisely estimated. From the results of the evaluation, we can see that the elapsed time of a query is proportional to the number of returned triples of the query. We can also precisely estimate the elapsed time for a query that returns a known number of triples.

VI. Conclusion and Future Work

In this paper, we addressed the important issue of efficient storage and retrieval of RDF documents and proposed an object-oriented approach to this issue. We proposed an object-oriented data model for our RDF data store and a decomposition algorithm for extracting RDF building blocks. We also proposed a generic RDF API for accessing and querying RDF data in the data store, as well as a query-answering algorithm that combines the functionality of the generic RDF API and the emerging SPARQL query language. Experiments showed that the proposed approach is efficient and effective.

As for future work, we are going to make our decomposition algorithm more comprehensive to process the inferring affairs of RDF data. Besides, we are going to improve the performance of our generic RDF API. Query rewriting and optimization of the SPARQL language are expected to be our long-term research objectives.

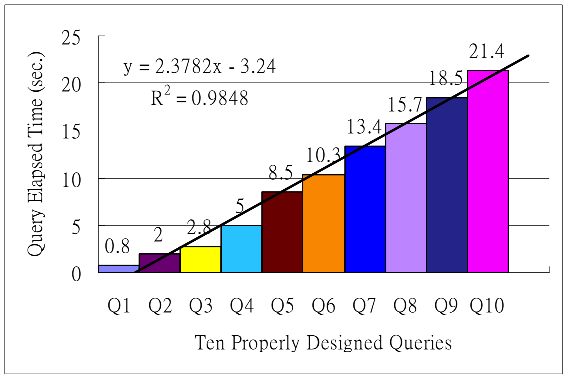


Fig. 6. Evaluation results of the proposed querying process.

Acknowledgment

The author would like to express his appreciation for the financial support from the National Science Council of Republic of China under Project No. NSC95-2221-E-031-005.

References

[1] G. Klyne, J. J. Carroll and B. McBride, *Resource Description Framework (RDF): Concepts and Abstract Syntax,* W3C Recommendation, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210, February 2004.

[2] F. Manola, E. Miller and B. McBride, *RDF Primer,* W3C Recommendation, http://www.w3.org/TR/2004/REC-rdf-primer-20040210, February 2004.

[3] D. Brickley, R. V. Guha and B. McBride, *RDF Vocabulary Description Language 1.0: RDF Schema,* W3C Recommendation, http://www.w3.org/TR/2004/REC-rdf-schema-20040210, February 2004.

[4] T. Berners-Lee, J. Hendler, and O. Lassila, *The Semantic Web,* Scientific American, http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21, 2001.

[5] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura, "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases," *ACM Transactions on Internet Technology,* vol. 1, no. 1, pp. 110-141, 2001.

[6] K. Runapongsa and J. M. Patel, "Storing and Querying XML Data in Object-Relational DBMSs," *Lecture Notes in Computer Science,* vol. 2490, pp. 266-285, March 2002.

[7] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton, "Relational Databases for Querying XML Documents: Limitations and Opportunities," in *Proceedings of the 25th International Conference on Very Large Databases,* Edinburgh, Scotland, 1999.

[8] D. Florescu and D. Kossmann, "Storing and Querying XML Data using an RDBMS," *IEEE Data Engineering Bulletin, vol. 22, no. 3, pp. 27-34*, 1999.

[9] A. L. Hors, et al., *Document Object Model (DOM) Level 2 Core Specification Version 1.0,* W3C Recommendation, http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113, November 2000.

[10] E. Prud'hommeaux and A. Seaborne, *SPARQL Query Language for RDF,* W3C Candidate Recommendation, http://www.w3.org/TR/2006/CR-rdf-sparql-query-20060406, April 2006.

CXPath: a Query Language for Conceptual Models of Integrated XML Data

Diego de Vargas Feijó, Claudio Naoto Fuzitaki, Álvaro Moreira[*]
Renata de Matos Galante[†] Carlos Alberto Heuser[‡]
Universidade Federal do Rio Grande do Sul – UFRGS
Instituto de Informática, Porto Alegre, RS, Brasil
Email: {diego, fuzitaki, afmoreira, galante, heuser}@inf.ufrgs.br

Abstract

In order to search for the same information in heterogeneous XML data sources, on the Web or in multiple databases in an enterprise, one must write a specific query in accordance with the structure of each XML source. A better solution is to state a single query against a global conceptual schema and then translate it automatically into an XML query for each specific data source. CXPath (for Conceptual XPath) has been proposed as a language for querying XML sources at the conceptual level. In this paper we improve the language original proposal by extending it with queries using inheritance and self-relationships, and by giving a formal specification of the criteria for validating CXPath queries against conceptual models.

1. Introduction

Data integration systems provide users with a uniform interface to multiple data sources. It is required when querying different sources on the Web, querying multiple databases within an enterprise, and querying disparate parts of a large-scale scientific experiment. Without data integration, the user must have knowledge about the structure of each data source and write a query for each one of them.

The XML format has been extensively used to represent and to interchange data among users and applications, specially through the Web [20]. There are several works on the integration of semistructured data [2, 13, 12] and XML sources [6, 8, 11, 18, 14]. We focus on the problem of performing queries on heterogeneous XML data sources related to some specific domain.

In this context, the main challenge is to deal with different XML representations of semantically equivalent data. A solution to this problem requires [10]: *(i)* a global (unified) representation that captures XML source schemata of a specific domain; *(ii)* a language to write global queries in accordance to the global representation; *(iii)* a translation mechanism to convert these global queries into queries in accordance to the schema of each XML source; and *(iv)* an instance integration mechanism to unify query results coming from different XML sources into a single query result matching the global representation.

In this paper we consider the first two points above since we present a global representation (i.e., a conceptual model) that can express concepts such as inheritance and self-relationships, and we extend and formally define the language CXPath [3] for querying this representation.

CXPath (for *Conceptual XPath*) is a language for querying conceptual models that result from the integration of XML sources. CXPath is one of the outcomes of a research effort that has lead to a semi-automatic and bottom-up process for semantic integration of XML Schemata called BinXS [14].

The contributions of this paper are: (i) an extension of CXPath queries to deal with inheritance and self-relationships, not present in the original language proposal, and (ii) a formal specification of the criteria for validating global queries, written in the CXPath language, against a conceptual model.

A conceptual model, where inheritance and self-relationship can be represented, is more expressive and, at the same time, simplifies the task of designing web applications that have to deal with heterogeneous XML data sources. The extensions allow more concise queries and a more natural representation of the schema integration. The resulting formalization, in its turn, provides a precise description of the language and can be used as a reference for implementors and query planners.

The rest of this paper is structured as follows. Section 2 presents the conceptual model CXPath is based on extended with inheritance and self-relationship. Section 3 presents the main features of the CXPath query language through a series of examples and illustrates how it can be used to build queries at the conceptual level. Section 4 has a set of inference rules that specify the criteria for validating CXPath queries. Section 5 discusses related work. Main ideas of this paper, and future work, are summarized in Section 6.

[*]This work has been partially supported by CNPq under grant No. 481516/2004-2 and Fapergs under grant No. 0412264.
[†]This work has been partially supported by Fapergs under grant No. 0412264, CNPq No. 550.845/2005-4 and CNPq No. 475.743/2004-0
[‡]This work has been partially supported by CNPq under grant No. 473310/2004-0

2. Conceptual Model

A conceptual model for defining a global schema has been adopted, instead of the logical XML model, because the XML model is unable to abstract several XML schemata at the same time. Considering the domain of bibliographical references, for instance, a many-to-many authorship relation between `Publication` and `Person` may be represented by two different XML schemata: *(i)* one `Publication` (ancestor element) associated to many `Persons` (descendent elements); or *(ii)* one `Person` (ancestor element) associated to many `Publications` (descendent elements). A global XML schema would be able to represent only one of these possibilities. However, a conceptual model directly represents many-to-many relationships (many `Publications` associated to many `Persons` , and vice-versa), without imposing a strict navigation order between them.

Figure 1 shows a simple conceptual schema for a domain of bibliographical references. `Publication` is a non-lexical concept (solid rectangle), being composed by information about `Title`, `Year` and `Person` . `Title`, `Year` and `Name` are lexical concepts (dotted rectangles), holding textual information. An anonymous *association relationship* is defined between `Publication` and `Year`, denoting that a publication has one associated year information, and a year is associated to one or more publications. An association relationship named `author` is defined between `Publication` and `Person` , denoting that a person may be an author of several publications. There is also anonymous association relationships between `Publication` and `Title`, and between `Person` and `Name`.

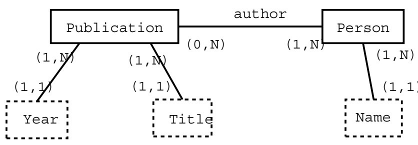


Figure 1. A simple conceptual model.

A discussion about the process of building a conceptual model from heterogeneous XML sources is out of the scope of this paper (it is described in detail in [14]).

2.1. Inheritance

In this paper, the conceptual model presented in [14] is extended with inheritance and self-relationship. An *inheritance relationship* is a binary relationship between a generic concept and a sub-concept. Figure 2 illustrates the modeling of an *inheritance relationship* where `Publication` is the generic concept, `Article` and `Book` are sub-concepts. In the example, `Article` extends `Publication` with a relationship `referee` to the concept `Person` , and `Book` also extends `Publication` . A model like the one depicted in Figure 2 can be the result of a data source such as the one shown in Figure 3(a), where a publication element has books and articles as subelements.

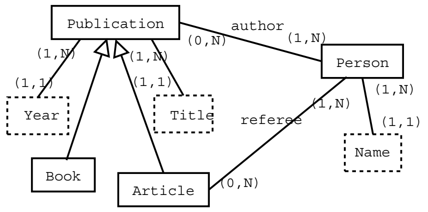
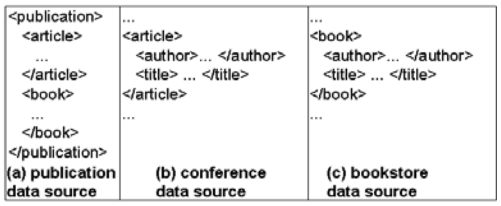


Figure 2. Inheritance relationship.



<publication>	...	...
<article>	<article>	<book>
...	<author>... </author>	<author>... </author>
</article>	<title> ... </title>	<title> ... </title>
<book>	</article>	</book>
...	...	...
</book>		
</publication>		
(a) publication data source	(b) conference data source	(c) bookstore data source

Figure 3. Data sources that give rise to inheritance in a conceptual model.

Alternatively, it also could result from integrating the data sources shown in Figure 3(b) and Figure 3(c), where article and book elements have a common structure.

2.2. Self-relationship

A publication can make references and/or it can be referenced by other publications. Figure 4 exemplifies a situation where a *self-relationship* named `bibliography` is convenient. A publication, in the self-relationship `bibliography` depicted below, can play two different roles: either it `has_reference` or a `is_reference_of` of another publication. A model like the one depicted in

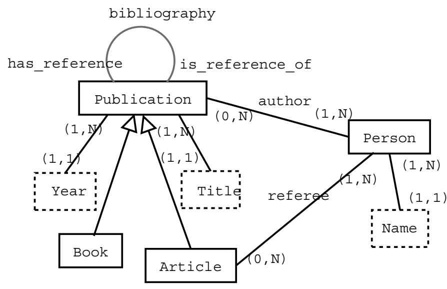
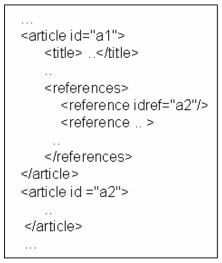


Figure 4. Self-relationship.

Figure 4 can be the result of a data source such as the one shown in Figure 5, where an article “a1” makes reference to article “a2”. Both inheritance and self-relationship were absent from the initial proposal for the conceptual model [14] and its associated query language [3]. They are essential not only for writing more concise queries but a conceptual model where these relationships can be represented is more expressive thus simplifying the task of designing web applications.



```
…
<article id="a1">
    <title> ..</title>
    ..
    <references>
        <reference idref="a2"/>
        <reference .. >
        ..
    </references>
</article>
<article id ="a2">
    ..
 </article>
 …
```

Figure 5. Data source that gives rise to a self-relationship in a conceptual model.

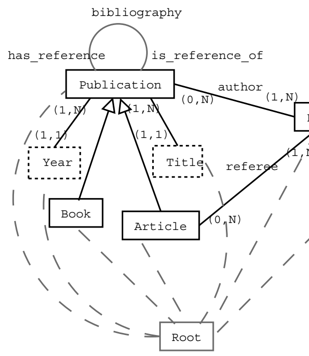


Figure 6. Example with Root.

3. CXPath

Although syntactically based on XPath, CXPath has a different semantics because it is applied to a different data model. XPath is suitable for navigating in XML documents that are tree-based structures, whereas CXPath is suitable for navigating over a conceptual schema that is a graph-based structure.

In the remaining of this section, we informally explain the semantics of CXPath queries over conceptual models by making a parallel with the semantics of XPath queries (we assume the conceptual model given in Figure 4).

Concept names instead of element names. In XPath, XML elements are referred by their labels (element names). In CXPath, concept names are used instead of element names. A concept name refers to all instances of that concept in the conceptual base.

Root elements and absolute path expressions. An XML instance has a *root element*, and an XPath expression that begins with slash (an *absolute* path expression) starts navigating from it. This can be done because the data model of an XML source is a tree. However, because its data model is a graph, a conceptual base does not have an initial element. The CXPath semantics for the absolute path expression is that the navigation may start at any concept in the conceptual base. A query like `/Article` for instance, retrieves all instances of the concept `Article`.

In order to keep the formal treatment of relative and absolute paths more uniform, a special concept, called *Root*, is introduced. It is distinguished from the others because it has a unique unnamed relationship with all other lexical and non-lexical concepts. Thus, it is always possible to navigate from the *Root* concept to any other concept. Observe though, that the name *Root* is not available when writing CXPath queries. Figure 6 shows the same model of Figure 4 extended with the *Root* concept. As *Root* is present in all conceptual models connected to all concepts it can be omitted in order to keep the presentation neat.

Navigation operator (slash operator) and relative path expressions. In XPath, the slash operator, when not appearing in the beginning of a path expression, has the semantics of "*navigate to the child elements*". As our data model is a graph and not a tree, the semantics of "/" has been changed to "*navigate to the related elements*". For verifying whether a navigation between two concepts is valid, it is necessary to examine if there is a relationship between then represented in the conceptual model. For example, the query `/Publication/Title` starts navigation from the `Root` concept and retrieves all instances of concept `Title` that are related to the instances of concept `Publication`.

Inheritance allows the navigation between concepts that do not have direct relationships. Considering Figure 6, the following query `/Article/Title` retrieves the titles of all publications that are articles. This navigation is possible because the specialized concept `Article` inherits all the relationships of the generic concept `Publication`.

Qualified navigation operator, relationship name and predicates. When more than one relationship relates two concepts, the identification of a specific relationship to be navigated may be needed. For example, the expression `/Article/Person/Name` retrieves all names of `Person` that are related to instances of `Article`, thus including relationships `author` (inherited from `Publication`) and `referee`, while the query `/Article/{author}Person/Name` retrieves all names of persons that are authors of articles. This selection of relationships by name is exclusive of CXPath, and there is no counterpart in XPath.

It is also possible to restrict values by using a predicate. A query like

```
/Article[Year="2007"]/{author}Person/Name
```

for instance, retrieves all names of persons that are authors of articles produced int the year 2007. A restriction in CXPath, not present in XPath, is that, in predicates, paths should end with a lexical entity.

Qualified navigation operator and role name. In the case of self-relationships, a *role name* can be necessary to indicate the direction of the navigation. Considering the Figure 4, the following query

```
/Publication[Year="2007"]/
{bibliography.has_reference}Publication
```

for instance, retrieves all publications that are reference for publications in the year 2007. Another query

```
/Publication[Year="2007"]/
{bibliography.is_reference_of}Publication
```

retrieves all publications that make references to 2007 publications.

4. Validating CXPath against Conceptual Models

Not all syntactically correct CXPath expressions are valid queries. A query such as `/Person/Year` for instance, is not valid against the conceptual model of Figure 4, since there is no navigation path connecting these two concepts. Another syntactically correct but invalid query is

```
/Article[{author}Person = {referee}Person]/
Title
```

`Person` is a non lexical concept and non lexical concepts can not be compared.

A query language is nothing more than a programming language with special purposes [7] and the validation problem, in relation to a model, is similar to the typing problem in programming languages. For this reason, we adopt a style that is widely used in the formal definition of programming languages, and also in the formal description of the static and dynamic semantics of the XPath and XQuery languages [9]

In what follows, we give a set of rules that specify the conditions that must hold for a CXPath query to be valid against a conceptual model. Each rule has the form

$$\frac{\mathit{premise_1} \ldots \mathit{premise_n}}{\mathsf{CM}, \mathit{cxt_1} \ldots \mathit{ctx_m} \vdash \mathit{CXPath\ expression}}$$

where $n \geq 0, m \geq 0$, meaning that, if all the premises hold, the $\mathit{CXPath\ expression}$, when considered in a context given by $\mathit{cxt_1} \ldots \mathit{ctx_m}$, is valid in relation to a conceptual model CM.

All the rules are defined on the syntactical structure of CXPath queries and there is exactly one rule for each construct. For these reasons, an inference algorithm can be easily extracted from them. We have implemented a prototype in Haskell that take as input a conceptual model and a CXPath query, and returns whether the query is valid or not. The implementation is available in `www.inf.ufrgs.br/~cnaoto/cxpath`. The syntactical structure of CXPath is given in Figure 7. A CXPath expression can be a relative path expression or an absolute path expression. An absolute path expression may be just "/" (slash) or a "/" followed by a relative path expression. A relative path expression may optionally specify a relationship and role name, followed by a required concept identifier, followed by an optional predicate. In the grammar *op* stands for the relational operators.

Before explaining the rules we start by giving a formal definition of conceptual model as discussed in Section 2. A lexical concept is represented by a pair (c, t), where c represents the lexical concept name, and t represents its type (string or integer).

An *association relationship* is represented by a n-tuple (c_1, c_2, r, p_1, p_2), where c_1 is a source concept name, c_2 is

$$\begin{array}{lcl}
\mathit{CXPath} & ::= & \mathit{RelPath} \\
 & | & \mathit{AbsPath} \\
 & & \\
\mathit{AbsPath} & ::= & / \\
 & | & /\mathit{RelPath} \\
 & & \\
\mathit{RelPath} & ::= & \mathit{Rel\ id\ Preds} \\
 & | & \mathit{Rel\ id\ Preds}\ /\mathit{RelPath} \\
 & & \\
\mathit{Rel} & ::= & \{\ \mathit{RelName}\ \} \\
 & | & \{\ \mathit{RelName.RoleName}\ \} \\
 & | & \varepsilon \\
 & & \\
\mathit{Preds} & ::= & [/\mathit{RelPath}_1\ \mathit{op}\ /\mathit{RelPath}_2]\ \mathit{Preds} \\
 & | & [/\mathit{RelPath}_1\ \mathit{op}\ \mathit{RelPath}_2]\ \mathit{Preds} \\
 & | & [/\mathit{RelPath}\ \mathit{op}\ \ \mathit{Literal}]\ \mathit{Preds} \\
 & | & [\ \mathit{RelPath}_1\ \mathit{op}\ \ /\mathit{RelPath}_2]\ \mathit{Preds} \\
 & | & [\ \mathit{RelPath}_1\ \mathit{op}\ \ \mathit{RelPath}_2]\ \mathit{Preds} \\
 & | & [\ \mathit{RelPath}\ \mathit{op}\ \ \mathit{Literal}]\ \mathit{Preds} \\
 & | & \varepsilon
\end{array}$$

Figure 7. CXPath Grammar.

a target concept name, and r is the relationship's name (ϵ when the relationship is unnamed). When c_1 and c_2 are the same, the n-tuple represents a self-relationship, in this case p_1 and p_2 are the names of the roles assumed by concept instances in the relationship (ϵ when the self-relationship has no roles - or when it is not a self-relationship)[1].

An *inheritance relationship* is represented by a pair (c_g, c_e), where c_g and c_e are the generic and specialized concept names, respectively. Both c_g and c_e must be non-lexical concepts.

Finally, a conceptual model CM is given by a tuple (NL, L, IR, AR) where: *NL* is a set of non-lexical concepts, *L* is a set of lexical concepts, and *IR* and *AR* are sets of inheritance and association relationships between concepts, respectively.

The rules are grouped into rules for absolute path expressions, relative path expressions, relationships, and predicates. This classification follows the syntactic categories in Figure 7. In what follows, we explain each one of these group of rules. In some rules we write CM_{AR} for the component AR of the conceptual model CM, and similarly for other components of a conceptual model.

Absolute Path Expressions. We have two rules for validating absolute path expressions: one for each clause in the grammar of Figure 7 for $\mathit{AbsPath}$ (/ and $/\mathit{RelPath}$). By the rule (APE1), the absolute path expression / is always well formed:

$$\frac{}{\mathsf{CM} \vdash_{\mathsf{APE}} /} \qquad (\textsc{Ape1})$$

Rule (APE2) says that, to validate an absolute path $/\mathit{RelPath}$, it is necessary to validate $\mathit{RelPath}$ using the group of rules for relative path expressions (the premise of rule (APE2)). Note that for typing relative path expressions it is necessary to add to the context the concept that antecedes it (the Root concept in this case):

[1]In order to keep the formal treatment simpler we omit the cardinality information from tuples representing association relationships since they are not relevant for the purpose of validating CXPath queries against a conceptual model.

$$\frac{\mathsf{CM}, Root \vdash_{\mathsf{RPE}} RelPath}{\mathsf{CM} \vdash_{\mathsf{APE}} /RelPath} \quad (\textsc{Ape}2)$$

Relative Path Expressions. The rule (RPE1) is for path expressions in the format $Rel\ id\ Preds$:

$$\frac{\mathsf{CM}, id_1, id_2 \vdash_{\mathsf{REL}} Rel \qquad \mathsf{CM}, id_2 \vdash_{\mathsf{PRE}} Preds}{\mathsf{CM}, id_1 \vdash_{\mathsf{RPE}} Rel\ id_2 Preds} \quad (\textsc{Rpe}1)$$

and rule (RPE2) below is for path expressions in the format $Rel\ id\ Preds/RelPath$:

$$\frac{\begin{array}{l}\mathsf{CM}, id_1, id_2 \vdash_{\mathsf{REL}} Rel \\ \mathsf{CM}, id_2 \vdash_{\mathsf{PRE}} Preds \qquad \mathsf{CM}, id_2 \vdash_{\mathsf{RPE}} RelPath\end{array}}{\mathsf{CM}, id_1 \vdash_{\mathsf{RPE}} Rel\ id_2 Preds/RelPath} \quad (\textsc{Rpe}2)$$

Observe that the first two premises of these rules are the same (one for the relationship Rel, and the other for the predicates $Preds$). The third premise of rule (RPE2) validates $RelPath$ against both the conceptual model, and the last identifier of its previous path expression (id_2 in these two rules).

Relationships. The verification if a relationship between concepts in a path expression is valid is done by verifying three possible situations. The first situation verifies named relationships ($RelName$); the second, besides named relationship, verifies role names ($RelName.RoleName$); the third has neither named relationship nor role name (ϵ).

For simplicity, the representation of relationships in a conceptual model is done by indicating a source and a destination concept. But the navigation can be done in any direction. Thus, the source concept may act as source or destination, and the destination may act as source. For this reason, all rules must take this in consideration and to verify if there is a relationship that has the source concept (or destination) and the destination (or source) matching the concepts related in the path expression.

$$\frac{\begin{array}{c}\mathsf{CM} \vdash id_1 \prec id_{g1} \\ \mathsf{CM} \vdash id_2 \prec id_{g2} \\ (c_1, c_2, RelName, p_1, p_2) \in \mathsf{CM}_{AR} \\ (c_1 = id_{g1} \wedge c_2 = id_{g2}) \vee (c_1 = id_{g2} \wedge c_2 = id_{g1})\end{array}}{\mathsf{CM}, id_1, id_2 \vdash_{\mathsf{REL}} \{RelName\}} \quad (\textsc{Rel}1)$$

$$\frac{\begin{array}{c}id_1 = id_2 \\ (id_1, id_2, RelName, p_1, p_2) \in \mathsf{CM}_{AR} \\ p_1 = RoleName \vee p_2 = RoleName\end{array}}{\mathsf{CM}, id_1, id_2 \vdash_{\mathsf{REL}} \{RelName.RoleName\}} \quad (\textsc{Rel}2)$$

$$\frac{\begin{array}{c}id_1 \prec id_{g1} \\ id_2 \prec id_{g2} \\ (c_1, c_2, \epsilon, p_1, p_2) \in \mathsf{CM}_{AR} \\ (c1 = id_{g1} \wedge c_2 = id_{g2}) \vee (c_1 = id_{g2} \wedge c_2 = id_{g1})\end{array}}{\mathsf{CM}, id_1, id_2 \vdash_{\mathsf{REL}} \epsilon} \quad (\textsc{Rel}3)$$

Rule (REL2) is for typing self-relationships with associated role names. In this case, the source and the destination concepts must be the same.

Rules (REL1) and (REL3) are analogous; both contemplate the possibility of inheritance relationship between the concepts. Rule (REL1) is for named relationships while rule (REL2) is for unnamed relationships.

The notation $id_1 \prec id_2$, present in rules (REL1) and (REL2), means an inheritance relationship, where id_1 is the specialized concept and id_2 is the generic concept. The relation $id_1 \prec id_2$ holds when $(id_1, id_2) \in \mathsf{CM}_{IR}$. This relation is also reflexive and transitive.

Predicates. A predicate in CXPath is a relational operation between two expressions (path expressions and/or literals). The rules (PR1), (PR2), and (PR3) are for predicates with an absolute path as the left operand of op. The rules for predicates with a relative path, or a literal as the left operand are trivial variations and, for this reason, we omit them from the paper.

$$\frac{\begin{array}{c}LastId(RelPath_1) \in \mathsf{CM}_L \\ LastId(RelPath_2) \in \mathsf{CM}_L \\ \mathsf{CM} \vdash_{\mathsf{APE}} /RelPath_1 \\ \mathsf{CM} \vdash_{\mathsf{APE}} /RelPath_2 \\ \mathsf{CM}, id \vdash_{\mathsf{PRE}} Preds \\ \mathsf{CM} \vdash LastId(RelPath_1)\ op\ \ LastId(RelPath_2)\end{array}}{\mathsf{CM}, id \vdash_{\mathsf{PRE}} [/RelPath_1\ op\ \ /RelPath_2]Preds} \quad (\textsc{Pr}1)$$

$$\frac{\begin{array}{c}LastId(RelPath_1) \in \mathsf{CM}_L \\ LastId(RelPath_2) \in \mathsf{CM}_L \\ \mathsf{CM} \vdash_{\mathsf{APE}} /RelPath_1 \\ \mathsf{CM}, id \vdash_{\mathsf{RPE}} RelPath_2 \\ \mathsf{CM}, id \vdash_{\mathsf{PRE}} Preds \\ \mathsf{CM} \vdash LastId(RelPath_1)\ op\ LastId(RelPath_2)\end{array}}{\mathsf{CM}, id \vdash_{\mathsf{PRE}} [/RelPath_1\ op\ \ RelPath_2]Preds} \quad (\textsc{Pr}2)$$

$$\frac{\begin{array}{c}LastId(RelPath) \in \mathsf{CM}_L \\ \mathsf{CM} \vdash_{\mathsf{APE}} /RelPath \\ \mathsf{CM}, id \vdash_{\mathsf{PRE}} Preds \\ \mathsf{CM} \vdash LastId(RelPath)\ op\ Literal\end{array}}{\mathsf{CM}, id \vdash_{\mathsf{PRE}} [/RelPath\ op\ \ Literal]Preds} \quad (\textsc{Pr}3)$$

One requirement for a predicate to be considered valid is that, if a path expression is one of its operands, its last concept must be lexical.

Finally note that the rules make use of the following auxiliary function, called $LastId$, which simply returns the name of the last concept of a path

$$\begin{array}{l}LastId(Rel\ idPreds) = id \\ LastId(Rel\ idPreds/RelPath) = LastId(RelPath)\end{array}$$

Comparisons. The comparisons are relational operations involving lexical concepts and literals. The rules for validating comparisons are trivial and so are omitted.

5. Related Work

In the field of data integration, there are several works related to the integration of semistructured data [2, 13, 12] and XML sources [6, 8, 11, 18, 14, 17]. In particular, we are using BinXS [14] that is a semi-automatic and bottom-up process for semantic integration of XML Schemata. We adopted a conceptual model for defining a global schema instead of the logical XML model, because the XML model is unable to abstract several XML schemata at the same time.

We believe that our approach is more general than those mentioned above because CXPath is an XPath based language for building queries over a conceptual schema that is an abstraction of several XML sources, avoiding that the user must know the schema of each source to formulate queries. We have chosen for the conceptual level a language that is based on the concept of path expression to simplify the process of translation of a query at the conceptual level to a query at the XML level, and discarded languages like entity-relationship algebras [16] and SQL that are based on the join operation. Examples of query languages that are based on the concept of path expression are OQL for the object-oriented model [4], Lorel for semi-structured data [1] and XPath for the XML model. As we also aim at simplifying the process of learning of the proposed language for those acquainted with the XML, we chose to base the conceptual level query language on the XPath.

Recently there has been a lot of research activity in the formal definition of XML related languages such as XPath and XQuery [9]. A type system for a XML query language is given in [5] and it is used to verify if the query language operations respect the XML schema restrictions. [19] specifies a formal semantic for XML Schema while W3C Consortium [20] has an effort to specify an operational semantics and a type system for both XPath and XQuery [9].

6. Summary and Future Work

In this paper, we improve the original proposal of CXPath [3] by extending it with queries using inheritance and self-relationships, and by giving a formal specification of the criteria for validating CXPath queries against conceptual models.

We believe that this work is an essential step towards establishing other results about our approach for querying heterogeneous XML sources. Among these issues we mention establishing that CXPath queries lead to the same results as those of XPath queries produced by a translation process.

A next step to establish a proof of correctness of our aproach is to define rules that characterize the result of a CXPath query concerning a conceptual model, in other words, what should be the conceptual values resulting from conceptual queries expressions, considering a given conceptual model.

Considering the many XML sources that compose the Conceptual Model, we have to show that each translated query returns an XML value that, when abstracted, is contained in the expected result of the conceptual query. For achieving this we still need to formalize the translation method and the abstraction process.

In another direction, we are investigating the requirements related to the query containment [15] in data integration contexts, such as query optimization, independence of queries from updates, and query rewriting using views. A formal definition of query validity is essential in order to investigate these issues rigorously.

References

[1] S. Abiteboul. Querying semi-structured data. In *ICDT*, pages 1–18, 1997.

[2] S. Bergamaschi, S. Castano, M. Vincini, and D. Beneventano. Semantic integration of heterogeneous information sources. *Data & Knowledge Engineering*, 36(3):215–249, 2001.

[3] S. D. Camillo, C. A. Heuser, and R. S. Mello. Querying heterogeneous XML sources through a conceptual schema. In *Intl. Conf. on Conceptual Modeling (ER)*, volume 2813 of *LNCS*, pages 186–199, Chicago, IL, USA, Oct 2003. Springer.

[4] R. Cattell, D. Barry, D. Bartels, M. Berler, J. Eastman, S. Gamerman, D. Jordan, A. Springer, H. Strickland, and D. Wade. *The Object Data Standard: ODMG 3.0*. Morgan Kaufmann, San Francisco, 2000. 280p.

[5] D. Colazzo, G. Ghelli, P. Manghi, and C. Sartiani. Types for correctness of queries over semistructured data. In *WebDB*, pages 19–24, 2002.

[6] I. F. Cruz, H. Xiao, and F. Hsu. An ontology-based framework for XML semantic integration. In *IDEAS*, pages 217–226, 2004.

[7] C. J. Date. Some principles of good language design (with especial reference to the design of database languages). *SIGMOD Record*, 14(3):1–7, 1984.

[8] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD Conference*, 2001.

[9] D. Draper, P. Fankhauser, M. FERNÁNDEZ, A. Malhota, K. Rose, M. Rys, J. SIMÉON, and P. Wadler. XQuery 1.0 and XPath 2.0 formal semantics, May 2005. In W3C Working Draft. ¡http://www.w3.org/TR/2003/WD-xquery-semantics-20030502/¿.

[10] A. Elmargamid, M. Rusinkiewcz, and A. Sheth. *A Management of Heterogeneous and Autonomous Database Systems*. Morgan Kauffmann Publishers, 1999.

[11] B. F. Lóscio and A. C. Salgado. Generating mediation queries for XML-based data integration systems. In *SBBD*, pages 99–113, 2003.

[12] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *VLDB*, pages 49–58, 2001.

[13] P. McBrien and A. Poulovassilis. A semantic approach to integrating xml and structured data sources. In *CAiSE*, pages 330–345, 2001.

[14] R. S. Mello and C. A. Heuser. BInXS: A process for integration of XML schemata. In *Intl. Conf. Advanced Information Systems Engineering (CAISE)*, volume 3520 of *LNCS*, pages 151–166, Porto, Portugal, Jun 2005. Springer.

[15] T. D. Millstein, A. Y. Halevy, and M. Friedman. Query containment for data integration systems. *Journal of Computer and System Sciences*, 66(1):20–39, 2003.

[16] C. Parent and S. Spaccapietra. An entity-relationship algebra. In *ICDE*, pages 500–507, 1984.

[17] A. Poggi and S. Abiteboul. XML data integration with identification. In G. M. Bierman and C. Koch, editors, *DBPL*, volume 3774 of *Lecture Notes in Computer Science*, pages 106–121. Springer, 2005.

[18] P. Rodríguez-Gianolli and J. Mylopoulos. A semantic approach to xml-based data integration. In *ER*, pages 117–132, 2001.

[19] J. Siméon and P. Wadler. The essence of XML. In *POPL*, pages 1–13, 2003.

[20] W3C XML Work Group. XML - Extensible Markup Language, 2006. `http://www.w3.org/XML`.

OWL_d^e: Extending Knowledge for Web Ontology Language

Hichem Zaït, Aïcha Mokhtari
Computer Science Laboratory, LRIA/USTHB BP 32, El-Alia Bab Ezzouar Algiers, Algeria
E-mail:{hichem.zait@gmail.com, aissani_mokhtari@yahoo.fr}

Abstract

Recent semantic web based on the Web Ontology Language (OWL), with its Description Logic compatible sublanguage (OWL-DL), explicitly excludes defaults and exceptions. However, as a few concepts are definable using only strict knowledge, the terminological knowledge bases contain partially defined concepts, which represents the main cause of noise and silence in information retrieval. In this paper, we propose $OntoDL_d^e$ as a description language including default (d) and exception (e) connectives. Based on the later, we propose OWL_d^e as an extension of OWL with new RDF/XML constructors representation and we show the mapping between the two proposed languages. Finally, we give the structural concept algebra of the proposed solution and we prove that its computation is of polynomial complexity.

1 Introduction

The use of Web ontologies is essential for the creation of the semantic Web. However, we can't reach the real semantic without ensuring a complete definition of concepts using default knowledge. The works to extend conceptual definitions with default knowledge based on description logic becomes very important for many applications which appear to require default reasoning, at least if they are to be engineered in a maintainable way.

In (classical) DLs, concepts are defined with strict conceptual knowledge. However, defining concepts with default knowledge has several advantages. It enlarges the set of potentially definable concepts; indeed, this overcomes the obstacle that the need for subsumption algorithms to be tractable had led to such restrictions in DLs that a lot of concepts can't be defined.

In the literature there has been much work on the treatment of defaults in frame and DL-based systems [13, 15, 14, 16, 2, 7], where several formal tools borrowed from non monotonic logics have been adapted to the framework of description logics. Such an adaptation is not trivial, because description logics are not, in general, propositional languages, therefore it gives rise to both representational and reasoning problems. The above cited works are focused on defaults, thus not taking into account several other non-first-order features. Other work [8] presented a framework to express more non-first-order features but doesn't take into consideration Web ontology specifications. [9] proposed $AL_{\delta\epsilon}$, a description language including default (δ) and exception (ϵ) connectives for concept definition. However, $AL_{\delta\epsilon}$ excludes some Web ontology language connectives such as disjunction.

In this paper we define, inspired from [9], a specific description logic $OntoDL_d^e$ which allows us to express the OWL-DL with consideration of default and exception knowledge. Thus, we propose OWL_d^e language, an extension of OWL-DL including *default* and *exception* constructors and we give the grammar and the algebric semantic on which it is based. In section 2 we present a definition of the description language $OntoDL_d^e$. In section 4, we introduce OWL_d^e as the extension of OWL-DL and we present the expression of OWL_d^e using $OntoDL_d^e$. The structural concept algebra is given in section 4.2. Finally, we prove in section 4.4 that the subsumption in $OntoDL_d^e$ is for polynomial complexity.

2 The $OntoDL_d^e$ language

In this section we propose $OntoDL_d^e$ as a description language including default (d) and exception (e) connectives for concept definition and formally define the subsumption relation using an algebric semantic. It includes also all necessary connectives which will be mushed with OWL constructors (section 4).

$OntoDL_d^e$ is inductively defined from a set **R** of primitive roles and a set **P** of primitive concepts, augmented by the constant concept $\top$(top). In below abstract syntax rule d and e are two unary connectives, we use C_d to express that

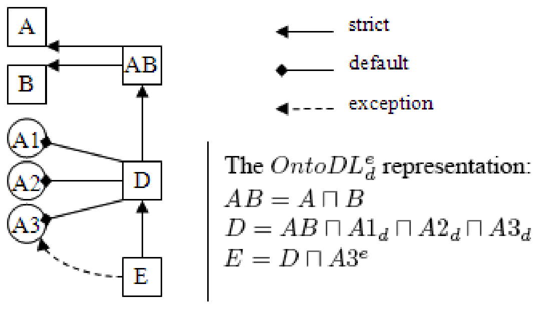
C is a default concept and we use C^e to express that C is defined as an exception. $\sqcap$ and $\sqcup$ are two binary connectives, $\forall$ and $\exists$ enable universal quantification on role values.

C,D $\rightarrow \top$	most general concept
$\vert\bot$	less general concept
$\vert P$	primitive concept
$\vert\neg P$	negation of a primitive concept
$\vert C \sqcap D$	concept conjunction
$\vert C \sqcup D$	concept union
$\vert\forall R : C$	value restriction
$\vert\exists R : C$	cardinality restriction
$\vert \geq nP$	minimal cardinality restriction
$\vert \leq nP$	maximal cardinality restriction
$\vert C_d$	default concept
$\vert C^e$	exception of the concept C

In the following we assume that concepts are partially ordered by a subsumption relation. A concept B is subsumed by a concept A if A is more general than B. Thus, concept B will inherit all properties of concept A.

For example, using $OntoG_d^e$ [1] (figure 1) the concept AB is subsumed by concepts A and B. The concept D is subsumed by the concept AC and by three default concepts ($A1$, $A2$ and $A3$). The concept E is subsumed by the concept D and by the the concept $A3$ by exception. We use rectangle to represent a class and circle to represent a concept property.

In the following we show how we can combine different conceptual terms in order to define new relations.



The $OntoDL_d^e$ representation:
$AB = A \sqcap B$
$D = AB \sqcap A1_d \sqcap A2_d \sqcap A3_d$
$E = D \sqcap A3^e$

Figure 1. Representation of default and exception concepts in $OntoG_d^e$

Equational system and definitions

The equational system for $OntoDL_d^e$ highlights the main properties of connectives and gives an equivalence relation between conceptual terms. Also, it reduces the number of properties used in the same definition.

Below, we define relations such as the associativity, commutativity, idempotence and neutral elements. We give also some other definitions which can be used when defining combination of inherited concepts [2].

Let us consider the following set of equations, where A, B, C belong to $OntoDL_d^e$:

$\sqcap$	$(A \sqcap B) \sqcap C = A \sqcap (B \sqcap C)$	def1.1
	$A \sqcap B = B \sqcap A$	def1.2
	$A \sqcap A = A \sqcap A = A$	def1.3
	$\top \sqcap A = A$	def1.4
	$A \sqcap \neg A = \bot$	def1.5
	$A \sqcap \bot = \bot$	def1.6
$\sqcup$	$(A \sqcup B) \sqcup C = A \sqcup (B \sqcup C)$	def2.1
	$A \sqcup B = B \sqcup A$	def2.2
	$A \sqcup A = A \sqcup A = A$	def2.3
	$\top \sqcup A = \top$	def2.4
	$A \sqcup \neg A = \top$	def2.5
	$A \sqcup \bot = A$	def2.6
d	$(A \sqcap B)_d = A_d \sqcap B_d$	def3.1
	$A \sqcap A_d = A$	def3.2
	$A^e \sqcap A_d = A^e$	def3.3
	$A_{dd} = A_d$	def3.4
e	$(A_d)^e = A^e$	def4.1
	$(A^e)^e = A^e$	def4.2

For example, if we replace the definition of concept D in concept E as defined in figure 1 we obtain the following syntax:

$D = AB \sqcap A1_d \sqcap A2_d \sqcap A3_d$
$E = D \sqcap A3^e$
$\quad = AB \sqcap A1_d \sqcap A2_d \sqcap \underline{A3_d \sqcap A3^e}$

The new syntax contains double definitions of concept $A3$, the definition by exception $A3^e$ and the inherited default definition $A3_d$. If now we apply the definition 3.3 above we replace the default definition of concept $A3$ by the exception, the new syntax will be:

$$E = AB \sqcap A1_d \sqcap A2_d \sqcap A3^e$$

Before presenting the OWL_d^e language, let us describe the Web Ontology based on DL.

3 Web Ontology based on DL

In order to allow sharing and reuse of ontologies on the Semantic Web, a common ontology language is required.

[1] Described separately in other work. We created $OntoG_d^e$ to represent graphically the ontology concepts based on $OntoDL_d^e$ language. We give here just a quick overview for demonstration purpose.

[2] All definitions are semantically proved.

OWL Abstract Syntax	DL syntax
Class axioms	
Class(A partial $C_1 \dots C_n$)	$A \sqsubseteq C_i$
Class(A complete $C_1 \dots C_n$)	$A \equiv C_1 \sqcap \dots \sqcap C_n$
EnumeratedClass(A $o_1 \dots o_1$)	$A \equiv \{o_1 \dots o_1\}$
SubClassOf($C_1 C_2$)	$C_1 \sqsubseteq C_2$
EquivalentClasses($C_1 \dots C_n$)	$C_1 \equiv \dots \equiv C_2$
DisjointClasses($C_1 \dots C_n$)	$C_1 \sqcap C_j \sqsubseteq \bot$
ObjectProperty(R super(R_1) … super(R_n))	$R \sqsubseteq R_i$
domain(C_1) … domain(C_n)	$\top \sqsubseteq \forall R^-.C_i$
range(C_1) … range(C_n)	$\top \sqsubseteq \forall R.C_i$
[inverseOf(R_o)]	R_o^-
[Symmetric]	R^-
[Functional]	$T \sqsubseteq \leq 1R$
[InverseFunctional]	$T \sqsubseteq \leq 1R^-$
[Transitive]	Trans(R)
SubpropertyOf(Q_1 Q_2)	$Q_1 \sqsubseteq Q_2$
EquivalentProperties($Q_1 \dots Q_n$)	$Q_1 \equiv \dots \equiv Q_n$
SameIndividual($o_1 \dots o_n$)	$o_1 = \dots o_n$
DifferentIndividuals($o_1 \dots o_n$)	$o_i \neq o_j, i \neq j$

Table 1. Axioms in OWL DL and *SHOIN*(D)

The W3C has developed two ontology languages for use on the Semantic Web. The first is RDFS [5], which was developed as a lightweight ontology language. The second language is OWL [6], which is a more expressive ontology language based on Description Logics [3].

OWL consists of three species, namely OWL Lite, OWL DL and OWL Full, which are intended to be layered according to increasing expressiveness. OWL Lite is a notational variant of the Description Logic *SHIF*(D); OWL DL is a notational variant of the Description logic *SHOIN*(D) [11]. It turns out that OWL DL adds very little in expressiveness to OWL Lite [12]. OWL Lite and OWL DL pose several restrictions on the use of RDF and redefine the semantics of the RDFS primitives; thus, OWL Lite and OWL DL are not properly layered on top of RDFS. The most expressive species of OWL, OWL Full, layers on top of both RDFS and OWL DL.

In this paper we are mainly concerned with the most well-known and most investigated species of OWL, namely OWL DL, which can be seen as an alternate notation for the Description Logic language *SHOIN*(D). In the remainder of this section we will explain OWL DL using Description Logic syntax. Table 1 shows axioms mapping between the OWL DL abstract syntax and the syntax of the Description Logic *SHOIN*(D).

A Description Logic knowledge base consists of two parts, namely the TBox and the ABox. The TBox consists of a number of class and property axioms; the ABox consists of a number of individual assertions (see Table 1). Here, C refers to a description, T refers to a concrete datatype; D refers to either a description or a datatype. R refers to an object property name, Q refers to an object or datatype property ; o and t refer to object and concrete values, respectively. A class axiom in the TBox consists of two class descriptions, separated with the GCI (General Class Inclusion, or subsumption; $\sqsubseteq$) symbol or the equivalence symbol ($\equiv$), which is equivalent to GCI in both direction (i.e. $\sqsubseteq$ and $\sqsupseteq$). Similarly, a property axiom consists of two property names, separated with the subsumption ($\sqsubseteq$) or the equivalence ($\equiv$) symbol. A description in the TBox is either a named class (A), an enumeration ($\{o_1, \dots o_n\}$), a property restriction ($\exists R.D, \forall R.D, \exists R.o, \geq nR, \leq nR$, analogously for datatype property restrictions), or an intersection ($C \sqcap D$), $union(C \sqcup D)$ or complement ($\neg C$) of such descriptions (Table 2). Individual assertions in the ABox can be individual (in)equality ($o_1 = o_2, o_1 \neq o_2$) assertions (Table 1).

However, OWL DL does not allow us to express default and exception knowledge. In next section, we propose OWL_d^e as an extension of OWL DL including default and exceptions which allows more complete definition of ontology concepts.

4 The OWL_d^e language

In this section, we present an extension of the OWL language by adding two constructors. These two constructors allow us a more representation flexibility of concepts with the default and exception knowledge. The RDF representations of "default" and "exception" constructors are the followings:

```
<rdfs:Class rdf:ID="default">
  <rdfs:label>default</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Class"/ >
</rdfs:Class>

<rdfs:Class rdf:ID="exception">
  <rdfs:label>exception</rdfs:label>
  <rdfs:subClassOf rdf:resource="#Class"/ >
</rdfs:Class>
```

In table 2 we show a mapping between the OWL_e^e abstract syntax and the syntax of the Description Logic $OntoDL_d^e$.

For example, we consider above definitions of the concepts E and we tray to give the corresponding representation in OWL_d^e:

$E = AB \sqcap A1_d \sqcap A2_d \sqcap A3^e$

The OWL_d^e representation should be:

OWL_d^e **Abstract Syntax**	$OntoDL_d^e$ **syntax**
A (URI Reference)	A
owl:Thing	$\top$
owl:Nothing	$\bot$
$intersectionOf(C, D)$	$C \sqcap \ldots \sqcap D$
$unionOf(C, D)$	$C \sqcup \ldots \sqcup D$
complementOf(C)	$\neg C$
$oneOf(o_1 ... o_n)$	$\{o_1 ... o_n\}$
restriction(P someValuesFrom(C))	$\exists P : C$
restriction(P allValuesFrom(C))	$\forall P : C$
restriction(P value(o))	$\exists P : o$
restriction(P minCardinality(n))	$\geq nP$
restriction(P maxCardinality(n))	$\leq nP$
$defaultOf(C)$	C_d
$exceptionOf(C)$	C^e

Table 2. Descriptions in OWL_d^e and $OntoDL_d^e$

```
<owl:Class rdf:ID='E'>
  <owl:intersectionOf rdf:parsetype='Collection'>
    <owl:Class rdfs:about='D'>
    <owl:default>
      <owl:Class rdfs:about='A1'>
    </owl:default>
    <owl:default>
      <owl:Class rdfs:about='A2'>
    </owl:default>
    <owl:exception>
      <owl:Class rdfs:about='A3'>
    </owl:exception>
  </owl:intersectionOf>
</owl:Class>
```

To make such syntax possible we need to extend the OWL DL grammar as well. The OWL_d^e grammar is presented in the next section.

4.1 OWL_d^e grammar

An OWL ontology in the abstract syntax contains a sequence of annotations, axioms, and facts. OWL ontologies can have a name. Annotations on OWL ontologies can be used to record authorship and other information associated with an ontology, including imports references to other ontologies. The main content of an OWL ontology is carried in its axioms and facts, which provide information about classes, properties, and individuals in the ontology. Below, we note $Annotation$ by $annot$:

$$\begin{aligned} &ontology \leftarrow' Ontology('[ontologyID]directive')' \\ &directive \leftarrow' Annot('ontologyPropertyIDontologyID')' \\ &\quad |'Annot('annotationPropertyIDURIreference')' \\ &\quad |'Annot('annotationPropertyIDdataLiteral')' \\ &\quad |'Annot('annotationPropertyIDindividual')' \\ &\quad |axiom \\ &\quad |fact \end{aligned}$$

In general, we divide OWL_d^e axioms into: Class Axioms, Descriptions, Restrictions and Property Axioms.

More than OWL-DL, descriptions in the OWL_d^e abstract syntax include class identifiers, restrictions default and exception. Descriptions can also be boolean combinations of other descriptions, and sets of individuals.

$$\begin{aligned} &description \leftarrow classID \\ &\quad |restriction \\ &\quad |'defaultOf('description')' \\ &\quad |'exceptionOf('description')' \\ &\quad |'unionOf('description')' \\ &\quad |'intersectionOf('description')' \\ &\quad |'complementOf('description')' \\ &\quad |'oneOf('individualID')' \end{aligned}$$

4.2 Structural Concept Algebra

First, we define **P** as the set of primitive concepts complemented with the set $\overline{\mathbf{P}}$ defined as follows:

$\overline{\mathbf{P}} = \{\overline{p}/p \in \mathbf{P}\}$

Let $\mathcal{C}$ be the structural concept algebra of $OntoDL_d^e$, we define the domain of $\mathcal{C}$ as follows:

$$\mathcal{C} = I_D^E,$$

where:

$$E_0 = 2^{\mathbf{P} \cup \overline{\mathbf{P}}} \times 2^{\emptyset}$$
$$E_{n+1} = 2^{\mathbf{P} \cup \overline{\mathbf{P}}} \times 2^{E_n}$$
$$E = \bigcup_{n \geq 0} E_n$$
$$D = 2^{\mathbf{P} \cup \overline{\mathbf{P}}}$$

I is the mapping function from $OntoDL_d^e$ into OWL_d^e language. Consequently, for an element A of $\mathcal{C}$, we note A_σ the set of strict definitions of the concept A, A_ϵ the set of exception definitions and A_δ the set of default definitions. We write:

$$I_{A_\delta}^{A_\sigma, A_\epsilon}$$

We present in table 3 the structural concept algebra $\mathcal{C}$ corresponding to OWL_d^e:

For example, we give structural denotations of concepts described in figure 1 as follows:

OWL_d^e **Abstract Syntax**	$\mathcal{C}$
owl:Thing	$I_{\emptyset}^{\emptyset,\emptyset}$
owl:Nothing	$I_{\emptyset}^{\emptyset,\emptyset}$
$intersectionOf(C, D)$	$I_{\{g(C_\delta,D_\epsilon)\backslash D_\sigma \cup g(D_\delta,C_\epsilon)\backslash C_\sigma\}}^{\{C_\sigma \cup D_\sigma\},\{C_\epsilon \cup D_\epsilon\}}$
$unionOf(C, D)$	$I_{\{g(C_\delta,D_\epsilon)\backslash D_\sigma \cup g(D_\delta,C_\epsilon)\backslash C_\sigma\}}^{\{C_\sigma \cup D_\sigma\},\{C_\epsilon \cup D_\epsilon\}}$
$complementOf(C)$	$I_{\emptyset}^{\{\overline{P}\},\emptyset}$
$oneOf(o_1...o_n)$	$I_{\emptyset}^{\{o_1...o_n\},\emptyset}$
$defaultOf(C)$	$I_{g(C_\delta,C_\epsilon)}^{\emptyset,\emptyset}$
$exceptionOf(C)$	$I_{\emptyset}^{\emptyset,C_\epsilon}$

Table 3. Descriptions in OWL_d^e and $OntoDL_d^e$

- The denotation of AB is $I_{\emptyset}^{\{A,B\},\emptyset}$
- The denotation of D is $I_{\{A1,A2,A3\}}^{\{A,B\},\emptyset}$
- The denotation of $E \cap D$ is $I_{g(\{A1,A2,A3\},\{A3\})}^{\{A,B\},\{A3\}}$
- The denotation of E will be $I_{\{A1,A2\}}^{\{A,B\},\{A3\}}$

The function g is used to cut out exceptions in defaults. We give more details about this function in the next section.

4.3 Substitution algorithm for OWL$_{\delta\epsilon}$

The principal of the function g is to get in entry two sets l and d. The set l contains a list of default definitions, while the set d contains the list of exceptions. For each concept a of the set d, we search if there exists a concept b from l such that $a = b$ and we replace the default definition by the exception.

```
$g : 2^D \times 2^D \rightarrow 2^D$ such that
$g(l, d) =$
    if $d = \emptyset$
      then return $l$
      else $res \leftarrow l$
           for all $a \in d$
            if there exists $b \in l$ such that $a = b$
              then $res \leftarrow g(res\backslash b, d\backslash a)$
           endfor
           return $res$.
```

In the next section we prove that g is of polynomial complexity

4.4 Computational complexity in OWL$_{\delta\epsilon}$

In this section, we study the computational complexity of the function g.

Proposition. The function g is of polynomial complexity.

Proof. Let n and m be the respective lengths of the two sets l and d. We know that the function g is a recursive function and that the lengths of the sets are decreasing for each new recursive call of the g. For each substitution, the length of the set d decreases $(m - 1)$, then the new research will continue on the rest of the set l without the substituted concept $(n - 1)$.

We note g^o the complexity of g, and g_i^o the complexity of the function g at the step i of the substitution.

$$\begin{aligned}
&g_1^o \leq n \times m \\
&g_2^o \leq (n-1) \times (m-1) \\
&\vdots \\
&g_n^o \leq (n-(n-1)) \times (m-(n-1)) \quad (with\; m > n-1) \\
&\quad \leq (1) \times (m-n+1) \\
&\quad \leq m-n+1
\end{aligned}$$

We know that:

$$g^o = g_1^o + g_2^o + \ldots + g_n^o$$

Then:

$$\begin{aligned}
g^o &\leq (n \times m) + ((n-1) \times (m-1)) + \ldots + (m-n+1) \\
&\leq (n \times m) \times n \quad (n\; times) \\
&\leq mn^2
\end{aligned}$$

5 Conclusion

In this paper we proposed OWL_d^e as an extension of the ontology Web language OWL by adding two new constructors *default* and *exception*. In order to make such extension possible, we proposed a specific description language $OntoDL_d^e$ which includes default and exception connectives. Then, we demonstrated who we can make mapping between the two languages and we presented the extension of grammar and the structural concept algebra. Finally, we proved that the proposed solution lays to a polynomial computational complexity.

We demonstrated in this paper the necessity to extend the Ontology Web Language with default and exception knowledge and the feasibility to do it based on description logic theory. We believe that such extension will solve most of the problem related nowadays to the Web semantic researches. However, the whole formalism is developed in a parallel work in our laboratory and includes as well the study of different cases of conflicts in definitions, especially for inherited concepts, and the complete algebra structure with theorems and proofs which are not presented in this paper. We are also focusing in the conception of a complete software including both language and graphical interface $OntoG_d^e$.

As a future work, we intend to study the problem of mapping between heterogenous ontologies by using $OntoDL_d^e$ as an intermediate language, the ontologies not based on default and exception knowledge will be completed by default values. Therefore, we can offer a complete and transparent mediation system between end users and heterogenous ontologies.

References

[1] G. Attardi, M. Simi. A Description-Oriented Logic for Building Knowledge Bases. In Proceedings of the IEEE, vol. 74, n. 10, 1335–1344, 1986.

[2] F. Baader, B. Hollunder. Embedding defaults into terminological knowledge representation formalisms. J. Auto. Reason. 14, 149-180, 1995.

[3] F. Baader, D. Calvanese, D. Mcguinness, D. Nardi, P. Patel Schneider. The description logic handbook. Cambridge (UK): Cambridge university press, 2003.

[4] R.J. Brachman. I Lied about the Trees Or, Defaults and Definitions in Knowledge Representation. The A.I. Magazine, vol.6, Number 3, 80–93, 1985.

[5] D. Brickley, R.V. Guha. RDF vocabulary description language 1.0: RDF schema. Recommendation 10 February 2004, W3C, 2004.

[6] M. Dean, G. Schreiber eds. OWL Web Ontology Language Reference. W3C Recommendation 10 February, 2004.

[7] F.M. Donini, D. Nardi, R. Rosati. Non-first-order features in concept languages. In Proceedings of the 4th Conference of the Italian Association for Artificial Intelligence (AI*IA95). Number 992 in Lecture Notes in Artificial Intelligence. Springer-Verlag, 91-102, 1995.

[8] F.M. Donini, D. Nardi, R. Rosati. Description Logics of Minimal Knowledge and Negation as Failure. In ACM Transactions on Computational Logic, vol. 3, num. 2, pages 177–225, 2002.

[9] P. Coupey, C. Fouquer. Extending Conceptual Definitions with Default Knowledge. In Computational Intelligence, Volume 13, Number 2, 258–299, 1997.

[10] T.R. Gruber. Towards principles for the design of ontologies used for knowledge sharing. In Roberto Poli Nicola Guarino, editor. International Workshop on Formal Ontology, Padova Italy, 1993.

[11] I. Horrocks, P.F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. In Proc. of the 2003 Int. Semantic Web Conf. ISWC, 2003.

[12] I. Horrocks, P.F. Patel-Schneider, F. Van Harmelen. From SHIQ and RDF to OWL: The making of a web ontology language. Journal of Web Semantics 1(1):7–26, 2003.

[13] R. Nado, R. Fikes. Semantically sound inheritance for a formally defined frame language with defaults. In Proceedings of the 6th National Conference on Artificial Intelligence (AAAI87). 443448, 1987.

[14] L. Padgham, T. Zhang. A terminological logic with defaults. In Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI93). 662-668, 1993.

[15] J. Quantz, V. Royer. A preference semantics for defaults in terminological logics. In Proceedings of the 3rd International Conference on the Principles of Knowledge Representation and Reasoning (KR92). Morgan Kaufmann, Los Altos, 294-305, 1992.

[16] U. Straccia. Default inheritance reasoning in hybrid KL-ONE-style logics. In Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI93). Morgan Kaufmann, Los Altos, Chambery (France), 676-681, 1993.

Using Ontologies to Represent Software Project Management Antipatterns

Dimitrios Settas, Ioannis Stamelos
Dept. of Informatics,
Aristotle University of Thessaloniki
Thessaloniki, Greece
{dsettas,stamelos}@csd.auth.gr

Abstract

In spite of numerous knowledge sharing and reuse mechanisms, the provision of intelligent advice to software project managers still remains an open issue. Antipatterns provide information on commonly occurring solutions to problems that generate negative consequences. These mechanisms are documented using informal structures that do not readily support knowledge sharing and reuse. For this, we need better-structured representations. The formalism of Bayesian Networks (BN's) has been proposed to capture and model software project management antipattern uncertainty manually or automatically, through an antipattern knowledge base. The antipattern ontology proposed in this paper, specifies the conceptual structure of the antipattern knowledge base, encodes tacit software project management knowledge into computer understandable form and will allow the sharing and reuse of this knowledge by software tools. Furthermore, the issue of capturing and quantifying uncertainty in the antipattern ontology is addressed by including the concepts of antipattern BN models and their corresponding OWL ontology in the design of the generic antipattern ontology.

Keywords: Antipattern Ontology, Bayesian Networks, Antipattern Knowledge Base

1. Introduction

Software project management antipatterns suggest commonly occurring solutions [1] to problems regarding dysfunctional behaviour of managers or pervasive management practices that inhibit software project success [2]. These mechanisms can manage all aspects of a software project more effectively by bringing insight into the causes, symptoms, consequences, and by providing successful repeatable solutions [3]. The readers that are not familiar with antipatterns can use [2],[3] as introductions to the topic.

Software project management antipattern catalogues [1],[2],[3],[4],[5] have been documented using informal templates and unofficial structures that attempt to make antipatterns easy to remember. Such structures do not readily support knowledge sharing and reuse because they can only be used among people. The amount of defined antipatterns and the amount of printed documentation is increasing to the extent that it becomes difficult for it to be effectively used. Furthermore, different templates have been proposed [2], [3], which can be used to document a software project management antipattern. As a result software project management using antipatterns has not become a common practice in the practitioners' community. For antipatterns to become a widespread practice, a better structured antipattern representation is required.

While there is a considerable amount of literature about ontologies, the issue of representing antipatterns using ontologies has not been addressed. Uncertainty concerns every aspect of ontologies [6],[7] and it is one of the most important criteria that need to be taken into account in the selection of an appropriate knowledge representation [8]. In this paper, the issue of quantifying uncertainty in the antipattern ontology is addressed by including the concept of antipattern Bayesian Network (BN) models [5] in the ontology itself. Furthermore, a corresponding OWL ontology of an antipattern BN model is included in the generic model of the antipattern ontology.

BNs have been recently suggested [5] for modeling software project management antipatterns. This formalism provides a natural, logical and probabilistic framework to depict software project management antipatterns and can be used by project managers to illustrate the effects of uncertainty on a project management antipattern. The antipattern knowledge base model [9] can overcome the problems associated with the manual construction of BNs by automatically constructing antipattern BN models. However, due to the problems associated with antipattern documentation, an ontology is required to provide the conceptual model of antipatterns. Furthermore, the antipattern ontology will benefit the antipattern knowledge base model [9] by providing it with a description of concepts, attributes of concepts, relationships among concepts, constraints on these relationships, defining therefore knowledge reference structure of the domain of software project management antipatterns. Following the official process of ontology construction [10], the antipattern ontology was first specified and then designed.

In particular this framework is exemplified by using ”The Standards” software project management antipattern [3]. This antipattern explains why standards fail to address the needs of many organizations and provides a refactored solution in order to resolve the unbalanced forces, causes, symptoms and consequences of this antipattern [3]. We chose this issue to illustrate the power of including BNs and their corresponding OWL ontology in the generic antipattern ontology in the context of antipattern knowledge base approach [9]. This paper is organized as follows: section 2 describes the background, related work and the literature review used in our research. Section 3 describes the specification and design of the antipattern ontology. Section 4 exemplifies the proposed ontology approach in the context of the antipattern knowledge base. Finally in section 5 findings are summarized and conclusions are drawn.

2. Background and Related Work

2.1 Background

By listing project management antipattern catalogues project managers can identify potential problems and provide a refactored solution in a practical and reusable manner. According to Brown et al. [1], a software project management antipattern can be the result of a manager not having sufficient knowledge or experience in solving a particular problem.

The antipattern knowledge-base [9] uses a many-to-many interaction of software managers and antipattern contributors through the antipattern knowledge base. The knowledge base provides the means to intelligently disseminate computer-based software project management antipatterns and can be used together with queries of software managers in order to create Bayesian Network (BN) models of software project management antipatterns [5] in an automated manner. This is achieved using the Knowledge based model construction (KBMC) framework [8], which is a combination of first-order logic and Bayesian networks.

According to Devedzic [11], one can draw an analogy between libraries of ontologies and catalogues of software patterns. Patterns and antipatterns are not ready-to-use building blocks as are ontologies and ontologies can also be seen as knowledge skeletons of a domain. However, ontologies are more general and common-sense oriented and less concrete than antipatterns. The most important difference between these two notions is that ontologies are encoded in computer understandable form and can be used by intelligent agents, while antipatterns can only be used among people. Joining these two notions will encode important software project management knowledge into a computer understandable form to allow the sharing and reuse of this knowledge by computer tools that implement intelligent agent technology [12]. Furthermore, an ontology can provide a framework for categorizing empirical studies and organizing them into a body of knowledge [13]. In this paper, this framework addresses the software project management antipattern domain. Thus, antipattern researchers will be able to provide a context within which specific questions about antipatterns can be investigated.

2.2 Related Work

Software patterns have been recently represented using ontologies [14], [15]. Antipatterns are the latest generation of design pattern research and are related with patterns in the sense that design patterns can evolve into antipatterns [1]. The difference is in the context: An antipattern is a pattern with inappropriate context and is particularly useful in the case of knowledge representation and knowledge management, because it captures experience and provides information on commonly occurring solutions to problems that generate negative consequences. An antipattern is a new form of pattern that has two solutions. The first is a solution with negative consequences and the other is a refactored solution, which describes how to change the antipattern into a healthy solution. These fundamental differences between patterns and antipatterns inhibit the use of the software pattern ontology [14],[15] in order to represent antipatterns. The activities in the research of antipatterns are mainly focused in the invention of new antipatterns.

As already mentioned it is important to address the issue of uncertainty in the selection of an appropriate knowledge representation. One of the most widely used standards that have emerged for web ontologies is OWL [6]. However, the current definition of OWL does not take uncertainty of knowledge into account [6],[7]. It has been proposed that in cases where OWL is inefficient in capturing uncertainty, special means should be used [6]. BayesOWL [7] is a framework which extends and supplements OWL for representing and reasoning with uncertainty based on BNs. This framework [7] provided a set of rules and procedures that directly translate an OWL ontology into a BN structure and a method that utilizes available probability constraints about classes and interclass relations in constructing the conditional probability tables (CPTs) of the BN.

The issue of uncertainty has also been addressed using other approaches, including directly embedding uncertainty information in the knowledge base using probabilistic description logics [16] and using Bayesian networks [17] but the reasoning support in tools for such extensions is lacking [6]. In this paper, the issue of capturing uncertainty in the antipattern ontology is addressed by including one or more BN models of an antipattern [5] in the antipattern ontology.

3 Specification and design of Antipattern Ontology

3.1. Specifying the Antipattern Ontology

In this paper, only a part of the ontology construction process is considered and ontologies are treated as more complete specifications of antipatterns. As already mentioned, project management antipatterns are specified using informal presentation styles of ([3], [2], [4]), For the specification of the software project management antipattern ontology Laplante's [2] template (see Table 1) attributes are used.

In the construction of an ontology, existing ontologies must be integrated. Since, an antipattern ontology does not exist, pattern ontologies can be partially integrated. As a result, following the recent specification of the pattern ontology [14], the antipattern ontology can be specified using the notion of antipattern relationships. In the specification of the antipattern ontology, antipatterns and their relationships must be included [14]. These are expressed according to a formalism which was originally proposed for software systems [18]:

- If the refactored solution of an antipattern A1 uses an antipattern A2, then A1 uses A2.
- If the explanation of an antipattern A1 is a specialization of an antipattern A2, then A1 refines A2.
- If the application of an antipattern A2 is demanded in the application of antipattern A1, then A1 demands A2.
- If an antipattern A1 and an antipattern A2 provide different refactored solutions for the same problem then, antipattern A1 is the alternative of A2.

Finally the antipattern ontology takes into account the three different kinds of antipatterns, which according to Brown *et al.* [1] are software development, software architecture and software project management antipatterns.

3.2. Designing the Antipattern Ontology

There exist different formalisms that can be used to express ontologies, such as the Knowledge Interchange Format and knowledge representation languages [19], [11].

Table 1. Antipattern attributes for the specification of the antipattern ontology.

Name	A short name that conveys the antipattern's meaning.
Central Concept	The short synopsis of the antipattern in order to make the antipattern identifiable.
Dysfunction	The problems with the current practice.
Explanation	The expanded explanation including causes and consequences.
Band-Aid	A short term coping strategy for those who don't have the influence nor time to refactor it.
Self-Repair	The first step for someone perpetuating the antipattern.
Refactoring	The required changes in order to remedy the situation and their rationale.
Identification	An assessment instrument consisting of a list of questions for diagnosis of the antipattern.

However, a generally accepted notation for representing ontologies has not been proposed yet [11]. The Unified Modelling Language notation has been generally accepted in object-oriented design because it provides a graphical notation that represents classes, objects and their relationships in different views. UML has also been proposed as a suitable notation to represent ontologies [6],[13],[20]. In this paper, UML was used to express and design the antipattern ontology. UML was chosen because several software tools have been developed that enable the representation of ontological knowledge using UML [20]. Furthermore, an eXtensible Stylesheet Language Transformation (XLST) based approach can be used on UML models in order to generate Web Ontology Language (OWL) [21]. Such tools are deployed because practitioners are already familiar with UML and there is often no need to learn how to use specific ontology tools [21].

In UML information is represented in class diagrams [20]. The use of UML has been proven successful in the construction of an ontology as a static model using a UML class diagram and an object diagram [19]. Class diagrams have also been used to describe the software maintenance process ontology [13]. A UML profile for OWL has been recently proposed in [6], but it is based on a dated version of OWL. An initiative to provide a standard mapping between UML and ontology languages such as OWL is in progress under the auspices of the Object Management Group (OMG). In the design of the antipattern ontology diagram (see Fig. 1), UML notation was used to describe the different relationships that an antipattern might have according to the specification of the antipattern ontology. Furthermore, the antipattern ontology includes the attributes of an antipattern according to the specification of the antipattern ontology. The design also addresses the different kinds of antipatterns and illustrates the relationship between the concepts of antipatterns, antipattern BN models and the OWL ontology that corresponds to an antipattern

BN model. This relationship indicates that an antipattern can be modelled with one or more Bayesian Network models. However each antipattern BN model only corresponds to a single OWL ontology.

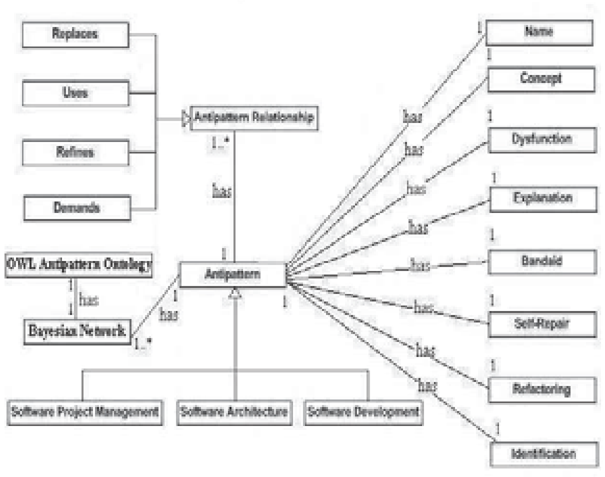


Figure 1. The Generic Antipattern Ontology UML Diagram

4 The antipattern knowledge base model

4.1. The use of Bayesian Networks in the Antipattern Ontology

In this section we exemplify the use of BNs in the antipattern ontology, which conceptually defines the antipattern knowledge base (See Fig. 2). The antipattern knowledge base system [9] aims towards the automatic construction of antipattern BN models and the intelligent dissemination of computer-based software project management antipatterns. The model uses a many-to-many interaction of software managers and antipattern contributors through the antipattern knowledge base. The main feature of this architecture is the collective knowledge, built by a combination of human work and machine learning. Another important feature is that the knowledge base receives real-world feedback on the quality and correctness of their contributions in the form of queries and their outcomes, and the resulting knowledge is therefore much more likely to be both relevant and correct [8].

The interaction of the antipattern contributors with the antipattern knowledge base includes:

- **Antipatterns.** Antipatterns will be expressed as rules and facts in the Horn clause subset of first-order logic, which is used to capture a broad-range of real-world knowledge.

The interaction of software managers with the antipattern knowledge base includes three kinds of information:

- **Queries.** Queries are predicates with open variables that will be provided by the user from a graphical user interface. Information concerning the query can also be captured from the outside system that the query is referred to. Finally a utility value will be associated with queries in order to indicate the value of the query answer to the software manager. This will encourage software managers, who are also contributors to provide high quality antipatterns.

- **Replies.** In order to generate replies, the open variables for which the query predicate is true are instantiated.

- **Feedback.** If the reply of the query satisfies a software manager then he/she can report positive feedback to the knowledge base. On the other hand if the reply is not satisfactory then the error can be reported. For example, if the query is ”Where on the World-Wide Web can I find an antipattern on heterogeneous pair-programming developer personalities?” and the answer is a URL, the software manager can visit the URL and provide feedback if the web-page was successfully found or not.

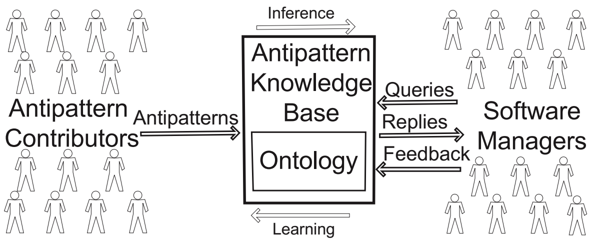


Figure 2. Antipattern knowledge base information streams

By including the BN model of an antipattern in the ontology proposed in this paper, each antipattern is associated with at least one corresponding BN model. The BN models that are included in the knowledge base can be either created manually or automatically using the KBMC framework by. To answer a query, KBMC extracts from the knowledge base the antipattern Bayesian network (See Fig. 3) containing the relevant knowledge. Every grounded predicate that is relevant to a specific query becomes a node in the Bayesian network model. Once this conversion takes place, any standard BN inference technique may be used to answer the software developers' queries [8]. The resulting BN model is proportional to the number of query rules and not to the size of the entire knowledge base. This is another advantage of using the KMBC framework.

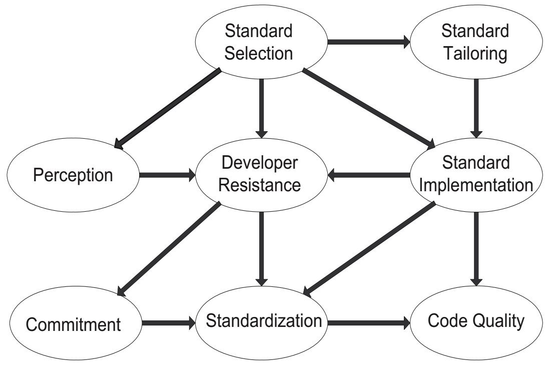


Figure 3. "The Standards" Antipattern BN model

Besides the power of probabilistic reasoning provided by BN itself, BN's are used in the antipattern ontology because of the structural similarity between the DAG of a BN and the RDF graph of OWL ontology: both of them are directed graphs, and direct correspondence exists between many nodes and arcs in the two graphs [17]. According to Pan *et el.* [7] a set of rules and procedures can be used for direct translation of an OWL ontology into a BN structure (a directed acyclic graph or DAG). However, the probabilities that are required in both translation and mapping can be obtained by using text classification programs, supported by associating to individual concepts relevant text exemplars retrieved from the web [7].

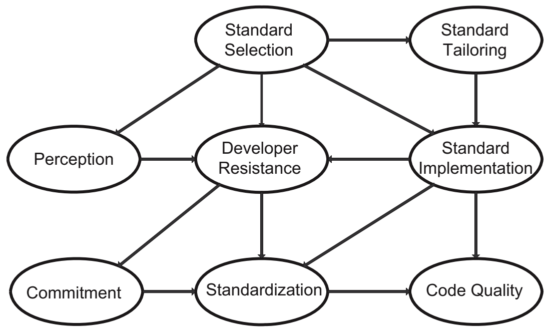


Figure 4. "The Standards" Antipattern corresponding OWL Ontology

In the approach proposed in this paper the same set of rules are used to translate an antipattern BN model (See Fig. 3) directly into a corresponding OWL ontology (See Fig. 4). The general principle underlying the structural translation rules is that all nodes in BN are translated to OWL classes [7]. The arrows represent the influence relation. In this graphical example (See Fig. 4) OWL concept properties and data types are not implemented since the BN model nodes (See Fig. 3) do not specifically define them.

5. Conclusion

In this paper the ontological representation of a software project management antipattern provided a source of precisely defined terms that can be communicated across people and software tools. A common project management ontology is an important step towards software tool interoperability because using the antipattern ontology, a specific antipattern can be much more easily used from different software project management tools. Furthermore, a knowledge-base approach using the antipattern ontology was proposed to support the technology of software project management antipatterns. This approach serves as a framework to allow the easy acquisition of software project management tacit knowledge using antipatterns. The antipattern knowledge base allows this knowledge to become explicit and provides the theoretical and technical foundation to successfully acquire and represent tacit knowledge encoded in antipatterns.

In particular this framework was exemplified by using "The Standards" software project management antipattern [3]. We chose this issue to illustrate the power of including BN models of antipatterns and their corresponding OWL ontology in the antipattern ontology. Further technical development of the internal knowledge base KBMC algorithm is required for further research and evaluation of the proposed framework. Furthermore, a richer set of data from empirical investigations would be more helpful in representing several project management antipatterns. Finally, a web-based community of software project management antipattern contributors must be created in an effort to develop, evaluate this environment and allow its on-line use by software project managers worldwide.

References

[1] William J. Brown, Raphael C. Malveau, Hays W. "Skip" McCormick III, Thomas J. Mowbray: AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. Wiley Computer publishing (1998)

[2] Philip A. Laplante, Colin J. Neil: Antipatterns: Identification, Refactoring, and Management. Taylor and Francis (2006)

[3] William J. Brown, Hays W. ”Skip” McCormick III, Scott W. Thomas: AntiPatterns in Project Management. Wiley Computer publishing (2000)

[4] Kuranuki Y., Hiranabe K.: Antipractices: AntiPatterns for XP Practices. Agile Development Conference (2004)

[5] Settas D., Bibi S., Sfetsos P., Stamelos I., Gerogiannis V.: Using Bayesian Belief Networks to Model Software Project Management Antipatterns. Proceedings of the 4th ACIS International Conference on Software Engineering Research, Management and Applications (SERA) (2006) 117–124

[6] David Taniar, Johanna Wenny Rahayu: Web Semantics and Ontology. Idea Group Publishing(2006)

[7] Pan, R., Ding, Z., Yu, Y, and Peng, Y.: A Bayesian Network Approach to Ontology Mapping. Proceedings of the Fourth International Semantic Web Conference(2005)

[8] Richardson M., Domingos P.: Building large knowledge bases by mass collaboration. Proc. of the 2nd Int. conf. on Knowledge capture. (2003) 129–137

[9] Settas D., Bibi S., Sfetsos P., Stamelos I., Gerogiannis V.: A Computer Supported Bayesian Network Approach to Model Software Project Management Antipatterns. Submitted to Springer LNCS.

[10] Fridman, N. N., McGuinnesS, D. L .: Ontology Development 101: A Guide to Creating Your First Ontology. Knowledge Systems Laboratory (2001)

[11] Vladan Devedzic: Understanding Ontological Engineering. Communications of the ACM **45**(4ve) (2002) 136–144

[12] Vladan Devedzic: Ontologies: Borrowing from Software Patterns.Intelligence **10**(3) (1999) 14–24

[13] Kitchenham, B. A., Travassos, G. H., von Mayrhauser, A., Niessink, F., Schneidewind, N. F., Singer, J., Takada, S., Vehvilainen, R., and Yang, H.: Towards an ontology of software maintenance. Journal of Software Maintenance **11**(6) (1999) 365-389

[14] Rosario Girardi, Alisson Neres Lindoso: An Ontology-based Knowledge base for the Representation and Reuse of Software Patterns. ACM SIGSOF Software Engineering Notes. **31** (1) (2006)

[15] Jean-Marc Rosengard and Marian F. Ursu: Ontological Representations of Software Patterns. Proceedings of KES04. Lecture Notes in Computer Science **3215** (2004) 31–38

[16] Guigno,R., Lukasiewicz, T.: A probabilistic extension of SHOQ(D) for probabilistic ontologies in the Semantic Web. Proceedings of the 2002 European Conference on Logics in Artificial Intelligence (JELIA). Springer-Verlag. (2002) 86–97.

[17] Ding, Z., Peng, Y.Pan, R.: A Bayesian approach to uncertainty modelling in OWL ontology. Proceedings of the 2004 Int. Conference on Advances in Intelligent Systems (2004)

[18] Agns Conte,Mounia Fredj, Ibtissem Hassine, Jean-Pierre Giraudin, and Dominique Rieu:A Tool and a Formalism to Design and Apply Patterns. OOIS 2002 Springer LNCS **2425** (2002) 135–146

[19] Cranefield S., Purvis M.: UML as an Ontology Modelling Language. Proc. of the Workshop on Intelligent Information Integration, 16th Int. Joint Conference on AI (1999)

[20] Kogut, P. A., Cranefield, S., Hart, L., Dutra, M., Baclawski, K. and Kokar, M. M. and Smith, J. E.: UML for Ontology Development. The Knowledge Engineering Review **17**(1) (2002) 61–64

[21] Gasevic D., Djuric D., Devedzic V., Damjanovic V., From UML to Ready-To-Use OWL Ontologies, 2nd IEEE Int. Conference on Intelligent Systems (2004) 485–490

Service Composition Using Planning and Case-Based Reasoning*

Kuan-Hsian Huang and Alan Liu
Department of Electrical Engineering
Center for Telecommunication Research
National Chung Cheng University
Chiayi, 621, Taiwan
aliu@ee.ccu.edu.tw

Abstract

Planning has been commonly applied to Web service composition recently. However, most of automated systems of Web service composition contain two problems. First, most of them overlook some user needs which sometimes combine services provided by systems themselves and services from external systems to provide a much more flexible service model. Second, most of them do not record information about service providers having already served the users and about plans having already been processed in order to speed up the pace and facilitation of systems providing services. Therefore, this paper presents a method of merging internal and external service systems to reach users' needs. The internal service resides in the local system, and the external one means a Web service provided by external service providers. We apply techniques of planning to combine both types of services, so that we can create plans made of a series of operations to satisfy user needs. We also apply Case-Based Reasoning to store plans and related information into a case base, so that it creates plans in much faster way when users have similar needs.

1 Introduction

Service composition is a challenging research topic [1]. To solve such problem, different approaches have been proposed, and there are a few reported systems, such as MIND [2] and Pistor [3], which use planning from artificial intelligence. Our work is to add a feature of user satisfaction in service composition using planning and also provide a mechanism which remembers the usage of previous service request. For this reason, we have designed a system which uses planning, namely Hierarchical Task Network Planning (HTNP) [2], in service composition along with Case-Based Reasoning (CBR) [4] for reusing old experiences.

* This work was supported in part by Ministry of Economic Affairs (Taiwan) under the grant 95-EC-17-A-02-S1-029 and National Science Council (Taiwan) under the grants 95-2752-E-008-002-PAE and 94-2213-E-194-010.

Our prototype system described in this paper uses intension-aware goal models [5] and the concept of personal ontology [6] to provide a service which is most suitable to the user. By using CBR, the system remembers how the user has been satisfied by certain services in order to reuse the previous experiences in future usage.

The work reported in [2] uses HTNP as the base in deriving services. The OWL-S files provided by service providers are converted to the domain in the tool, SHOP2, and the service request entered by the user will be used in planning with the domain information. Finally, a description file resulted from the service composition in OWL-S is produced. In addition to using HTNP for service composition, the authors in [7] provide an algorithm, called Enquirer, in the query manger to obtain information. The advantage is that it produces a reasonable result in the initial stages when information is still lacking. An approach in planning as model checking is used in [3]. The authors use MBP Planner [8] for solving nondeterministic and partial observable problems along with problems associated with extended goals. CBR is used in [9] for service composition, and the authors use six different kinds of relationship among services. The service name along with its service description is used for retrieving a case for providing a solution.

For the discussion above, Table 1 summarizes the related work with features and criteria, such as the main methods used; the capability in handling unexpected situation, in achieving a goal, and in recording what service providers had been used; and the ability in performing composition among internal services and external services. All systems have the ability in achieving their goals from the known providers, but only some of them can handle unexpected situations. Since none of the systems learns the providers that they have used, every service request is processed from scratch. In addition, none of the systems feature the ability in composing internal and external services together. If a system has its internal services, it can use such services in place of external services. This will save resources in searching and transmission. In addition, such internal services may also play the roles of substituting external services in case of failure.

Table 1. Comparison

Work	Main method	Unexpected situation	Goal achievement	Learning	External and internal services
[2]	HTN Planning	No	Yes	No	No
[7]	Planning as Model Check	Yes	Yes	No	No
[8]	HTN Planning	Yes	Yes	No	No
[9]	CBR	No	Yes	No	No

Table 2. Four methods for obtaining external services

Action	Precondition	Status change
Select_Service	Service (S) Select (S)	Haveservicedata (S) ¬ Select (S)
Check_Service_Data	Haveservicedata (S) ¬Have_main_service_data Main_goal (S)	Have_main_service_data
Check_all_Condition	Sort_Condition (S) (C) Filter_Condition (S) (C) Haveservicedata (S)	Check_Ready ¬ Sort_Condition (S) (C) ¬ Filter_Condition (S) (C)
Show_Data	Check_Ready Have_main_service_data	Finish

Our research is to use HTN planning with CBR for providing the user services. For the first time user, the system uses HTN planning to compose a service by exploring external and internal services for a possible combination. At the same time, the usage is learned by the system through the CBR mechanism. If the user issues a similar service request in the future, the CBR mechanism will find a composite service from the previous experience. In the next section, we explain our method which is a combination of a planning method and a CBR mechanism. Section 3 describes the process of implementation for our prototype system. The last section states the conclusion.

2 Combined Method

One feature of our system is to make it possible to combine the internal services originated at the user's local system and with the external services from other service providers. Another aim is to record the usage of a particular user in order to deliver a composite service without reorganizing it from scratch if the service request has been sent previously. Two AI techniques used in our system, planning and CBR, are discussed here in this section. Our method of combining planning and CBR may be considered as an extension to the work in [11]. If the service request cannot be solved through CBR, the planner takes over and find a composite service for the user. If the solution can be found using the CBR system, then the user has the choice of reusing the solution or to recomputing a new solution.

2.1 HTN Planning

In using a planning mechanism, more information will be required other than a single service request from a user. For this reason, we have studied methods for expanding keywords for providing more information for a planner. From our previous research in analyzing user intension and ontology, a system transforms the user query into a goal model which consists of the following attributes [5]:

1. Actions: user's intended action
2. Objects: user's preferred type of service
3. Constraints: user's constraints toward a service
4. Parameters of objects: user's preference toward a service

When using an external or internal service, we sometimes need to face unexpected situations like time out or execution failure during the execution of a plan. In order to solve nondeterministic problems like this, a monitoring mechanism is added to our planning mechanism. In using HTN planning, we insert such monitoring mechansim to actions with potential failures.

In this paper, we use P = (T, S) as our definition of planning, where P is a plan generated, T is a set of tasks, and S is the initial status of a plan problem. The planner uses S and T to come up with a plan. In the task network, T, t' represents a sub-task. A sub-task may be a method or an operator, so if t' represents a method, then t' needs to be decomposed further until t' becomes an operator. An operator represents the basic action in this paper, and the basic action cannot be divided. Thus, decomposition stops when t' is an operator.

A subtask, t_1, may be viewed as four methods, such as Select_Service, Check_Service_Data, Check_all_Condition, and Show_Data. Table 2 summarizes the precondition for those actions and the stauts changes afterward. In Table 2 and Figure 1, S represents a service provider and C represents the parameter for filtering information after a service.

In Figure 1, we represent Table 2 as a state chart, in which we can see that what preconditions need to be satisfied before executing a method and what status change a method brings after execution. In the figure, we see that to execute Show_data, we need to satisfy two statuses, Check_ready and Have_main_service_data. These two statuses rely on two methods, Check_all_condition and Check_service_data. These two methods instead rely on Select_service.

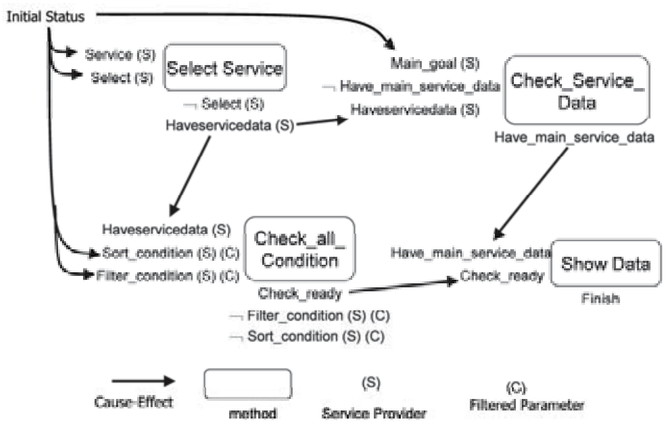


Figure 1. Relationship between four methods

Those four methods in Table 2 can be further divided into other methods or operators. Taking the method, Select_Service as example, we can divide it into two types, the primary service and the secondary service. These two types are then defined as Main_Goal and Seond_Gaol, respectively, and they have different preconditions. This is depicted in Figure 2.

As for the method, Check_all_Condition, the main purpose is to check whether or not sorting or filtering status still exists in the service. If there is a service still needing sorting or filtering status, then Check_all_Condition will clean the status after applying sorting or filtering and produces Check_Ready status.

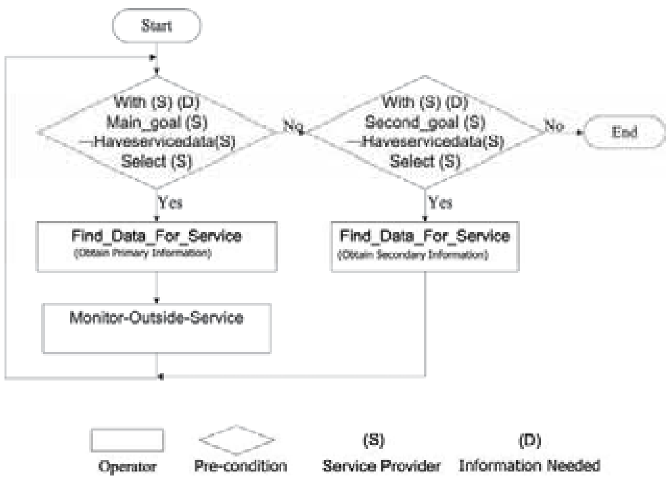


Figure 2. Flow of select_service

2.2 Process with CBR

A case is divided into two parts, the description part and the solution part. We use three attributes, action, object, and constraints, to define the description part of a case. As for the solution part, it includes the locations of the OWL-S files by service providers, the parameters needed for the service providers, and the plans.

At the case retrieval stage, we use the goal model provided as the features for finding a case. At the case reuse stage, we check the content of the goal model and the retrieved case. If the content is an exact match, then we can just simply copy the solution part as a composite service for the user. If there is still a difference, an adaptation mechanism will modify the solution part accordingly.

As for the prototype reported here, we have not yet come up with a satisfactory adaptation method, so we simply use substitution for the adaptation part. For that we find the data of Constraints and Parameters of objects for comparison and apply If-Then rules to modify the content. Currently, we have two categories of rules, oone based on constraints and the other based on objects. The following are some of the rules:

A. Rules based on constraints (18 rules)

- **IF** Case_Constrain ='Before Date' **AND** New Constrains='After Date'
  Then Update Case_Constrains_Data
- **IF** Case_Constrain ='Before Date' **AND** New Constrains='About Date'
  Then Update Case_Constrains_Data
- **IF** Case_Constrain ='After Date' **AND** New Constrains='Before Date'
  Then Update Case_Constrains_Data
- ...

B. Rules based on Parameters of objects (8 rules)

- **IF** Case_Date != New_Date
  THEN Case_Date = New_Date

- **IF** Case_DepartureTime != New_DepartureTime
  THEN Case_DepartureTime = New_DepartureTime
- **IF** Case_ ArrivalTime != New_ ArrivalTime
  THEN Case_ ArrivalTime= New_ ArrivalTime
- ...

When all services in a plan generated by the HTN planner are completed, we can apply a QoS mehods [10] to evaluate the result. If the result is good, then this successful case will be stored in the case base.

3 System Implementation

We use a domain independent planning tool, JSHOP2 [12], as our planner. For defining a planning problem and planning domain we use definitions as follows:

`(defproblem problem-name domain-name` `([` $a_1 a_2 \cdots a_n$ `]) T )`, where problem-name is given by the user and domain-name are picked from the planning domain. a_1, a_2 and to a_n are the initial states, and T represents the tasks which can satisfy the initial states.

`(defdomain domain-name (` $d_1 d_2 \cdots d_n$ `))`, where domain-name is given by the user, and d_1, d_2 all the way up to d_n represent operations and methods.

JSHOP2 transforms the planning problem and planning domain into Java code, and plans are delivered after executing the Java code. Planning problems vary according to different users. We also use OWL-S API [13] in developing our system. With this API, we can execute an OWL-S file containing atomic process in WSDL grounding or use sequence, unordered, and split to control composite process in OWL-S.

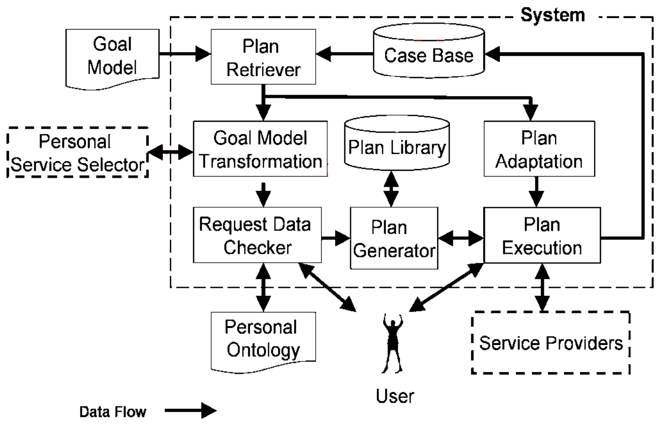


Figure 3. System architecture

Figure 3 shows the system architecture consisting of service providers and personal service selector in the outer part with goal models and personal ontology coming through external interface. In this prototype, plan execution is a simple function which is assumed to execute the resulting plans. Here we describe the six components depicted in Figure 3:

Plan Retriever (PR): This component retrieves a case for a plan if there is a case to be found. It then transfers goal models to the Plan Adaptation component. If there are multiple similar cases, the system displays the choices for the user. However, if no case is applicable, then the goal model will be passed to the Goal Model Transformer.

Goal Model Transformer (GMT): It uses the Action and Object fields in a goal model to analyze what type of a provider a user is seeking. If there is a need in using external services, the parameters related to input and output will be given to the personal service selector.

Request Data Checker (RDC): Its main purpose is to check whether the information for executing a service is complete or not. If it lacks some information, then personal ontology is first checked. If there is more information needed, then the user will be requested to enter some information.

Plan Generator (PG): It uses JSHOP2 to generate plans according to the problem domain previously defined and the data provided by the RDC.

Plan Adaptation (PA): It checks the goal model and the case retrieved in the case base for their differences. If there is a difference, then there is a set of rules to modify the solution part.

Plan Execution (PE): This is to verify how well plan can be executed. When a plan is executed, then an evaluation will be carried out. If the plan is satisfactory, the plan will be stored as a case in the case base.

The sequence of system execution is shown in Figure 4. The dashed rectangle represents the outer system, and a regular rectangle represents a particular function. Since there are many functions involved in all process, the functions are organized as four objects and represented as ovals. When the system receives the goal model from the external system, it executes plan retrieval and this will cause a search in a case base. If there is a match, then the goal model is transferred to the Data Gatherer (DG).

After the goal model enters the DG, it will utilize the GMT and the RDC. This sequence is depicted in Figure 5. In the GMT, the goal model is analyzed to determine what types of services the user prefers. If the system does not have enough internal services for the user, then the system uses the Personal Service Selector to pick a suitable service. After receiving a suitable service, the RDC is performed. The RDC first checks to see if enough information is supplied for the service. If the information is not sufficient for the RDC, the user or the personal ontology is consulted. The user preferences will be passed to the PG by the RDC.

The DG delivers service information and planning results to the PG, where Select_Plan_Generate and Order_plan_Generate are performed, as shown in Figure 6. This process is to set up the plan generator and to produce suitable plans to execute. Figure 7 shows the sequence diagram of the CBR Planner, which compares the goal model with the previous cases to decide whether to perform the adaptation process. The similarity measurement is basically to check the parameters of objects and constraints. If both sections are matched, then no modification is necessary. If there is no exact match, then the rules are triggered to modify the solution part of the case retrieved.

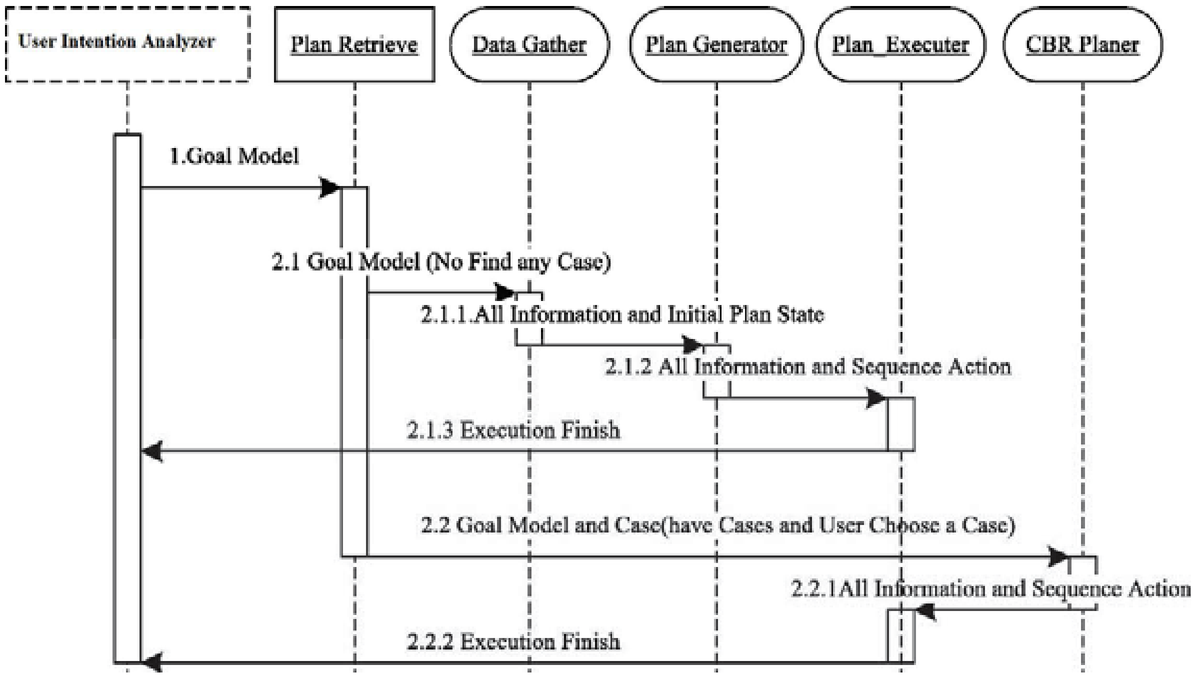

User Intention Analyzer
Plan Retrieve
Data Gather
Plan Generator
Plan_Executer
CBR Planer
1.Goal Model
2.1 Goal Model (No Find any Case)
2.1.1.All Information and Initial Plan State
2.1.2 All Information and Sequence Action
2.1.3 Execution Finish
2.2 Goal Model and Case(have Cases and User Choose a Case)
2.2.1All Information and Sequence Action
2.2.2 Execution Finish

Figure 4. Sequence diagram of our system

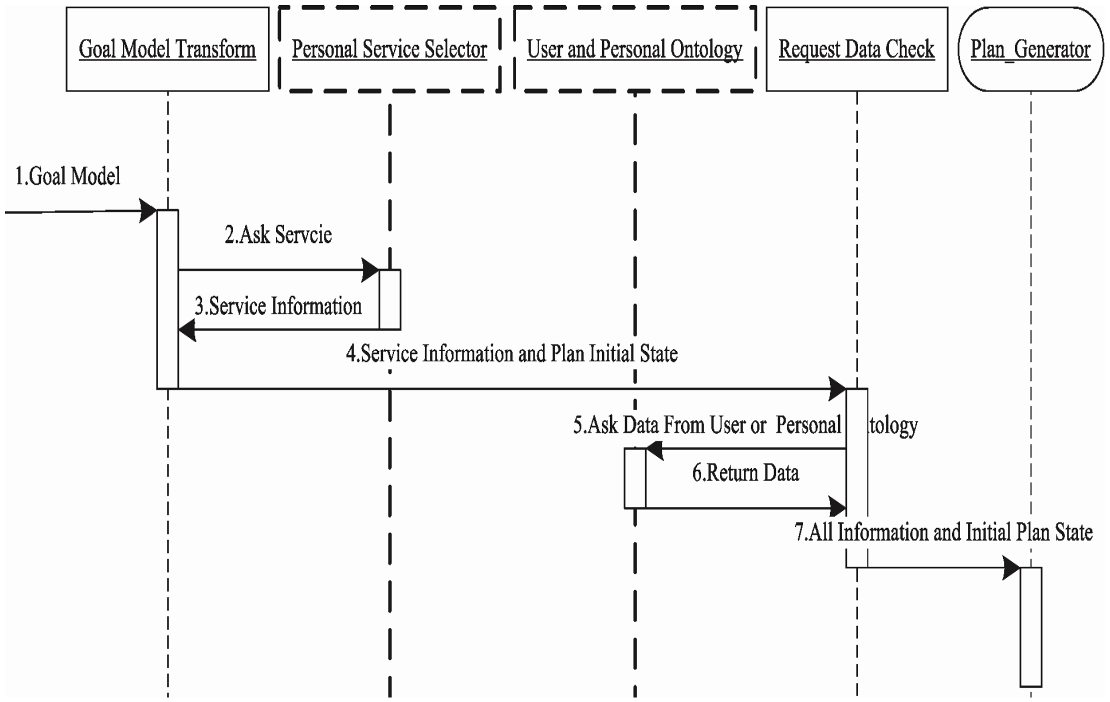

Goal Model Transform
Personal Service Selector
User and Personal Ontology
Request Data Check
Plan_Generator
1.Goal Model
2.Ask Servcie
3.Service Information
4.Service Information and Plan Initial State
5.Ask Data From User or Personal tology
6.Return Data
7.All Information and Initial Plan State

Figure 5. Sequence diagram of data gatherer

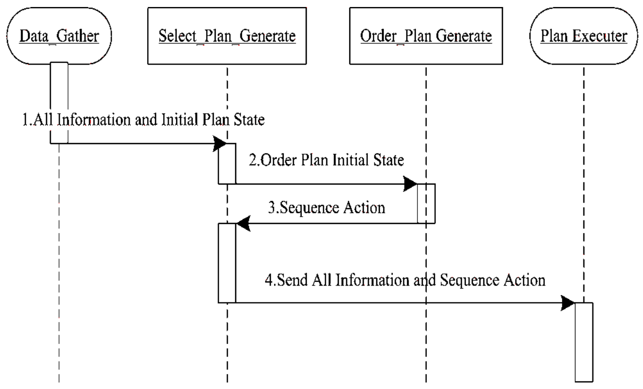

Data_Gather
Select_Plan_Generate
Order_Plan Generate
Plan Executer
1.All Information and Initial Plan State
2.Order Plan Initial State
3.Sequence Action
4.Send All Information and Sequence Action

Figure 6. Plan generator

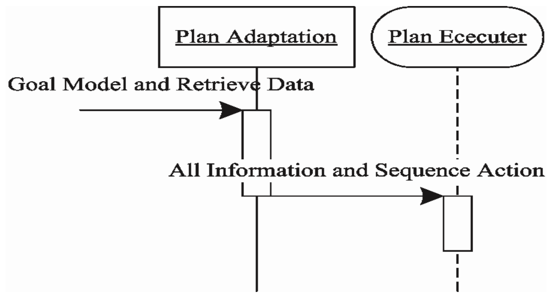

Plan Adaptation
Plan Ececuter
Goal Model and Retrieve Data
All Information and Sequence Action

Figure 7. CBR planner

4 Conclusion

Based on the observation made for Table 1, our aim was to introduce a system which has a learning capability. In addition, the system is designed to be able to perform service composition with internal and external services. We built a prototype system which accepts a service request from a user through the intention analysis system which produces a goal model by extending the service request with more keywords representing the intention. We used simulated Web services for traveling services including airline tickets and other means of services for executing our prototype. The result was satisfactory by using a planner for building a composite service from scratch and also reusing experiences through the CBR system. Our observation is that the planner gives flexibility in adding or deleting services. The CBR system was also effective in providing a composite service quickly.

References

[1] M. Nikola, M. Miroslaw, "Current Solutions for Web Service Composition," Proc. of IEEE Internet Computing Online, Vol. 8, no. 6, 2004, pp. 51-59.

[2] E. Sirin, B. Parsia, D. Wu, J. Hendler and, D. Nau, "HTN Planning for Web Service Composition Using SHOP2," Journal of. Web Semantics, vol. l, no. 4, 2004, pp.377–396.

[3] M. Pistore, F. Barbon, P. Bertoli, D. Shaparau, and P. Traverso, "Planning and monitoring web service composition," Proc. of International Conference on Artificial Intelligence, ethodologies, Systems, and Applications (AIMSA), 2004, pp.106-115.

[4] A. Aamodt, E. Plaza, "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches," AI Communications. Vol. 7 no 1, 1994, pp. 39-59.

[5] C.H.L. Lee and A. Liu, "Toward Intention Aware Semantic Web Service Systems," Proceedings of the IEEE International Conference on Services Computing (SCC 2005), 2005, pp.69-76.

[6] M.N.Huhns and L.M.Stephens, "Personal Ontologies," IEEE Internet Computer, 1999, pp. 85-87.

[7] M. Pistore and P. Traverso, "Planning as Model Checking for Extended Goals in Non-deterministic Domains," Proc. of 7th. IJCAI'01. AAAI Press, August 2001.

[8] P. Bertoli, A. Cimatti, M. Pistore, M. Roveri, and P. Traverso, "MBP: a Model Based Planner," Proc. of the IJCAI'01 Workshop on Planning under Uncertainty and Incomplete Information, Seattle, August 2001, pp.93-97.

[9] B. Limthanmaphon and Y. Zhang, "Web Service Composition with Case-Based Reasoning," Proc. of the Fourteenth Australasian database conference on Database technologies 2003, pp.201-208.

[10] S. Kalepu, S. Krishnaswamy, and S. W. Loke, "Verity：A QoS Metric for Selecting Web Services and Providers," Proc. of the 4th Intl Conf. on Web Information Systems Engineering Workshops (WISEW'03), Dec. 2003, pp. 131–139.

[11] H. Munoz-Avila, et. , "SiN: Integrating Case-based Reasoning with Task Decomposition," Proc. of Seventeenth International Joint Conference on Artificial Intelligence, 2001.

[12] O. Ilghami and D. S. Nau, "A general approach to synthesize problem-specific planners," Tech Report CS-TR-4597, UMIACS-TR-20060, University of Maryland, 2003.

[13] E. Sirin et al., OWL-S API, http://www.mindswap.org/2004/owl-s/api/index.shtml , 2004.

MDA-based Ontology Development: A Study Case

Eluzaí Souza dos Santos, Célia Ghedini Ralha, Hervaldo Sampaio Carvalho
Departamento de Ciência da Computação, Instituto de Ciências Exatas, Universidade de Brasília
Faculdade de Medicina, Universidade de Brasília
Campus Universitário Darcy Ribeiro, Caixa Postal 4466, Brasília, Cep 70.910-900, Brasil
eluzai@cic.unb.br, ghedini@cic.unb.br, carvalho@unb.br

Dragan Gašević
School of Computing and Information Systems, Athabasca University
1 University Drive, Athabasca, AB T9S 3A3, Canada
dgasevic@acm.org

Abstract

In this paper, we present an experience to enable the integration of computational systems using the eXtensible Stylesheet Language Transformation (XSLT)-based approach for automatic generation of the Web Ontology Language (OWL) from a UML model developed at the medical domain. We discuss similar approaches to automatic transformation between a metamodel for OWL based on the Meta-Object Facility (MOF) and an associated UML profile. Although MDA-based techniques are promising for ontology development, there has not been practical study cases to demonstrate how this technology can be used in real scenarios. In our efforts to support the model-driven development of OWL ontologies from its Ontology UML Profile (OUP), we illustrate the axiomatic verification of the OUP-developed ontology extension with the use of Racer reasoner and Protégé.

1 Introduction

Generally speaking, the Semantic Web initiative tries to establish better semantic connections between different resources on the Web. These resources can be specified by ontologies. Ontologies are responsible to capture the semantics of a domain by deploying knowledge representation primitives, enabling a machine to partially understand the relationship between concepts in a domain [17].

The standardization of the Web Ontology Language (OWL) by the World Wide Web Consortium (W3C) contributed heavily to the wide-spread use of ontologies [11]. In 2003, the Object Management Group (OMG), a standardization consortium for various aspects of software engineering including the well-established Unified Modeling Language (UML) [23], replied to this by issuing a Request for Proposal (RFP) for an Ontology Definition Metamodel (ODM) [21]. There were some submitted proposals to the OMG. The final draft was sent last June and they are now waiting for its adoption [24].

The intention of the RFP was to provide a MOF [22] metamodel which allows for processing ontologies in the same technical space where other modeling technologies are defined in (e.g., UML, CWM, or any other Domain-Specific Modeling Languages - DSMLs). The main benefit of this approach is that one can use the same tools for processing ODM, UML, or any DSMLs that can be transformed by using the same model-to-model transformation language (i.e., QVT), stored by the same XML approach (i.e., XMI) and managed by the same APIs (e.g., JMI) [26].

In this paper, we present a study case using the XSLT-based approach for automatic generation of the OWL ontology from a UML model applied to the medical domain, according to the approach presented in [15]. The work presented here is motivated by the fact that once this MDA-based technology is available, what is missing are real use-cases that will demonstrate how the technology can be used. Moreover, the previous work did not have a strong verification of ontologies by using ontology reasoners such as Racer [1], which may lead to developing ontologies that are, for example, inconsistent.

2 Background Work

Before presenting the used approach to MDA-based automatic OWL ontology development, we summarize concepts related to the OMG's Model Driven Architecture (MDA) [8, 12, 25] and its MOF model [22], the Ontology

Driven Architecture (ODA) [27] and the ODM [21], which can be used together to maximize and take advantage of an hybrid approach.

This work is based on the MDA proposal defined by OMG. MDA is considered a conceptual framework, formed by a four-layer architecture. Its objective is to describe the basic requirements that should be developed to maximize the reuse, portability and interoperability, as well as to clarify the relationship among them to offer support to the code generation [14, 12].

The MDA architecture is formed by the Meta-metamodel (MOF)-M3 layer. MOF defines an abstract language and a framework to specify, build and manage independent technology of metamodels. All the metamodels and patterns, defined by the MOF are located in the Metamodel layer-M2 (e.g., the place where UML is defined). The Model layer-M1 contains the models of the real world, represented by defined concepts in a corresponding metamodel of the M2 layer (e.g. UML metamodel). Finally, in the Instance layer-M0 there are objects of the real world. The patterns proposed by MDA of OMG make possible the metadata administration and integration. Among them, we highlighted the XML Metadata Interchange (XMI). XMI is a pattern that maps MOF to XML. In that way, there are defined XML tags used to represent the MOF metamodel in XML.

While MDA provides a powerful framework for Software Engineering and systems, the Semantic Web technologies provide a natural extension of this framework through the use of ontologies, bounding the semantic paradigm to MDA model originating the ODA. Thus, ODA is a complement to MDA since an ontology is an explicit conceptual model, with semantics based on formal logic, where the description of components can be consulted. Also, components that are indirectly necessary can be pre-loaded or they can be verified to avoid inconsistency in the system's configuration; thus, much of the development is done at execution time [27]. In this way, ODA maintains the original flexibility of the configuration and execution of the application server, but it adds new capacities for the system's developer and user.

3 Related Work

Considering the ODM model and the OMG's RFP, we may briefly describe some previous work related to different research groups involved in the efforts to define a metamodeling architecture for ontology development. Cranefield was the first researcher to propose the use of UML as a graphic language to represent ontologies [10]. According to his proposal, a domain expert designs an ontology graphically, that is exported by the UML tool in the XMI format. The XMI file goes by a XSLT transformation which can result in an ontology at the RDFS language and Java classes.

[7] propose an ODM for OWL DL language based on MOF. The metamodel uses Object Constraint Language (OCL) to define restrictions on the models, that are codified visually using the profile UML proposed. However, the authors did not present a practical implementation that uses the ODM and profile proposed.

[6] present a metamodel for OWL DL and OWL Full through the use of an UML profile based on MOF. In this work, we find the presentation of two tools that implement ODM submitted to OMG: the Visual Ontology Modeler (VOM) [2] and Integrated Ontology Development Toolkit (IODT) [3]. VOM is a tool developed by Sandpiper enterprise and actually is implemented as a plugin to IBM/Rational Rose. According to [6] VOM is compatible with the metamodels ODM and the profile for RDFS/OWL proposed by them. IODT is a toolkit for ontology-driven development, an Eclipse-based ontology-engineering environment, and an OWL ontology repository that supports RDFS/OWL parsing and serialization, transformation between RDFS/OWL and other data-modeling languages.

Gašević [15] presents an ODM proposal that supports OWL DL. In their work they propose the definition of different specifications, in agreement with the requirements of ODM: Ontology UML Profile and two mapping forms - OWL and ODM, ODM and Ontology UML Profile, and from Ontology UML Profile to other profiles. A better view of this proposal will be presented in Section 4.

4 The Used Approach

While ODM is not concluded, some proposals present alternatives to implement part of the mappings for the RFP. According to [12], to use the graphic modeling capacities of UML, the ODM should use a UML profile. This is a concept used to adapt the basic builders of UML for some specific purpose; in other words, it means to introduce new types of modeling elements. Although, in [6] we also find the UML profile based on MOF approach, in the article there is just a description of tools (e.g. VOM) and we could not test their UML profile in the VOM tool.

Figure 1 illustrates how the OUP transformation work according to [14]. In [16] we found a good explanation of OUP: a profile that allows the graphic edition of ontologies using diagrams UML, as well as other means that come from the traditional use of current CASE tools for UML modeling. In this paper, we have used this approach for XSLT transformations of OUP-based ontologies into the OWL language recommended by ODM.

XSLT is a W3C recommendation that allows interoperability of information. In XSLT, the information is seen as a tree of abstract nodes. The declarative nature of the leaves demarcation of style turns XSLT more accessible to non computer specialists users. XSLT includes builders that

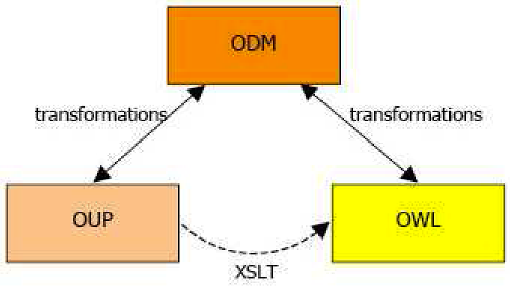


Figure 1. Used approach for transformations.

can be used to identify and to interact with the structures found in our source of information. The information that is being transformed can travel in any order, how many times are necessary, to produce the wanted result [18].

In [14], an ODM architecture is presented. In operational terms, the XMI exported file from the UML CASE tool serves as input for an XSLT processor that produces a document in the OWL format. This OWL file can be imported in an ontology development tool, where the ontology can be further worked, extended or refined. This approach will be illustrated in section 5 and the study case workflow can be understood through Figure 2. In previous publications [16, 12], authors used the Wine Ontology to test the solution, since this example ontology is distributed with Protégé environment [4].

5 The Study Case

Studying domain models, we noticed that they contain enough information to be the base of an ontology, being an artifact with central role during the whole software development. Therefore, the focus of the study is the transformation of MDA languages for OWL [10], so that the resulting ontology can be used for practical implementation in computational systems. For execution of this study case we used Poseidon for UML Community Edition 4.2 as CASE tool and Protégé 3.1.1 as ontology tool. As an ontology development methodology, we used the 101 Method [20], described in details in [13]. This methodology was used to extend the OWL automatic generated ontology which we intend to be represented in OWL DL to perform the axiomatic verification.

Figure 2 illustrates the complete study case workflow used in this work. Note that it describes the whole tool chain (Poseidon, Xalan, Protégé, Racer) along with documented models used (UML, OUP, XSLT, OWL) to depicts a real setting of how ontologies can be developed based on UML tools. The dashed rectangle in this figure illustrates the phases that we consider the main contribution of this

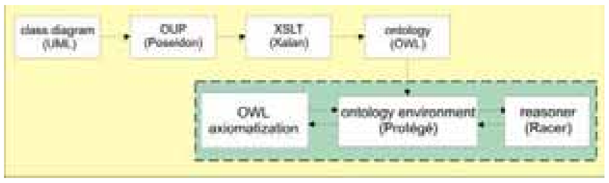


Figure 2. The Study Case Workflow.

work, since the previous work in the related literature does not include ontology verification, through the use of reasoner tools, what may lead to ontologies' inconsistence.

Now, we describe the three phases of our study case. (i) Use of the system's UML documentation, more specific the domain model, to adapt it to OUP and to export as a XMI document. (ii) To use the XMI document to generate a preliminary OWL ontology. The XMI document will serve as input for a XSLT processor (Xalan), that will produce as output a OWL document. (iii) To import the OWL document in an ontology tool (Protégé), where it will be analyzed to create an extended ontology with all the axioms necessary for the development of other related systems considering the chosen domain.

Starting from Protégé, and after having initialized the Racer reasoner, we used the following inference services to validate the ontology: verification of the consistence and classification of the ontology. Following this approach, our experiments used two different modules of GSWeb a subproject of GIMPA's project used at the University of Brasília Hospital (HUB), better described in Section 5.1.

5.1 GIMPA's Project

GIMPA is a project that deals with medical information of patients allied to Brazil's public health system at the Braslia University Hospital (HUB) [9]. GIMPA is subdivided into different modules to deal with different medical information of patients and medical assistance. One of these modules is the GSWeb, an information management system of individual patient's that includes a patient's Electronic Record. Besides the clinical information, the Eletronic Patient Record (EPR) includes automatic communication to different medical equipments.

The several modules of GSWeb include evaluation and diagnostic report related to different exams that involve aspects of the heart and other organs: (i) Echocardiogram (ECO)-uses ultrasounds to examine the heart, getting images in one and two dimensions that allow to evaluate the size, thickness and movement of diverse cardiac structures, giving anatomical and functional evaluations; (ii) Cardiac Cintilography (CINT)-verifies the anatomy and functionality of the heart by means of isotopics radio markers; (iii) Electrocardiogram (ECG)-accomplishes the graphic representation of the heart electric activity on the body surface, interpreting anatomical data and cardiac functionalities; (iv) Thorax Radiography (X-Ray)-the use of radiation for the scientific examination of material structures, radioscopy; (v) Computerized Tomography (CT)-verifies the anatomy and functionality of the heart with contrast and without contrast and (vi) Nuclear Magnetic Resonance (RMN)-through the distribution of atoms, it evaluates the anatomy and cardiovascular function.

The ECO and X-Ray modules were used in our study case. ECO is a system to register information related to the echocardiogram exam aiming the generation of corresponding reports. The information used in ECO comes from the patient's electronic record and the equipment that accomplishes the echocardiogram exam. The X-Ray system aims to support diagnosis and treatment of diseases related to thorax and other organs. At the time of our experiments this GSWeb module was under development and documentation, and we have used it to integrate to ECO module. For that purpose, we developed the ontology that included all the concepts and relationships of ECO module and then extended our ontology to include the concepts and relationships of X-Ray. This aim was possible because the reports in GSWEB are parameterized. A complete description of our experiments is in [13]. Section 5.2 presents an illustration of the study case using the phases presented in 5.

5.2 OUP Application

Figure 3 illustrates the OWL study case ontology automatically generated from UML through the OUP transformation. The ontology include concepts related to the heart's function and anatomy used by ECO module of GIMPA. After the transformation, the OWL ontology is ready to be imported into Protégé to be further refined, validated or extended. This figure is done using Jambalaya (Protégé's plug-in to visualize OWL ontologies). Note the hierarchical structure of the ontology, where the class *Parede* (i.e., wall) is dependent of *SubEstruturaAnatômica* (i.e., anatomical substructure); while the class *Átrio* (i.e., atrium) is dependent of *EstruturaAnatômica* (i.e., anatomical structure). Both classes *SubEstruturaAnatômica* and *EstruturaAnatômica* are dependent of the *Estrutura* (i.e., structure) class. Figure 4 shows part of the study case ontology focusing the *Estrutura* class. The whole ontology in both OUP and OWL formats are available at [5].

OUP uses the standard UML extension and customization mechanisms defined in the UML specification; i.e., stereotypes, tag definitions, tagged values and constraints. According to [14] stereotypes enable defining virtual subclasses of UML metaclasses, assigning them additional semantics. In the used approach, the stereotypes execute a central role at the classes and properties definition. For instance, to model ontology classes in OUP, we used stereotyped UML class <<*OntClass*>>. The other class types, like Enumeration, Union and Restriction, are also created with a new stereotyped class for each of them. As the UML metamodel is in the M2 layer of MDA, the extensions are also made in the M2 layer (see Section 2).

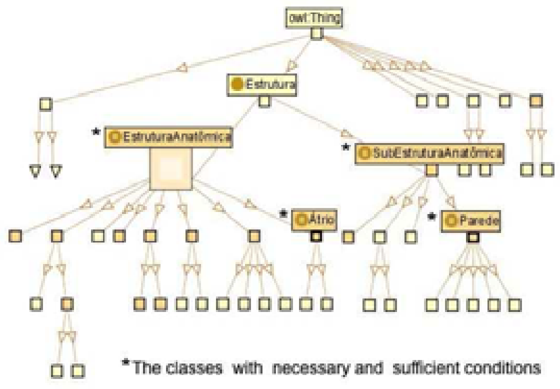


Figure 3. Ontology's Class Hierarchy.

In UML, an instance of a Class is an Object. Since ontology individuals and UML Object have some differences, OUP Individuals are modeled as stereotyped UML Objects with <<*OntClass*>>. In ontology languages (e.g. OWL) Property is a stand-alone concept, unlike the attributes in UML, which are part of the UML class they belong. According to the semantics of OWL (that OUP is based on) there are two types of properties: <<*ObjectProperty*>>, which can have only Individuals in its range and domain, and <<*DatatypeProperty*>> a string type.

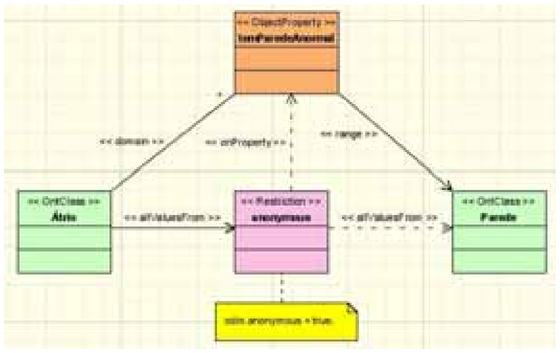


Figure 4. Study case class diagram with ontology properties.

Figure 4 shows a class diagram, an excerpt of the study case ontology, which depicts ontology properties modeled in UML. Note that the class *Átrio* (i.e., Atrium) is at the domain property of *temParedeAnormal* (i.e., hasAbnormal-Wall), that is a *<<ObjectProperty>>* whose range is the class *Parede*. The stereotype *<<Restriction>>* is used to refine the restrictions of properties. In this way, the class' restriction on a property the *Átrio*'s *<<ObjectProperty>> temParedeAnormal* has as *allValuesFrom* restriction the *<<OntClass>> Parede*. That means, each instance of the *Átrio* class must have at least one instance of the property *temParedeAnormal* whose range is the *Parede* class.

5.3 Axiomatic Verification

One of the key features of an OWL DL ontology is the fact that it can be processed by an automatic reasoner. The reasoner executes two main functions: the classification verification and the ontology consistence. The classification function is performed by testing all ontology classes to generate the ontology's inferred hierarchy. Another function is the consistency check, which is based on the classes description/condition to verify whether the classes can have instances. The reasoner uses the necessary and sufficient conditions to verify the ontology classification. According to [19], the classes need to be specified with necessary and sufficient conditions so that the reasoner can determine that any individual or class which satisfies these conditions must be a member of the class.

Figures (3,5) shows part of the study case with the class hierarchy ontology focusing the class *Estrutura*. The XSLT used generates automatically all classes with necessary conditions, what does not allow the classification verification by the automatic reasoner. Thus, we have redefined some classes with necessary and sufficient conditions, according to the domain specification, in order to allow the automatic verification of our generated ontology. To verify the classification verification and consistence of the OWL ontology, we used the Racer reasoner (educational license) with Protégé.

In our study case, the advantage of using the reasoner could be checked in practice. Using the option of consistence verification, it was possible to notice some mistakes at the ontology definition. Some domain properties of a class had been used in the definition of other classes accusing consistence error. For example, the property *temAnormalidade* (i.e., hasAbnormality) possessed by the domain class *SeptoInteratrial* (dependent of the class *EstruturaAnatômica* at the asserted part of the figure 5); however it was also used by the class *Átrio* (i.e., Atrium dependent of the class *EstruturaAnatômica* at the asserted part of the figure 5). Such fact generated an inconsistency, that spread to the other classes. The same also happened with some definitions of range. After the correction of these errors the ontology was ready to incorporate the X-Ray module of GSWeb system.

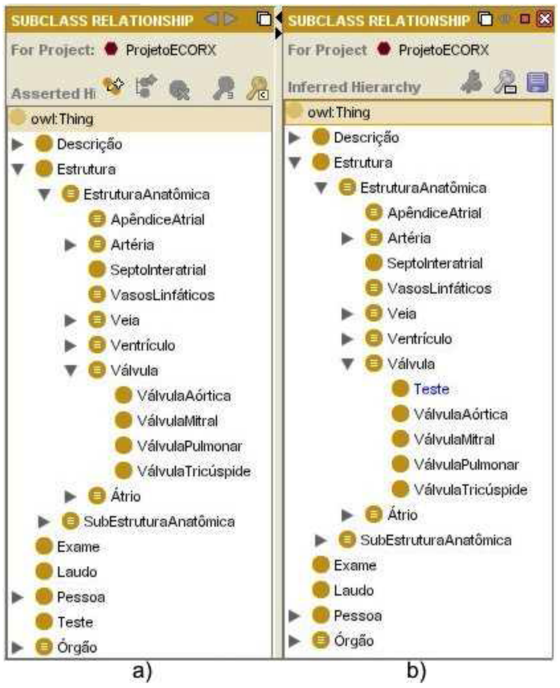


Figure 5. Asserted (a) and Inferred (b) hierarchy of Projects ECO and X-Ray ontology.

Once initialized the Racer reasoner, we used the classification inference service to classify the ontology. The OWL ontology classes are named asserted while the inferred classes are checked by Racer. Figure 5 illustrates the asserted (a) and inferred (b) hierarchy of Projects ECO and X-Ray ontology used in our study case. In order to use Racer's classification inference service, we have created a class named *Teste*, dependent on the *owl:Thing* class and with the ∀ *regurgitação Grau* necessary condition. Since we have defined the *Teste* class as described, Racer could correctly classify it as dependent of *Válvula* class, once *Válvula* class is defined to have as necessary and sufficient conditions ∀ *regurgitação Grau*.

6 Conclusions

In this paper, we have shown a practical realization of the XSLT-based approach, for automatic generation of the OWL ontology from a UML model developed at the GSWeb medical system. Our work was motivated by the fact that although MDA-based technology is available, we could not find real use-cases in the literature to demonstrate how this technology can be really used. We presented some related approaches to MDA-based automatic OWL ontology development, but we could not compare results since only the UML profile is available in [6], but the implementation is not. The approach used in this work is also important since it allows the use of free tools, e.g. Poseidon for UML Community Edition, Xalan, Protégé; while in the approach of [6] they define tools developed by Sandpiper (e.g., VOM - an add-in to IBM's Rational Rose product).

Our experiments demonstrated that to integrate different computational systems the presented approach is a reasonable solution. One obstacle at the beginning of our project was to acquire knowledge to work with the OUP standard and the UML extension and customization mechanisms defined in the UML specification; i.e., stereotypes, tag definitions, tagged values and constraints. Also to work in Poseidon with classes with the stereotype <<*Restriction*>> named *anonymous* is a bit okward, since it is hard to find out the alignments between properties and restrictions. Unfortunately, UML does not have notation to show values of tagged values in diagrams, but users may add that in a UML note connected to the <<*Restriction*>> as shown in Figure 3.

In order to illustrate the axiomatic verification of the generated ontology, we have extended the OWL study case ontology to allow the integration of other computational systems. The final ontology has 69 classes and 77 properties (including datatype and object). Once extended the OWL ontology, we could do the consistence verification using Racer. With our ontology classification test, we concluded that the presented approach can help to correct mistakes in the ontology definition. But for ontology developers using UML tools to obtain feedback of consistency of their ontology, they have to go to another tool such as Protégé to use a reasoner. This scenery could be improved if current UML tools executes two important functions: classification and consistence of the ontology. Also if current UML tools were adapted to OUP transformations it would be possible to transform from OUP to OWL and vice-versa.

The future research should explore ways to represent reasoning facts inferred by the reasoner in a UML tool. Another important aspect is to find ways how reasoners and ontology tools can be integrated (e.g., Web Services), so that they can support the proposed approach allowing ontology developers to stay inside a UML tool when developing ontologies and at the same time to be able to use reasoning services (e.g., FaCT allows using CORBA to use its reasoning services).

References

[1] `http://www.racer-systems.com/`.

[2] `http://www.sandsoft.com/products.html`.

[3] `http://www.alphaworks.ibm.com/tech/semanticstk`.

[4] `http://protege.stanford.edu/`.

[5] `http://www.cic.unb.br/~ghedini/`.

[6] S. Brockmans, R. M. Colomb, E. F. Kendall, E. K. Wallace, C. Welty, and G. T. Xie. A Model Driven Approach for Building OWL DL and OWL Full Ontologies. In *The 5^{th} Int'l Semantic Web Conf.*, Athens/GA, USA, 2006.

[7] S. Brockmans, R. Volz, A. Eberhart, and P. Lfler. Visual modeling of OWL DL ontologies using UML. In *Proc. of the 3^{rd} Int'l Semantic Web Conf.*, pages 198–213, Hiroshima, Japan, 2004.

[8] A. Brown. An introduction to Model Driven Architecture Part I: MDA and todays systems, February 2004. IBM developerWorks.

[9] H. S. Carvalho, C. J. N. C. Jr, and W. B. Heinzelman. Gerenciamento de Informações Médicas do Paciente (GIMPA). In *Proc. of VIII Congresso Brasileiro de Informática em Saúde*, Natal/RN, Brasil, 2002.

[10] S. Cranefield. Networked Knowledge Representation and Exchange using UML and RDF. *J. of Dig. Info.*, 1(8):118–143, 2001.

[11] M. Dean and G. Schreiber. OWL Web Ontology Language Reference. Technical report, W3C Consortium, 2004.

[12] D. Djurić, D. Gašević, and V. Devedžić. Ontology Modeling and MDA. *J. of Obj. Tech.*, 4(1):109–128, 2005.

[13] E. S. dos Santos. Uma Proposta de Integração de Sistemas Computacionais Utilizando Ontologias. Master's thesis, Departamento de Ciência da Computação, Universidade de Brasília, Brasília, Brasil, 2006.

[14] D. Gašević, D. Djurić, and V. Devedžić. Bridging MDA and OWL Ontologies. *J. of Web Eng.*, 4(2):118–143, 2005.

[15] D. Gašević, D. Djurić, and V. Devedžić. *Model Driven Architecture and Ontology Development*. Springer, NY, 2006.

[16] D. Gašević, D. Djurić, V. Devedžić, and V. Damjanović. A UML profile for OWL ontologies. In *Proc. of Model-Driven Architecture: Foundations and Applications - MDAFA*, pages 138–152, Linkoping, Sweden, 2004.

[17] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[18] G. K. Holman. What is XSLT?, 2000.

[19] M. Horridge. A practical guide to building OWL ontologies using the Protege-OWL Plugin and CO-ODE Tools, 2004.

[20] N. F. Noy and D. L. McGuinness. Ontology Development 101:a guide to creating your 1st ontology. Working Paper KSL-01-05, Stanford Knowledge Syst. Lab., March 2001.

[21] Object Management Group. Ontology Definition Metamodel-Request for Proposal, March 2003.

[22] Object Management Group. Meta Object Facility(MOF) Core Specification. Technical rep v2.0, OMG, January 2006.

[23] Object Management Group. UML: Infrastructure. Technical Rep v2.0, OMG, March 2006.

[24] Ontology Definition Metamodel. Sixth Revised Submission to OMG/ RFP ad/2003-03-40, 2006.

[25] J. D. Poole. Model-Driven Architecture: Vision, standards and emerging technologies. In *ECOOP'01: Proc. of the Work. on Metamodeling and Adaptive Obj. Models*, 2001.

[26] T. Stahl and M. Voelter. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.

[27] P. Tetlow, J. Pan, D. Oberle, E. Wallace, M. Uschold, and E. Kendall. Ontology Driven Architectures and Potential Uses of the Semantic Web in Software Engineering, May 2005. Editors' Draft W3C.

Towards Domain-Centric Ontology Development and Maintenance Frameworks

Faezeh Ensan and Weichang Du
Faculty of Computer Science, University Of New Brunswick, Fredericton, Canada
{*faezeh.ensan, wdu*}*@unb.ca*

Abstract

In this paper, we attempt to study and investigate ontology development and maintenance frameworks from a domain-centric point of view. By frameworks we mean the structures which have been designed to allow ontology engineers and domain experts to develop and maintain domain ontologies. Such frameworks usually specify particular phases for developing ontologies and provide implemented components for each phase. Our purpose is to analyze the suitability of a framework for developing ontologies which can fulfill the necessities of a specific domain. We have designed a comparison model for analyzing ontological frameworks. Using the model, we inspect how an ontological framework utilizes domain information resources for creating and maintaining ontologies, how much fineness and granularity the designed ontology can reach, and with how much maturity it supports the maintenance and integration capabilities in the development process.

1. Introduction

In his seminal paper, Gruber [6] defines an ontology as 'an explicit specification of a conceptualization'. Ontologies are used by applications, domain experts and users in order to reach consensus on various concepts of a domain of discourse for the purpose of collaboration and communication. There have been different proposals for the application of ontologies. Niles et al. target the creation of an high-level ontology for defining general-purpose concepts [10]. This ontology can be used as a foundation for other domain specific ontologies. On the other hand, other approaches attempt to design ontologies to describe parts or all of the concepts of a specific domain.

So far, numerous methodologies, methods and tools have been proposed for developing ontologies [4, 3]. In this paper, we are concerned with studying and analyzing ontology development and maintenance frameworks from a domain-centric point of view. By frameworks we mean those proposals that target the main parts of the ontology development process. They specify particular phases for building and managing ontologies and usually provide implemented components for each phase.

Our objective is to study the applicability and suitability of general-purpose ontology frameworks for creating domain-centric ontologies. We intend to analyze the methods for integrating domain specific information into such frameworks. We attempt to investigate how ontologies that are developed by a framework for a specific domain are extracted from domain specific knowledge and information, how they fulfill domain necessities and how they evolve based on arising requirements and changes.

We propose a domain-centric comparison model for analyzing and investigating ontological frameworks. This model has five dimensions. Through the 'Resources' dimension, we study the domain resources which are exploited by a framework in its ontology development process. The 'Scope and usage coverage' dimension specifies the types of applications which can utilize a developed ontology and how the scope of the ontology is determined.'Fineness and independency' examines how much domain granularity an ontology can reach through the employment of the development framework. It also investigates whether a framework can be applied to all domains or it is only designed for a specific one. The 'Maintenance capability' and 'Integration capability' dimensions study how completely a framework supports the maintenance of ontologies and how well it exploits other related ontologies in its process, respectively.

In Section 2, we classify a range of ontological frameworks. We select instances from each category and discuss more about their proposed processes. In Section 3, we fully introduce the proposed comparison model. Section 4 provides some concluding remarks.

2. Ontological Frameworks

We classify ontological frameworks into three main categories. In the first category, we are interested in the frameworks that attempt to take into consideration all of the possible activities of the ontology development process. These activities include the development of one or more ontologies for a domain of discourse, managing them and letting them to be automatically or manually changed, and keeping and manipulating several versions of the target ontologies. We have selected Text-To-Onto [7] and OntoLearn [9] as two case studies and will analyze them within our proposed domain-centric comparison model.

In the second category, the main aim of the selected frameworks is the management of ontologies, instead of their design. They mostly focus on managing ontologies, integrating several ontologies of the same domain and their evolution rather than constructing ontologies. These types of frameworks can be integrated with other discussed frameworks in order to cover all activities of the ontology engineering process.

Integrating several ontologies related to a specific domain is an important aspect of ontology maintenance. For a domain of discourse, there may be more than one ontologies that have been designed by developers with dissimilar approaches that describe different parts of the same domain. Integrating these ontologies has been the topic of several research papers. Wache et al. [15] give a good survey explaining possible ways for ontology integration. They argue over three approaches of using ontologies for describing a domain of discourse. Single ontology approach stands for a situation that there is only a global ontology for a domain. In this environment there may be other particular ontologies, but all of them should be mapped to the general one. Multiple ontologies, is the second approach discussed in [15]. In this approach there are several ontologies in the system that may be developed independently. In order to integrate such diverse ontologies, inter-ontology mapping algorithms are necessary. PROMPT [11] is a framework that has the appropriate capabilities for maintaining ontologies based on this approach. It has components that provide inter-ontology mappings, merging and alignment. In Section 2.3 we will talk about the PROMPT framework more completely. The third approach is the hybrid ontology approach, where there are several ontologies for a domain, but all of them share a common vocabulary (or upper ontology).

The third category of the selected frameworks is devoted to the frameworks which aim at developing or maintaining ontologies related to a very specific domain. We have chosen the medicine domain, one of the most popular domains that ontologies have been employed. So far, many ontologies as well as methodologies and methods have been proposed in bio-medicine, molecular biology and generally speaking, the medicine domain. Open Biomedical Ontology (OBO) library [2] is a collection of bio-ontologies which have been designed and shared across medical and biological domains. The Gene Ontology (GO) project provides 'a controlled vocabulary to describe gene and gene product attributes in any organism' [1]. We have selected Oasis [12], an integration framework for biomedical ontologies, which best fulfils our definition of ontological frameworks and will study it thoroughly in Section 2.4.

2.1 Text-To-Onto

Maedche et al. [7] have developed an almost mature ontology learning framework. The development process of this framework consists of the following steps:

Importing and reusing other ontologies: In this step, domain experts identify and select appropriate ontologies, schemas and conceptualizations which are related to the domain of discourse. This framework suggests making use of the FCA-MERGE [13], a bottom-up merging method, for integrating domain ontologies.

Ontology Extraction: This step is the major part of the proposed framework. The framework includes OntoEdit as a workbench to help experts collaboratively design ontologies; however, the framework also considers learning ontologies from resources other than experts. Text-To-Onto is a method suggested to be used for extracting ontology primitives from natural language documents. It includes several algorithms for extracting each ontological element so that experts or ontology engineers can choose the appropriate one under different conditions. The Text-To-Onto method first extracts domain related concepts by processing natural language documents using a shallow text processor. Domain concepts are selected based on frequency measures like TF-IDF. In the second step, the method hierarchically clusters domain concepts to form a taxonomic relationship between them. Non-taxonomic relations are the other ontological primitives that are extracted in the third step. Text-To-Onto applies association rule mining algorithms in order to acquire the relationships between the concepts.

Pruning the result ontology and refinement: The result ontology from previous phases may focus on some unnecessary details but may not include some important and useful information. In the pruning phase, some parts of the ontology are removed and then in the refinement phase some ontological elements are inserted. These steps are the ontology management parts of the framework.

This framework has an evolving nature. After each round, ontology engineers can decide to begin the process cycle again. In each phase, the information extracted in the previous cycles can be utilized as background knowledge.

2.2 OntoLearn

Navigli et al. [9] have designed a framework for learning domain based ontologies from relevant documents. Their ontology engineering framework is comprised of three phases: First, the main domain concepts and taxonomic relationships between them are automatically extracted from domain documents. In the next phase, these concepts are evaluated by the experts through a groupware package called ConSys, and finally all the new accepted concepts are inserted into the ontology using the SymOntoX tool [8]. SymOntoX accepts new concepts and relations from ConSys as incoming suggestions and inspects and applies them to the target ontology.

Phase one of the suggested framework has been realized by the OntoLearn system. OntoLearn itself has three phases. First, terminology concepts are extracted using a linguistic and syntactic parser. The extracted terminologies in the first phase may be compound phrases which consist of several terms. For example, the system may recognize 'bus service', 'coach service' and 'ferry service' as domain terminologies. Every term in a complex terminology may have several senses. Identifying the best sense for each term in the compound phrase is a challenging task. The second phase of OntoLearn is comprised of three sub-phases. First with a novel algorithm named SSI and using upper taxonomic knowledge sources like WordNet, one sense is assigned to each term in a complex phrase. In the second sub-phase, the OntoLearn method determines appropriate hierarchical relationships between compound terms by clustering synonym and similar terms as well as taxonomic information of WordNet. In the third sub-phase, for extracting non-taxonomic relationships between the components of a compound phrase, an inductive learner is employed. First, an ontology engineer manually tags some instances of domain terms with appropriate relationships relevant to the domain, and then the learner builds a tagging model. The inductive learning system needs a vector of features for each instance. The authors in [9] suggest that the feature vector of each compound phrase should be shaped by all of the hyperonyms of its terms. The OntoLearn framework includes methods for creating and managing ontologies in an evolving manner. Each new concept or relation is learnt by OntoLearn and validated by the experts and inserted through SymOntoX. The management tasks are done in a collaborative manner. SymOntoX define three levels for manipulating ontologies. A simple user can only view the ontology, a super user can read and write, and an ontology master can modify the ontology and validate the suggestions of the super users.

2.3 PROMPT

PROMPT [11] is an ontology management framework which is developed at the Stanford University. The main aim of PROMPT is to provide features for multiple ontology management tasks which include the maintenance of ontology libraries, import and reuse of other related ontologies, support for ontology versioning , merging, mapping and aligning ontologies and factoring independent sections of the ontology. PROMPT consists of four components:

iPROMPT is a component which helps its users merge different ontologies describing the same domain of discourse. Given two ontologies, iPROMPT finds similar concepts between them and suggests them as merging candidates to the ontology engineers. The users can follow the suggestions or select other concepts for merging.

AnchorPROMPT finds similarity between different ontologies; therefore, it facilitates their mapping and alignment. It contains a graph-based algorithm which finds similar ontological concepts of two ontologies, based on a given set of identified anchors.

PROMPTDiff controls ontology versioning. Utilizing some defined heuristics, this component attempts to match ontological elements of two versions of the same ontology and finds the changes that have been applied in the new version compared to the last one.

PROMPTFactor is a component that allows its users factor some part of a huge domain ontology for a specific use.

2.4 Oasis

Oasis [12] is an integration framework for bio-medical ontologies. It provides a warehouse for bio-medical concepts, saves them, finds similarity between the concepts of different ontologies and interactively with the domain experts, maps similar concepts onto each other. The Oasis framework consist of two major parts: a database and a novel mapping tools named IOMG. The Oasis framework has been mainly influenced by the OBO ontologies structure. The database is comprised of three types of tables. Firstly, the OBO tables which include terms and term2terms. Term2term are all ontological terms that have an is-a or has-a relationship with each other. Secondly, Mapping tables are tables that store the concepts that have been mapped by the framework. Finally, proprietary tables that maintain concepts from other ontologies rather than OBO.

IOMG is a tool designed for finding similarity between domain terms and mapping them. It also utilizes a graph-based representation of the OBO ontologies for finding possible similarities. Oasis defines three similarity metrics: linguistic similarity between term names, similarity between the definition of ontological terms that are provided for most of GO terms and also similarity between parents and children of each term in the graph representation. Based on these metrics, IOMG finds a similarity value between each pair of concepts for two given ontologies. IOMG attempts to find proper pairs from all of the candidate pairs by maximizing an objective function of all similarity values. Consequently, not only similarity of two terms is important for applying a mapping, but also the best conditions under which all of the mapped pairs have their highest similarity values is also considered. Ultimately all established links that cause a cycle in the graph are identified and introduced to domain experts as potential failure links.

3. A Domain-centric Comparison Model

The major goal of our proposed model is that given a specific domain for examination, it would assist the analysts in the selection process of an appropriate framework from the already existing huge number of methods and approaches in order to best fulfil the target domain's necessities. In the following, we thoroughly explain each of its dimensions and analyze the introduced frameworks based on these dimensions.

3.1 Domain Resources

Different frameworks suggest dissimilar information resources for extracting ontological elements. In many methods and frameworks, domain experts are reliable references for the development of ontologies. Based on this fact, the experts' role may significantly vary: some times the only references for every thing are the domain experts who solely or collaboratively design and maintain ontologies, while in other approaches experts only confirm what has been extracted by machine learning algorithms. For example in the PROMPT framework, the only available domain resources are the domain experts who modify ontologies whereas in the OntoLearn Framework, experts confirm and finalize ontological concepts extracted from other domain resources. In another approach, such as the model proposed in the Text-To-Onto framework, domain experts may specify information resources that should be utilized by automatic learning algorithms.

The upper ontology is another learning reference which may be used in the ontology development process. For instance,OntoLearn uses WordNet for disambiguating terms in compound phrases and extracting taxonomic and non-taxonomic relationships between terms. The other domain resource for developing ontologies are semi-structured or natural language Web documents that may be exploited by automatic algorithms for extracting concepts and relationships. Database schemas, domain thesauri and dictionaries are examples of semi-structured information resources. The Text-To-Onto and OntoLearn frameworks make use of domain related Web documents for extracting terminologies and relationships. Furthermore, the former one also exploits semi-structured domain resources. Oasis uses the synonym table of OBO along with the definitions related to each GO concept for finding similarity between medical concepts.

3.2 Domain Scope and Usage Coverage

Specifying the scope of the target ontology is an important subject which has been discussed in various methodologies. The first phase of Uschold and Grninger's methodology [14] is to specify the scope of the ontology. The specification phase of METHOONTOLOGY [5] also aims to find the scope of the ontology. What is important for us in this comparison model is how different frameworks specify the scope of the target ontology. Domain experts have an important responsibility in most cases. The domain scope of a framework can be investigated through experts and all of the available information. While there have been no suggestions for specifying the scope of an ontology in some frameworks, some others utilize a combination of these methods.

Observably, experts are the only reference for defining the ontology scope in frameworks which their development process is completely based on human efforts. They should decide on the degree of relevancy of a new concept to the domain and whether it should be inserted into to the ontology or not. Alternatively, some frameworks attempt to develop or evolve domain ontologies based on all of the available information. The OntoLearn framework aims to extract all of the ontological concepts and relationships from Web documents. However, experts can also influence identifying the borders of the developed ontology by confirming or rejecting ontological elements through the ConSys component. The Text-To-Onto framework also follows a similar approach.

In usage coverage, we consider the types of applications that can benefit from the designed ontology. We show whether all existing applications can utilize the created ontology or only specific ones which has been indicated by the domain experts can benefit from it. Navigli et al.[9] suppose that the coverage of an ontology should be very vast to make it useful. They state that all of domain related concepts should be available in a suitable ontology. Based on such definition, all of the applications related to that domain should be able to make use of the designed ontology and hence communicate on this basis. On the other hand, some frameworks attempt to evolve the domain ontology based on the users' needs. In this approach, only domain applications which are specified by the domain experts and ontology engineers can exploit the designed ontology. The PROMPT and Oasis frameworks are examples of such an approach.

3.3 Maintenance Capabilities

Several frameworks have been suggested for developing and managing ontologies. As was explained previously, some of the frameworks emphasize more on constructing ontologies whereas the others focus more on maintaining and evolving them. However, the ontology learning and constructing frameworks that we have chosen provide partial capabilities for ontology evolution.

The maintenance capabilities criterion investigates how thorough a framework supports the maintenance of domain ontologies. We have studied two items in this field: evolution support and also the quality of ontology manipulation. Preserving different versions of an evolving ontology, undoing changes applied to a version and retrieving the latest versions of an ontology are considered as factors indicating the degree of evolution support. Among the introduced ontological frameworks, only PROMPT provides evolution support to some extent. The PROMPTdiff component of the PROMPT framework is designed for representing changes between different versions of the ontology without making use of change logs. PROMPT considers conditions where the selected ontologies have been developed in different environments so a unified change log is not accessible. The framework does not mention other versioning activities like reversing the applied changes by particular users.

Ontology manipulation can be realized through either expert collaboration or the evolving lifecycle of the development process of a framework. The most popular approach in ontology maintenance is to allow domain experts manipulate ontologies and change it according to their needs. Nonetheless, implementation of this simple schema in a distributed environment could have some practical difficulties. Through our comparison model, we examine whether a framework provides the experts with the possibility of collaboratively managing ontologies in a distributed environment or not. The Text-To-Onto and OntoLearn frameworks, are instances of frameworks that allow experts manage ontologies in a collaborative manner.

An evolving lifecycle is another way to adapt a designed ontology to new conditions. New changes and modification can be learnt in each cycle during ontology development. Both OntoLearn and Text-To-Onto exploit this type of lifecycle for updating an under-development ontology over time.

3.4 Integration Capabilities

For a specific domain, there may exist other ontologies that have already been designed. Ontology development and management frameworks might utilize these ontologies in their process. In the proposed comparison model, we want to investigate how a framework exploits other existing ontologies. Some frameworks provide no integration capability, such as the case in the OntoLearn framework which does not suggest any method for utilizing other ontologies. As another approach, a framework may make use of other related ontologies for developing one inclusive domain ontology, such as the Text-To-Onto framework. Some other frameworks like Oasis and PROMPT follow another approach for integration. They attempt to concurrently integrate several ontologies for a domain of discourse in a 'multi-ontology environment'.

3.5 Domain Fineness and Independency

Domain fineness is concerned with how much granularity a framework attempts to achieve while enriching the domain ontology. Similar linguistic words may have different meanings in different domains. A good framework should be able to deal with each word based on its specific semantic in the target domain. Furthermore, domain specific relationships might exist that can only be extracted when a framework exploits domain knowledge and heuristics. The OntoLearn framework exploits domain specific information and knowledge for extracting ontology concepts to some extent. The disambiguation algorithm assigns domain related sense to each term of a compound phrase. Furthermore, in the case of non-hierarchal relationships among classes, the framework makes use of domain relative information for generating rules which tag compound terms with appropriate non-taxonomic relationships. The Oasis framework also utilizes domain specific features for integrating ontologies. OBO graph representation as well as the definitions that are provided for GO concepts is used by Oasis to find similarity between the concepts.

Through domain independency, we study dependency or independency of a framework to a specific domain. One can investigate whether and with how much accuracy a framework can be applied to different domains. In some circumstances, there is a tradeoff between these two criteria. The frameworks that attempt to apply too many domain specific heuristics and knowledge might be not completely domain independent. Among the introduced frameworks, Oasis is the only one which is completely dependent to the medical domain and cannot be employed by other domains.

4. Concluding Remarks

Ontological frameworks are those that attempt to suggest ontology engineers a process for developing or maintaining ontologies. They attempt to clarify each step of the process by providing an implemented component. They have not been designed for end-users nor applications. They target engineers who want to develop ontologies for the end-users. We have designed a model for comparing and investigating

Table 1. Analysis of the Ontological Frameworks From Domain-centric Perspective.

	Text-To-Onto	OntoLearn	PROMPT	Oasis
Experts' Role	Selection of Other Resources, Approve and finalize ontological elements learnt by the method.	Approve and finalize ontological elements learnt by the method.	Responsible for all tasks	Responsible for all tasks
Web Docs	Natural language documents from the Web	Domain-based web documents	-	-
Upper Ontologies	-	WordNet	-	-
Semi-Structured Resources	domain thesaurus and domain related schemas	-	-	OBO synonym tables and the definition related to each GO concept
Model for Scope Specification	All information/Domain Experts	All information/Domain experts	Domain experts	Domain experts
Usage Coverage	All applications	All applications	Specific applications	Specific Application
Evolution Support	-	-	Versioning	-
Ontology Manipulation	Evolving development life cycle&Collaborative manip.	Evolving development life cycle&Collaborative manip.	Standalone manip.	Standalone manip.
Integration Capabilities	Based on global ontology	-	Multi-ontology environment	Multi-ontology Environment
Domain Fineness	-	Use of domain knowledge for concept extraction	-	Use of domain knowledge for Integration
Domain Independency	Independent	Independent	Independent	Doamin dependant

such frameworks. Our framework focuses more on the domain capabilities of each framework. We investigate how an ontological framework can utilize domain knowledge and also how it can be used for a certain domain of interest. We extracted five criteria for investigating such frameworks. While some of criteria focus on generating a general view of the application of a given framework, the others attempt to find domain relevancy of it and also domain closeness of the generated results. Table 1 summarizes the results of the performed comparisons.

Considering the domain perspective, our main attempt for future research is to design a high-level methodology which assists engineers in developing domain-centric ontological frameworks. This methodology would allow the designers to specify the goal of the framework and its capabilities, the domain knowledge it intends to exploit, the domain coverage of the final ontology which is designed by the framework and also its dependency or independency to a specific domain. This methodology would be a meta-methodology which is designed not for developing ontologies but for crafting frameworks which manage and design ontologies.

References

[1] The gene ontology, http://www.geneontology.org/, last visited feb 2007.

[2] Open biomedical ontologies, http://obo.sourceforge.net, last visited feb 2007.

[3] O. Corcho, M. Fernandez-Lopez, and A. Gomez-Perez. Methodologies, tools and languages for building ontologies: where is their meeting point? *Data Knowl. Eng.*, 46(1):41–64, 2003.

[4] Y. Ding and S. Foo. Ontology research and development. part 1- a review of ontology generation. *Journal of Information Science*, 28(2):123–136, 2002.

[5] M. Fernandez-Lopez, A. Gomez-Perez, J. P. Sierra, and A. P. Sierra. Building a chemical ontology using methontology and the ontology design environment. *IEEE Intelligent Systems*, 14(1):37–46, 1999.

[6] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *Int. J. Hum.-Comput. Stud.*, 43(5-6):907–928, 1995.

[7] A. Maedche and S. Staab. Ontology learning for the semantic web. *Intelligent Systems, IEEE [see also IEEE Intelligent Systems and Their Applications]*, 16(2):72–79, 2001.

[8] M. Missikoff and F. Taglino. Symontox: a web-ontology tool for ebusiness domains. pages 343–346, 2003.

[9] R. Navigli and P. Velardi. Learning domain ontologies from document warehouses and dedicated web sites. *Comput. Linguist.*, 30(2):151–179, 2004.

[10] I. Niles and A. Pease. Towards a standard upper ontology. In *FOIS '01: Proceedings of the international conference on Formal Ontology in Information Systems*, pages 2–9, New York, NY, USA, 2001. ACM Press.

[11] N. F. Noy and M. A. Musen. The prompt suite: interactive tools for ontology merging and mapping. *Int. J. Hum.-Comput. Stud.*, 59(6):983–1024, 2003.

[12] G. Song, Y. Qian, Y. Liu, and K. Zhang. Oasis: A mapping and integration framework for biomedical ontologies. In *CBMS '06: Proceedings of the 19th IEEE Symposium on Computer-Based Medical Systems*, pages 611–616, 2006.

[13] G. Stumme and A. Maedche. FCA-MERGE: Bottom-up merging of ontologies. In *IJCAI*, pages 225–234, 2001.

[14] M. Uschold and M. Gruninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11:93–136, 1996.

[15] H. Wache, T. Vogele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hubner. Ontology-based integration of information - a survey of existing approaches. In *workshop on Ontologies and Information Sharing at the International Joint*, pages 108–117, 2001.

Service Oriented Architecture Empirical Study

Mohammad Abu-Matar
George Mason University
mabumata@gmu.edu

Jeff Offutt
George Mason University
offutt@ise.gmu.edu

Abstract

Service Oriented Architecture (SOA) has been proposed as model for distributed software development that surpasses the traditional Distributed Object Architecture (DOA) practices in many areas. However, no empirical studies have been conducted to verify the claimed benefits. This study is a first attempt at presenting empirical evidence regarding the benefits of SOA. It is a comparison between traditional DOA and SOA. The two technologies were compared on the basis of code size and development time. The results show that, as a whole, the subject application was faster to develop using the SOA method. However, the application size was larger than that developed in the DOA method.

Key Words: SOA, Web Services, EJB, SOA experimentation

1. Introduction

Traditional Distributed Object Architecture (DOA) used in building multi-tier Web based applications can reach its limit when highly distributed heterogeneous systems need to interoperate. Modern web applications need a new computing model where services are treated as self-contained modules that can be advertised, discovered, composed, and negotiated on demand.

Service Oriented Architecture (SOA) is a model for designing, building, and deploying distributed software applications. It emphasizes loosely coupled design approaches where disparate systems, with different computing platforms, can collaborate without major changes to their existing core architectures.

Despite wide industry and academic attention, no evidence has been presented to substantiate the claimed benefits of SOA. Therefore, empirical studies are needed to evaluate the value of SOA.

This paper reports on a first attempt to attain empirical evidence regarding the benefits of SOA. It is a comparison between traditional DOA using the EJB [4] technology and SOA using Web Services (WS) [3].

The main contributions of this paper are:

- An empirical investigation of SOA
- A comparison of the EJB and Web Services technologies
- A highlight of the value of tool support in implementing both technologies

The paper is structured as follows: section 2 describes the experiment in details, section 3 presents the results, section 4 discusses the results, and section 5 concludes the paper.

2. Experiment Description

This section describes the experiment in detail following the template presented by Wohlin et al**.** [1].

The first author designed and implemented the application in two technologies: EJB (DOA) and WS (SOA). It should be noted that the same design was used for both implementations to remove any design related effect on the results of the experiment.

The purpose of the experiment was to evaluate two Web applications developed using both EJB and WS with regard to time to market and size.

The authors were primarily interested in evaluating the benefits of one development methodology over the other, if any.

Two effects were studied in the experiment.

1. **Time to Market:** How long does it take to develop the constituent classes of each application? This is represented by hours of work (hrs).
2. **Size:** How large are the constituent classes of each application? This is represented by the number of line of code (LOC).

The **subject** is a PhD student (the first author) who has OO industrial software engineering experience. It should be noted that the subject did *not* have EJB or WS industrial experience.

The **object** is a distributed software procurement repository Web application written in Java. Constituent Java classes and interfaces[1] are the sole objects of this experiment. Configuration, unit testing, integration, and deployment files are not considered.

The server-side of the system consisted of three services: ItemsService (S1), PaymentService (S2), and UsersService (S3). The client-side consisted of JSP pages, a front controller servlet, and request handlers that interact with services on the server-side.

[1] WSDL files are treated as interfaces, because they serve as the public APIs for core services.

2.1 Experiment Planning

The **independent variable** (Factor) was the development method. There were two treatments of this factor: Web Services and EJB.

There were two **dependent variables:** time-to-market and code size. A simple comparison analysis was used, with the following symbols: The time-to-market for EJB implementation is ***t1.*** The time-to-market for WS implementation is ***t2.*** The measure of size (LOC) for EJB implementation is ***s1*** and the measure of size (LOC) for WS implementation is ***s2***.

With these variables, the null hypotheses are:

- **$H0_1$**: $t1 = t2$, the time-to-market is the same for both EJB and Web Service
- **$H0_2$**: $s1 = s2$, the size of code is the same for both EJB and WS

The alternate hypotheses are:

- **Ha_1**: $t1 \neq t2$, $t2 < t1$, the time-to-market is less for WS
- **Ha_2**: $s1 \neq s2$, $s2 < s1$, the size of code is less for WS

Measurements instruments: *Time-to-market* was measured by the number of hours (hrs) spent implementing the Web application's classes. The *Size* was measured by the number of lines of code (LOC) of the Web applications' classes.

2.2 Validity Evaluation

Internal validity is concerned with confounding factors that could affect the causal relation between the independent and dependent variables [1]**.** A main threat to internal validity stems from the IDE used to build the applications. The WebSphere IDE [2] has built-in support for both EJB and WS development. The IDE's built-in support can affect the time-to-market dependent variable. The same argument applies to the LOC dependent variable because of the varying amount of generated code that different environments produce.

External validity addresses the generalization of the experiment's results to industry. External validity is reduced by the following factors:

- Only one subject participated.
- Only one application was developed.
- Only one IDE, WebSphere, was used.

2.3 Experiment Operation

The subject developed the EJB server-side and client first, then he developed the WS server-side and client.

Development times were rounded to the nearest quarter hour and entered into tables. The subject then counted the lines of each class in each application using the LOC facility of the IDE and entered them into tables. Finally, these tables were transformed into MS Excel to calculate the figures.

Java classes were broken into two categories: classes that were *generated* by the IDE (*GEN*) and classes that were *written* by the subject (*WRT*). This is an important distinction since the LOC measure tends to be high for the WS method because of the amount of generated classes.

3. Results and Data Analysis

This section uses the notation introduced in section 2.1 to perform the comparison analysis.

3.1 Server-Side Data

Due to space constraints, only the main classes are presented in the following tables. Java beans and helper classes are not shown, but their corresponding results are included in the total LOC and Time numbers.

TABLE 1 – All Server-side EJB Services

Service	Classes	LOC	Hrs	GEN	WRT
S1	10	1139	3.75	3	7
S2	8	548	1.75	3	5
S3	10	765	2.75	3	7
Total	**28**	**2452**	**8.25**	**9**	**19**

t1s => measure of time for EJB server-side = 8.25 hrs
s1s => measure of size for EJB server-side = 2452 LOC

TABLE 2 – All Server-side WS

Service	Classes	LOC	Hrs	GEN	WRT
S1	5	919	3.25	0	5
S2	3	329	1.25	0	3
S3	5	517	2.25	0	5
WSDL	3	335	3.00	3	0
Total	**16**	**2100**	**9.75**	**3**	**13**

t2s => measure of time for WS server-side = 9.75 hrs
s2s => measure of size for WS server-side = 2100 LOC

3.1.1 Comparison Analysis

The difference between the two time-to-market measurements, $Dts = | t1s - t2s | = 1.5$ hrs. The difference between the two size measurements, $Dss = | s1s - s2s | = 352$ LOC.

Null hypotheses, $H0_1$, and $H0_2$ are *rejected.* For the first alternate hypothesis, Ha_1, the opposite is *verified.* The second alternate hypothesis, Ha_2, is *verified.*

3.2 Client-Side Data

Again, due to space constraints, only the main classes are presented in the tables.

Table 3 classes are basically Java handlers (Hs) that deal with HTTP Request and Response objects. JSP files were not included here since they are exactly the same for both EJB and WS.

TABLE 3 – EJB Client Classes

Service	Classes	LOC	Hrs	GEN	WRT
Hs	9	1036	7.5	0	9
Total	**9**	**1036**	**7.5**	**0**	**9**

t1c => measure of time for the EJB Clients = 7.5 hrs
s1c => measure of size for the EJB Clients = 1036 LOC

Table 4 reports on client classes that interact with the WS. Again, JSP files were not included since they are exactly the same for both EJB and WS.

TABLE 4 – Web Services Client Classes

Service	Classes	LOC	Hrs	GEN	WRT
Hs	9	906	4.75	0	9
S1	18	1557	0	18	0
S2	14	1233	0	14	0
S3	19	1656	0	19	0
Total	**60**	**5352**	**4.75**	**51**	**9**

t2c => measure of time for WS Clients = 4.75 hrs
s2c => measure of size for WS Client s = 5352 LOC

3.2.1 Comparison Analysis

The difference between the two time-to-market measurements, *Dtc* = | *t1c* – *t2c* | = 2.75 hrs. The difference between the two size measurements, *Dsc* = | *s1c* – *s2c* | = 4316 LOC.

Null hypotheses, $H0_1$, and $H0_2$, are *rejected.* The first alternate hypothesis, Ha_1, is *verified.* For the second alternate hypothesis, Ha_2, the opposite is *verified.*

3.3 Complete Implementation Data

This section compares the two methods as a whole – server and client side.

TABLE 5 – EJB Services and Client Classes

System	Classes	LOC	Hrs	GEN	WRT
Services	28	2452	8.25	9	19
Client	9	1036	7.50	0	9
Total	**37**	**3515**	**15.75**	**9**	**28**

t1a => measure of time for EJB = 15.75 hrs
s1a => measure of size for EJB = 3515 LOC

TABLE 6 – Web Services and Client Classes

System	Classes	LOC	Hrs	GEN	WRT
Services	13	1765	6.75	0	13
Client	60	5352	4.75	51	9
WSDL	3	335	3.00	3	0
Total	**76**	**7452**	**14.5**	**54**	**22**

t2a => measure of time for WS = 14.50 hrs
s2a => measure of size for WS = 7452 LOC

3.3.1 Comparison Analysis

The difference between the two time-to-market measurements, *Dta* = | *t1a* – *t2a* | = 1.25 hrs. The difference between the two size measurements, *Dsa* = | *s1a* – *s2a* | = 3937 LOC.

Null hypotheses, $H0_1$, and $H0_2$, are *rejected.* The first alternate hypothesis, Ha_1, is *verified.* For the second alternate hypothesis, Ha_2, the opposite is *verified.*

3.4 Written Implementation

This section compares only the written portions of the two methods for both services and clients.

TABLE 7 – Written EJB Services

System	Classes	LOC	Hrs
Services	19	1836	8.25
Client	9	1063	7.50
Total	**28**	**2899**	**15.75**

t1w => measure of time for EJB written = 15.75 hrs
s1w => measure of size for EJB written = 2899 LOC

TABLE 8 – Written WS Services

System	Classes	LOC	Hrs
Services	13	1765	6.75
Client	9	906	4.75
Total	**22**	**2671**	**11.5**

t2w => measure of time for WS written = 11.5 hrs
s2w => measure of size for WS written = 2671 LOC.

3.4.1 Comparison Analysis

The difference between the two time-to-market measurements, *Dtw* = | *t1w* – *t2w* | = 4.25 hrs. The difference between the two size measurements, *Dsw* = | *s1w* – *s2w* | = 228 LOC.

Null hypotheses, $H0_1$, and $H0_2$, are *rejected.* Alternate hypotheses, Ha_1, and Ha_2, are *accepted.*

4. Discussion

As a whole, the implementation time was less for the WS method than the EJB method. However, there was a steep learning curve to learn the WS facilities within the IDE. As a result, the experiment design did not account for such time.

The LOC for the WS was almost twice that of the EJB method. The big difference in the LOC for the WS method was due to the amount of generated code by the IDE.

4.1 Server-Side Implementation

For the server-side, the EJB method was faster. The main reason for this was the time it took to generate the

WSDL files in the WS method. It took about three hours to generate these files due to the intricacies of the IDE and the successive allocation of the services' classes in the right directories. This undermines the validity of the results since they depend on the IDE and subject's experience.

The LOC for the WS method was less than that of the EJB method. The reason was the higher number of generated files in the EJB method. The IDE generated three container-related classes for each service in the EJB method, whereas only one file was generated per service in the WS method. Nevertheless, the hypothesis was verified in this case. However, the IDE generated the extra files in the EJB method; therefore this result cannot be externalized to all SOA development.

4.2 Clients Implementation

The WS method was faster. The main cause for this result was that clients need to execute several steps to find and bind to EJBs, whereas in the WS method, clients use proxy files to invoke methods directly on WS.

Although it was faster to implement the WS client-side, the LOC was five times higher than that of the EJB method. This significant difference was due to the large number of generated files in the client-side of the WS. The majority of these files implemented a proprietary serialization and de-serialization mechanism for the value objects that are passed between clients and services. In fact, we were surprised by the large number and intricacies of these generated files. Again, the IDE affected this result and it definitely cannot be externalized.

4.3 Written Implementation

Because of the large number of generated files in various stages of both development methods, we also compared just the classes that were written by the subject.

There were 28 written classes in the EJB method and 22 written classes in the WS method. It took four hours less to implement the classes in the WS method. The LOC was slightly smaller in the WS method. Based on these results, the main hypothesis of this study holds for the written classes in the object application. In other words, it took less time to implement the written classes in the SOA method and these classes were smaller in size compared to the DOA method.

The results of this particular comparison are important, because they suggest that once developers get familiar with their IDEs, they can concentrate on their core business logic rather than on the intricacies of the IDEs. The idea is that developers will write the same number of core classes regardless of the used IDE.

5. Conclusions

We found no published evidence that support the claimed benefits of SOA. As a result, this experiment attempted to verify two claimed benefits of SOA: short time to market and small size of code (LOC). Web Services were used to represent SOA and EJBs were used to represent DOA.

The experiment hypothesized that applications developed using SOA are faster to implement and smaller in size compared to those developed using DOA. The results show that, as a whole, the implementation time for the WS method (SOA) was shorter than that of the EJB method (DOA). Furthermore, they show that the LOC of the WS method was almost twice that of the EJB method. These results confirm the time-to-market part of the hypotheses, but refute the LOC part.

The results also show that when compared based on the implementation of the server-side only, the EJB method was faster to implement. However, the LOC was smaller for the WS method than that of the EJB method.

When the client-side of the two methods was compared, the results show that the WS method was faster to implement. Conversely, the LOC of the WS method was five times higher than that of the EJB method. Lastly, when only the written classes were compared in both methods, the WS method was faster to implement and it had smaller LOC. Thus, the two parts of the hypothesis were confirmed in this case.

Although the results of the experiment were mixed, the overall hypothesis was confirmed when only the written code was compared. However, the overall conclusion suffers from some validity threats as explained in section 2.2. Mainly, the choice of the IDE may have confounded some results. Therefore, replication of this experiment is needed to enable researchers to draw more accurate conclusions. Future replication should include more subjects, several IDEs, design as an independent variable, and investigation of maintainability, reliability, and coupling as dependent variables.

References

[1] C. Wohlin, P. Runeson, M. Host, M.C. Ohlsson, B. Regnell, and A. Wesslen, *Experimentation in Software Engineering—An Introduction*. Kluwer, 2000

[2] IBM WebSphere main page, http://www-306.ibm.com/software/websphere

[3] World Wide Web Consortium (W3C), http://www.w3.org/2002/ws/

[4] Ed Roman, Rima Sriganesh, and Gerald Brose, Mastering Enterprise Java Beans. Wiley, 2005

Semantic Support to Reformulate Public Services in Terms of Life Events.

Luis Álvarez Sabucedo
Depto. of Telematics
Universidade de Vigo
Email: Luis.Sabucedo@det.uvigo.es

Luis Anido Rifón
Depto. of Telematics
Universidade de Vigo
Email: Luis.Anido@det.uvigo.es

Abstract— **Resources devoted to eGovernment make it possible the soaring amount of solutions and projects available nowadays. These projects take place in administrations from different levels and provide solutions that are not driven by a common infrastructure. Main shortcomings due to these phenomena are related to lacks of interoperability and difficulties for locating desired services. As a result, the digital gap may get bigger. To overcome these drawbacks, a semantic approach based on LifeEvents, a concept presented in the paper, is proposed. By means of LifeEvents, an interoperable manner to address needs from the point of view of the citizen is developed. As they are described using OWL, powerful semantic resources are available. A process of analysis in the domain has been conducted and this turns out in a powerful ontology that may support the characterization of Life Events. The goal is to allow not only already defined LifeEvents but also those identified in the future. Even more, once LifeEvents are defined by means of an ontology, they are susceptible of being published under the form of Semantic Web Services. Thus, it is possible to enable enhancements about composition, discovery and organization, as shown in the paper, within the framework of this proposal.**

I. Introduction

Currently, we are being witnesses of a huge effort in eGovernment development. As ICTs (Information and Communication Technologies) are getting a broad use, Public Administrations (here after PAs) are getting into this tendency. But it is important to consider that eGovernment is not solely a simple replacement of technology to provide a 24/7 service. Indeed, eGovernment solutions require the reengineering of all processes involved in the public service. As a matter of fact, this technology forces PAs to orient services toward a better performance by putting the citizen in the center of all supported tasks.

The goal of an eGovernment service is to conduct a complete end-to-end solution for citizens whenever it is possible. Thus, they are aimed at improving access to government, reducing service-processing costs and providing a higher quality of service.

The development of these platforms for eGovernment services has benefits for both government and citizens. As ICT allow the access to information and services, citizens and businesses can access and interact with PAs under a 24/7 model.

Regretfully, projects may be in quite different status. It is possible to classify them into several levels according to an UN criterion[1]:

- Emerging presence (stage I). Information is presented and documents are available only for download.
- Enhanced presence (stage II). The citizen can search for documents and perform more advanced operations, but the amount of information they can submit to PAs is limited.
- Interactive presence (stage III). Interactive services are available and government officials can be contacted by email, fax and telephone.
- Transactional presence (stage IV). Two-way interaction is supported and complex services (such as taxes, fees and postal services) are available.
- Networked presence (stage V). Final level that integrates all services under ICTs platforms and support a two-way open dialog between citizens and PAs.

At an international level, we can point out several initiatives: SAGA (Standards und Architekturen in eGovernment Anwendungen)[2] in Germany, e-GIF (eGovernment Interoperability Framework)[3] in United Kingdom, ADEA (l'Agence pour le dèveloppement de l'administration électronique)[4] in France, EIF (European Interoperability Framework)[5] for European solutions, FEAF (Federal Enterprise Architecture Framework)[6] in USA, . . . Unfortunately, most of them do not provide the description of an architecture to develop solutions or standards for service definition. Therefore, the provision of support available from different platforms is not possible and even reusing solutions from external applications requires a great effort.

International organisms involved in technological development play a role in this area too. We can outline some working groups from different environments: DGRC (Digital Government Research Center) by the National Science Foundation, the working group E-Gov DSIG from the OMG (Object Management Group), OASIS (Organisation for the Advancement of Structured Information Standards), the CEN (European Committee for Standardization) and others.

A main drawback identified in the domain is the lack of interoperability among developed solutions. A data based approach allows little interaction with services developed out of its original frame. Some limitations have been also noted on accessing services. The citizen, the end user, has problems locating the wished services or, even, the proper administration they are willing to deal with. The aim is to overcome some drawbacks detected in the domain.

To overcome them, this paper presents an alternative where eGovernment services are provided using a semantic support based on Life Events, here after LEs. LEs, as discussed later on, can be seen as those situations where citizens are compelled to deal with a PA in order to fulfill an obligation (i.e., pay a fine) or to make use of a right (i.e., request a grant).

Paramount features of this support will consist in the provision of a methodology to define services for different PAs, design interoperability mechanisms and make easier the task of locating the proper service.

The key for the definition of LEs lays on the identification of problems from the point of view of the citizen. The citizen should not be aware of the administrative procedure, just of their own needs. So, a methodology and a complete environment are provided to define LEs. In this way, it will be possible to define LE from different PAs to fulfill requests from the citizen in such a way that identified shortcomings can be easily overtaken. This concept is further elaborated in section 2.

While developing this formal tool, semantics happen to play a significant role. Semantic technologies make possible features related to the provision of a common ground to build up services related to the discovery of services or the orchestration of services are possible. We needed to choose one among the currently available semantic technologies. In the present case, we decided to use OWL[7], the W3C official recommendation, to represent the information. The process to build up the entire ontology is shown in section 3.

Dealing with Semantic Web Services is also an open issue. In this context as we will see, WSDL-S[8] is used to wrap services. This technology allows us to support in a standard and cost-effective manner advanced mechanisms of discovery and interoperability. This is possible due to simplifications in the definition of services and how they are linked to each other. A more detailed review of this point is provided in section 4.

This is not the first time, Life Events are proposed in the domain. Some previous initiatives are reviewed in section 5. Finally, on section 6, some conclusions and future works are yielded.

II. Outlook on LifeEvents

With the already discussed shortcomings in mind, we suggest a new formulation of the problem. As all interaction between the citizen and PAs is driven by the exercise of a right or the fulfilling of an obligation, we propose the definition of services provided by the administration in these terms. Thus, it is possible to focus on what the citizen is requesting and not on the PA itself.

Instead of developing services in a data layer from use cases expressed in natural terms, we work in the provision of a semantic description of services in a scalar and reusable manner. To model that idea, we define the so-called *Life Event*. This is the concept we use to refer to any particular situation a citizen must deal with and requires assistance, support or license from a PA. We can consider as "life events" situations such as getting certifications, paying a fine, getting married, moving, ...

Taking this into account, we establish a semantic based definition for LEs. These elements are going to play a main role in and are expressed using semantic terms shared by the whole system. The definition of a LE includes the following items:

- Task. Title for the considered operation. Folksonomies can play an interesting role as they provide support for semi-automatic enhancements of discovering services.
- Description. High level description of the desired operation expressed in natural terms from the point of view of the citizen.
- Input documents. All operations carried out by the administration require some input document. At least, the citizen must provide a signed form in order to invoke the operation. This element is similar to *preconditions* in some environments.
- Output documents. Of course, as a result of any performed operation, the PA in charge must provide an output expressed in terms of the ontology. This information will be put together into one or several documents. This output will vary its content from the expected document (i.e., a certification, a license, ...) to information about potential failures in getting the expected document.
- Scope. We must identify the scope of the operation (local, national, international, ...) in which we want to recognized it.
- Security Conditions. The conditions for the security mechanism involved in the whole process, such as identification of both parties, cipher methods, etc.
- Cost. The amount of money to be payed, the time it will take or other penalizations involved.
- Version. Life Events can be modified and changed from one version to another.

In order to transform natural or normal services to LE expressed in the proposed terms, we must follow a simple methodology that will be illustrated by the example LE "Moving". The proposed algorithm goes as follows:

1) Identify the particular problem we are dealing with in terms of features and PAs involved. In this example, we provide support for the situation of "Moving" that happens when a citizen wishes to move from a permanent address to a new one. This will involve the city council.
2) Decompose the main problem into several different ones that may be completed in a single step, i.e., each step must produce a document meaningful for the citizen. These new problems may become in new LE as well.
   For the sake of simplicity, we describe only some of the issues a family willing to move must deal with. In our example, several subprocesses can be considered: changing the school assigned for children, changing the medical center assigned, revoke transport bonuses, ...

3) For each procedure identified in the previous step, look for the input documents, scope and cost. These must be expressed in terms of the ontology of the system. If required, the ontology will be expanded.
   In the current example, input documents may be digital identity documents, certifications of the current address and a signed request. Note that each local council can impose particular conditions, for example, it is possible to ask for a document that proves the residence in the new address, i.e., a renting contract. The cost can be estimated by the city council as it wishes. And the scope of the operation is local.
4) Identify internal partial steps that the citizen could be interested in. These steps usually involve internal documents that may carry no meaning for the citizen but are relevant for the administration. Nevertheless, the citizen will be informed about future and past steps of his request.
   In this case, the internal steps that may be identified are updating the address for the purpose of official notifications.
5) Identify all possible documents created as potential final steps of the operation.
   In our example, the output document will consist of the certification of the new address if the operation was completed successfully. Of course, the service denial is possible in circumstances such as pending fines, new address out of the limits of the city council, ...
6) Update all services and agents that may be aware of the new service.

This proposed schema is suitable for the eGovernment field for several reasons: all operation require some input documents, the most common output in the service is a new document, there is no need (opportunity) for bargaining about services, there are limits and conditions very explicit about the data managing in terms of trustability and security (non-repudiation, privacy, integrity and confidentiality), and operations do not have real time constrains.

III. Applying semantic support

So far, a tool for modeling services with no relation to any particular technology has been presented. The task now is to introduce a semantic formulation in those concepts. As a result, advances in interoperability and discovery are expected to emerge.

Our approach takes advantage of the power of OWL to express the information relevant for the system. Nevertheless, we must keep in mind that OWL is just a tool to express knowledge with all its potential and limitations.

As developing an ontology is a common task, standard methods have been defined. Among all of them, Methontology[9], a process recommended by the Foundation for Intelligent Physical Agents (FIPA)[1] to develop ontologies, has been chosen. This methodology imposes several stages and phases to construct an ontology in an organized manner. The aim of this ontology is to provide support for LEs and, hence, LE is the main class in this ontology. The main slots are name, version, description, cost, etc. as shown in the previous section. Besides, we can identify some other relevant classes (see Fig. 1):

- Citizen. It supports the characterization of the citizen itself. This class is tagged with metadata and provides properties that explain its functionalities.
- PA. This class models the behavior of Public Administrations. Every PA involved in the system must be characterized according to some properties to make it possible interaction such as name, Public PKI Key, scope, etc.
- Document. This class defines legal proofs and includes properties such as owner, issuing entity, etc. Besides, for each sort of document supported, a new sub-class is defined; thus, each new document is modeled as a new instance in the proper subclass.
- Also classes to model regions and costs are provided

In this ontology, we have reused previously proposed metadata. For example, in the task of defining the citizen, one of the main classes in the system, FOAF (Friend of a Friend)[2] has been reused. Besides, to mark documents in the system, metadata from European standards has been reused, in particular, CWA 14859[10], CWA 14860[11] and CWA 13988[12]. This is part of a general philosophy leading toward the maximum possible agreement and reusability both for the ontology and the software based on it.

Several properties have also been identified regarding to the LEs (see Table I). These ones allow the implementation of mechanisms to discover which LE can be invoked or how they can be composed. For example, by means of the property *generates*, it is possible to discover which document can be achieved as the result of the completion of a certain LE.

To increase the possibilities of the reasoner in their duties, inverse relations are defined in any case that it is possible, i.e., if provided the relation *supports*, then the property *isSupportedBy* is also defined. This is possible thanks to the fact that identified relations can be considered as transitive.

To ensure the consistency of current and future individuals in the ontology, some rules have been defined: every LE generates some Document, all LE is supported by some PA,... Of course, further details about the conformance to local or national laws regarding documentation and legal procedures are no considered at this point and further implementations of the system should take care of it.

IV. Accessing Life Events

Once the knowledge that defines the problem has been formalized, the access to LE themselves as services has to be faced. The adopted solution is to wrap LE under the form of Semantic Web Services (SWS)[13]. This makes possible the provision of a LE pool and also related services.

[1]`http://www.fipa.org/`

[2]`http://rdfweb.org/topic/FoafProject`

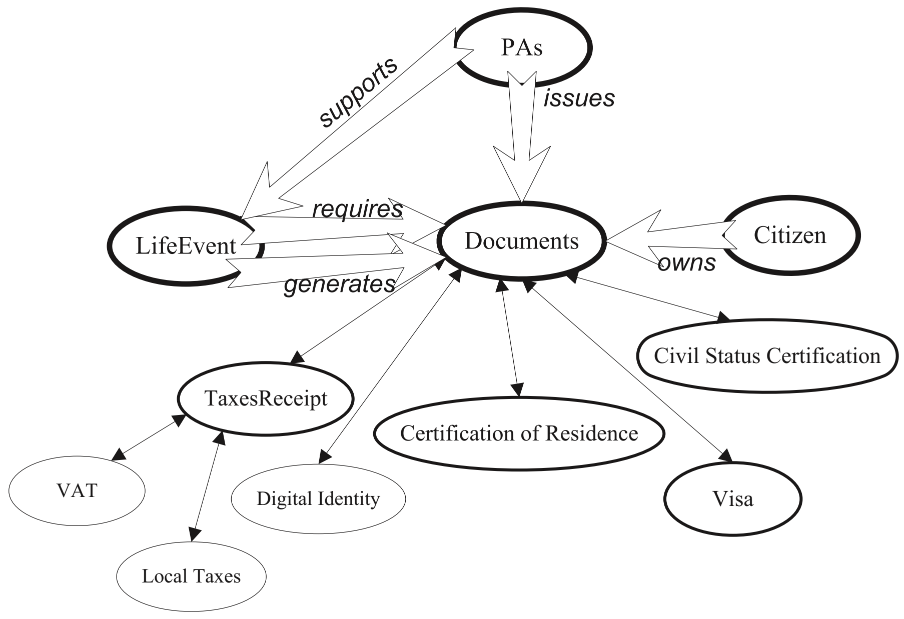


Fig. 1: Main classes and properties in the ontology

TABLE I: Properties identified in the system

Property	Domain	Range	Explanation
issues	PA	Document	Refers to which document is provided by who
requires	LE	Document	Refers to which document is required for a LE
supports	PA	LE	Refers to which LE is supported by a PA
owns	Citizen	Document	Refers to which Document is owned by a Citizen
generates	Document	LE	Refers to which Document is the output of a certain LE

The state of the art in the present moment about SWS is not steady. To meet requisites in our case, WSDL-S[8] was chosen. The motivation is its simplicity but power to express all required information. Also, as shown later on, this proposal fits quite nice in our definition of LE. Other options were rejected for different reasons. For instance, OWL-S[14] was seriously considered but it involved overhead and did not introduce any clear advantage in this particular case over WSDL-S, a much lighter technology. WSMO[15] was also considered but the use of a mediator does not really fit in our proposal.

Thus, for each LE supported in the system, a single Semantic Web Service is provided. Therefore, in each *operation* considered, the *inputs* and the *outputs* are defined in terms of the ontology developed. In particular, regarding to the LE *"Moving"*, input and output documents are the ones previously identified. Preconditions and effects supported in WSDL-S do not carry special meaning in our proposal.

It is quite simple, in a general case, to orchestrate LEs using a semantic reasoner as it only will have to link outputs and inputs expressed in terms from the same ontology and related properties identified in Table I. We must keep in mind that WSDL-S is just a tool to introduce semantics in LE. Others could be used with no structural changes. It is possible, if required, to migrate from WSDL-S to other technologies with little effort[16].

V. Related work

Similar initiatives to our proposal based on LE can be outlined. The first time the concept of LE in eGovernment was used was in the eGov project[17]. In that context, Life Events were defined as "situations of human beings that trigger public services". This approach to the concept is mainly used to model the internal process in the own PA and it was used as a guide to its implementation. Thus, it can be considered mainly as a back-office contribution.

Later on, this idea is reused in different official pages such as the Ontario's eGov site[3], Nova Scotia's one[4], the Irish eGovernment initiative [5] and many others. Those sites make use of the concept of LEs to index and locate services in a

[3]http://www.gov.on.ca/
[4]http://www.gov.ns.ca/snsmr/lifeevents/e/
[5]http://www.oasis.gov.ie/siteindex/by_life_event.html

web interface. The role performed in these cases is mainly related to facilitate the location of services to end users.

In literature, we can find some interesting initiatives, that make use, at different levels, of semantics applied somehow to this concept:

- The Finnish Web site, Suomi.fi [6], implements a taxonomy that allows a more formal classification of LE. Even, the approach is similar, the use of semantic is limited and only interaction with human users is allowed.
- The EIP.AT project[7]. It is developed in the University of Linz, Austria. Life Events and related relevant pieces of information are described using RDF support. Information retrieval mechanisms are supported by means of RDQL[18] queries.
- The SemanticGov project[19]. This ongoing project supported by the 6th Framework Program is aimed to develop a software infrastructure intended to provide support for PAs. Semantic technologies are expected to play a main role.
- The Access-eGov project[20]. An initiative, also in the 6th Frame Program, based on a peer-to-peer and service-oriented architecture that also takes advantage of semantics to improve accessibility and connectivity.

Our proposal goes a step further and suggests features not provided in these platforms such as the provision of a LEs pool from different PAs and support for its automatic management, semantic mechanisms to discover the proper LE, legal support for performed operations by means of signed documents, etc.

VI. Conclusions and future works

The main goal of this paper is the introduction of a semantic oriented model to describe eGovernment services in a formal manner. From the study of the domain and its requisites, an ontology is presented to be cornerstone of an entire system where service can be developed in an interoperable and scalable manner. This ontology is based on the use of LifeEvent as the elemental unit to drive all operations in the system.

From the use of LE, Semantic Web Services can be easily derived by using WSDL-S, a semantic technology. This lightweighted technology has been prooven to be powerful enough to support operations within the presented framework.

This proposal takes into consideration limitations and shortcomings from the technological environment, especially from the semantic environment where a lot of work is yet to be done. Nevertheless, some limitations related to the design of an accurate semantic matcher were overcome thanks to the approach selected where some simplifications are possible.

Besides, this solution bears in mind some requirements in this field:

- Third parties can develop their own agents according to the provided information.
- Flexibility is a must in this environment. This is the result of a platform that is frequently updated due to changes not just about technical issues but also about legal or administrative ones. Also, the system must be able to develop new features in a scalar way.
- Different administrations may be involved in a deal. These may use a different level of knowledge about the same citizen or require different data treatments or data models.
- Legal documentation to compel both, citizen and PAs, to fulfill operations must be created in each interaction.

This solution has been implemented under official support from Public Administration (PGIDIT06PXIB322285PR) and first prototypes are reporting promising results and they, shortly, will be at the disposal for several interested PAs.

[6]http://www.museosuomi.fi/suomifi

[7]http://eip.at

VII. Acknowledgment

We want to thank “Ministerio de Educación y Ciencia” and “Xunta de Galicia” for their partial support to this work under grants TIN2004-08367-C02-01 and PGIDIT06PXIB322285PR.

References

[1] United Nations, “Benchmarking e-government: A global perspective,” Web available, 2005, http://unpan1.un.org/intradoc/groups/public/documents/un/unpan019207.pdf.
[2] KBSt, “Saga,” Web available, 2005, http://www.kbst.bund.de/-,182/SAGA.htm.
[3] UK GovTalk, “e-GIF,” Web available, 2007, http://www.govtalk.gov.uk/.
[4] French Government, “Adea,” Web available, 2007, http://www.adae.gouv.fr/adele/.
[5] D. Enterprise and I. I. Unit, “European interoperability framework for pan-european egovernment services,” Web available, 2005, http://europa.eu.int/idabc/en/document/2319/5644.
[6] POPKIN Software, “FEAF,” Web available, 2007, http://government.popkin.com/frameworks/feaf.htm.
[7] W3C, “Web ontology language,” Web available, 2007, http://www.w3.org/2004/OWL/.
[8] W. W. W. Consortium, “Web service semantics - wsdl-s,” Web available, 2007, http://www.w3.org/Submission/WSDL-S/.
[9] Fernández-López, M., Gómez-Pérez, A, and Juristo, N., “Methontology: From ontological art towards ontological engineering.” *Symposium on Ontological Art Towards Ontological Engineering of AAAI.*, pp. 33–40, 1997.
[10] CEN, “Guidance on the use of metadata in egovernment,” Web available, 2007, http://www.cenorm.be/cenorm/businessdomains/businessdomains/isss/cwa/cwa14859.asp.
[11] ——, “Dublin Core eGovernment Application Profiles,” Web available, 2007, http://www.cenorm.be/cenorm/businessdomains/businessdomains/isss/cwa/cwa14860.asp.
[12] ——, “Guidance information for the use of dublin core in europe,” Web available, 2007, ftp://cenftp1.cenorm.be/PUBLIC/CWAs/e-Europe/MMI-DC/cwa13988-00-2003-Apr.pdf.
[13] “Semantic Web Services Interest Group,” Web available, 2007, http://www.w3.org/2002/ws/swsig/.
[14] OWL-S Coalition, “OWL-S: Semantic Markup for Web Services,” Web available, 2005, http://www.daml.org/services/owl-s/1.1/.
[15] SDK WSMO working group, “Wsmo,” Web available, 2005, http://www.wsmo.org/TR/d2/v1.1/.
[16] “D30v0.1 Aligning WSMO and WSDL-S,” Web available, 2007, http://www.wsmo.org/TR/d30/v0.1/.
[17] “egov: Online one-stop government,” Web available, 2005, http://www.egov-project.org/.
[18] A. Seaborne, “RDQL - A Query Language for RDF,” Web available, 2007, http://www.w3.org/Submission/RDQL/.
[19] “The SemanticGov Project,” Web available, 2006, http://www.semantic-gov.org.
[20] “Access-eGov,” Web available, 2006, http://www.accessegov.org/.

A Component-Based Solution and Architecture for Dynamic Service-Based Applications

Alessio Colzi, Tommaso Martini, Paolo Nesi, Davide Rogai
Department of Systems and Informatics, Distributed Systems and Internet Technology Lab (DISIT)
University of Florence, Florence, Italy, tel 0039-055-4796523, www.dsi.unifi.it, nesi@dsi.unifi.it

Abstract

Recently many new middleware frameworks are going to enforce more dynamism to their component-based models and systems. In this context, an application is build as the composition of several components/services. The main challenge is dynamic deployment and update of such components, during the application lifetime, including the discovering of the suitable components in the network and the decoupling of application code from component's nature, implementation. These features increase dynamism and the flexibility. In this paper, the main principles of a solution that allows the dynamic management, allocation and connection of remote service, including capabilities of deploying and updating of such components, during the application lifetime is proposed. The described solutions has been used to add those functionalities to a middleware based on .NET framework and has been accepted for defining the distributed services capabilities part of MPEG M3W (Multimedia Middleware) of ISO IEC23004-3, that is now becoming an official standard.

1. Introduction

The first implementations of middleware [1], [2] have been mainly realized as interoperability layers among heterogeneous computer based systems for general purpose usage. Recently, more specific middlewares have been proposed while those with general capabilities have been deeply integrated into the most diffused development platforms and/or operating systems. This trend has provoked the production of middlewares and frameworks for workflow applications, peer to peer applications, GRID solutions, industrial automation, home automation, multimedia [2], etc.

Most of the proposed component-based middleware models defines the same concept as building blocks, such as the (i) support for the reuse of components by specifying interface and contract of the offered services, (ii) portability of the framework among different platforms, (iii) interoperability of the framework by using different languages by providing "language bindings" of component access functionality, (iv) dynamic binding of component services in order to compose them in a suitable architecture, (v) exploitation of component services over a distributed environment (notification of available services, activation and invocation).

Other features can be required to acquire maximum benefits from the usage of a component-based framework for multimedia applications: real-time, robustness, security, interoperability, low-footprint, updating and trading.

Some of the most important middleware models are (in some cases, specific for multimedia): CORBA\CCM [1], DCOM, .NET, EJB, JINI, Robocop, PECOS, RUBUS, MPEG M3W [2], PECT, PBO, UPnP, OSGI, HAVi, KOALA, etc.[1], [4], [5], [6].

The main challenge is dynamic deployment and update of such components, during the application lifetime, including the discovering of the suitable components in the network and the decoupling of application code from component's nature, implementation. These features increase dynamism and the flexibility. In this paper, the main principles of a solution that address these aspects are proposed. This has been used to add those functionalities to .NET in relationships with AXMEDIS R&D IP FP6 project of the European Commission [7]. It inspired the definition of the "remote capabilities" part of MPEG M3W (MPEG Multimedia Middleware) of ISO IEC23004-3, that is now becoming an official standard.

The paper is organized as follows. In Section 2, motivations and rationales of the distributing computing model are discussed. Section 3 reports the main principles of the solution for enforcing the proposed dynamism in the component-based middlewares. In Section 4, the usages of the solution proposed in a middleware based on .NET and its adoption by MPEG M3W (explaining also how this work contributed to the standardization) are commented. An example is reported as well. Conclusions are drawn in Section 5.

2. Motivations for Distributed computing and services

In this section, the most relevant scenario is reported (it is related to the AXMEDIS project [4], and in general to network of embedded systems and PC for multimedia applications, see also MPEG M3W) as an example in which several component services, located on different devices should be usable in a transparent manner, independently from their location in the home network of multimedia devices (as shown in Figure 1). In this case, the services are published functionalities of some components put at disposal of other components in the network. In this scenario, a component should use transparently any of the multimedia components/services. In this context, the PCs may act as a service provider (as a sort of multimedia center) for the several multimedia devices that are present and connected. On the other hand, some devices can only use specific component services, deployed on and provided by other devices. Furthermore, any service/component should be updated from a third-party. Applications may reside on any device and can be developed and/or reused on different platforms; their lifetime is guaranteed by software component maintenance and update (e.g., via Internet). Component vendors can simply reuse the middleware runtime features to deliver software components to the user and the latter can treat his/her own software as a resource to be archived in a storage device for further/different usages. Applications can be developed and/or reused on different platforms; the life time of these platforms is guaranteed by the middleware runtime features (i.e., simply requesting new services to the middleware runtime, which provides retrieving and installation). The services/components can be of coding, decoding, adaptation or media processing in general.

In general, the component (see Figure 2b) groups a set of services. A service declares to implement some interfaces and models a functionality which is usable by an application. The component may also contain an additional set of information, which is relevant in order to properly use the included functionalities (e.g., documentation, formal specification of behavior, metadata, etc.).

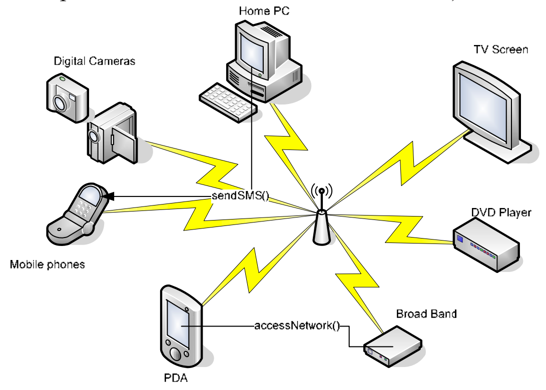


Fig. 1 – Home multimedia device network

Following the example of Figure 2b, Service1 has been created to implement InterfA, InterfB. InterfC. Application is unaware about how these interfaces have been implemented and who is providing the required functionalities. This approach opens to manage service functionality extension without any impact on the Application deployment. For example, Service1 which implements InterfX, could be extended in a second step to implement InterfY (with a richer set of methods). The Service can be easily updated, while the Application does not need to change, since it may continue to use InterfX. In this way, old and new applications can survive in the same distributed system.

In this context, it is very important to have a high dynamicity in the replacement/update of the components and in making them independent from the implementation and platform. In the following section, the achieved programming model is presented. It presents, as the most relevant feature, the *decoupling from the application code and the service implementation.*

3. Principles of the Middleware

In this section, the main architecture of the component based middleware is presented. It is mainly based on the concepts and functionalities of the:

- **Component Model,** defining the component package and how to access those implemented functionalities; The component model has been described in [8] and includes the:
  - **manifest** is a XML metadata section information including identifiers of the components, offered services, description of their interfaces, a list of dependencies for each services (for example, a service may depend on several other services for its correct execution), and all the relevant information needed by the system to manage the lifecycle of components and services.
  - **executable** code of the component.
  - **documentation:** human-readable compendium introduced since a component can be sold for integration or packaging purposes
- **RunTime Environment**, realizing a support service for the middleware capable of managing the compliant components in order to provide actual means to access their services. In Figure 2c, a simple usage flow is sketched: where an application, located on a device is using Service1 which has been deployed and installed on a remote system.

In Figure 2a, an overview of the concepts involved in the modeling of the component based framework and their relationships is given.

The decoupling of the applications from the components is possible by performing an “a-priori” standardization of the services in terms of “logical entities”, that is to define them as abstract functionality providers, as a set of implemented interfaces.

In order to access to service functionalities, the Application depends on the interfaces, which are well-known for the service standardization. The service is identified by a unique name, (e.g., namespace and name).

The Application needs also to use some support services in order to obtain access to component services. Those services, according to the object-oriented paradigm, are realized as object constructor, while in this case are referred to as "service activation". After the activation, a given instance is put in execution and maintained running by the runtime environment, and thus, it is ready to fulfill the application requests. Thus the general services for managing component services have to be exposed by the RunTime Environment of the middleware.

The typical component-based Application starts with a given service activation, that is performed "by name". The name is known by the Application as a standardized service ID (e.g., audio.mp3.MP3Decoder). The request has to be performed by the Application to the RunTime which produces an object instance that has to be cast to one of the interfaces implemented by the obtained service instance. The RunTime has the duty of finding the service in the network, making transparent the activities for the Application. In this manner, the Applications can be totally unaware about which host is providing the implementation it is exploiting. Thus, Applications can control and exploits the whole network environment.

The RunTime Environment has to support the component-model components and to avoid dependencies among (i) Application development languages, (ii) the Run Time Environment implementation and deployment, and (iii) the network topology and protocol. In addition, Applications must be capable of using the remote service instances as the local ones, even passing them as arguments for any other service method invocation.

3.1. An overview of RunTime Environment

The main functionalities/aspects of the RunTime Environment can be discussed in terms of Figure 3, and are the possibilities of: (i) decoupling from the application code and the service implementation, (ii) providing general services for managing services in a transparent manner, (iii) managing and propagating the exceptions.

The proposed solution allowed solving the requirements mentioned in the above section. In fact, any application depends only on the **IServiceBroker**, which is the gateway to any component service, and provides a class **ServiceManager** which implements such an interface. In order to allow reusing/exploiting of the running services/components, the Proxy design pattern has been applied. This allowed the Applications to work with any object which implements all the know interfaces for the related service and redirect them to IServiceBroker generic requests for a service method execution.

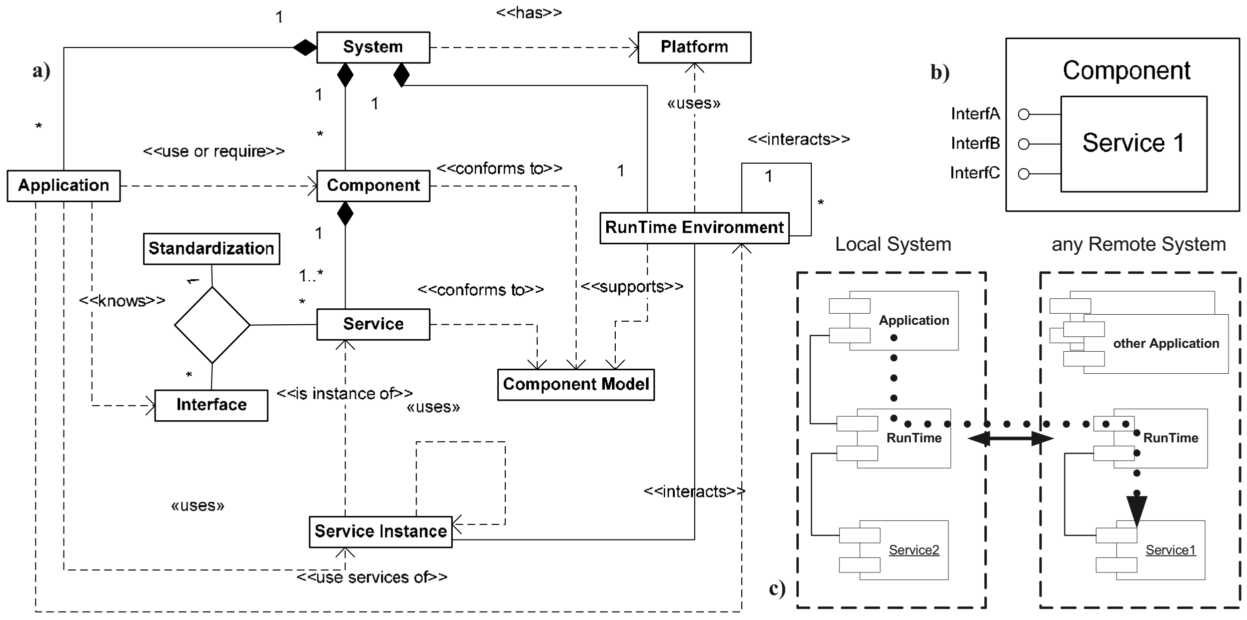


Fig. 2 – Programming model concepts: a) Entity overview b) Component c) Sharing of available services

The component manifest contains a list of the eventual other services that are needed to be activated to complete the required invocation. Other metadata information may refer to the execution profiling (e.g., computational complexity), and/or platform resource usage. All these data are readable at run-time (but also at design-time), when decision could be taken about which component/host is more suitable to fulfill a given request. Therefore, each component is enriched with a set of data attributes which are retrieved and processed by the Run Time Environment during the component exploration for service selection and connection.

In order to retrieve and use component services, the RunTime works with different framework management service/interfaces:

- **IServiceLocator** to find a service by querying the underlying distributed environment;
- **IComponentExplorer** to obtain information about a given component and its available services; any kind

of package attribute can be returned back as information accessible by all the system entities;

- **IServiceActivator,** responsible of using the component for instantiating a specific service; it returns a generic instance handle which identifies that instance for further requests;
- **IServiceHandler** to redirect a generic request to a specific invocation on the target service instance.

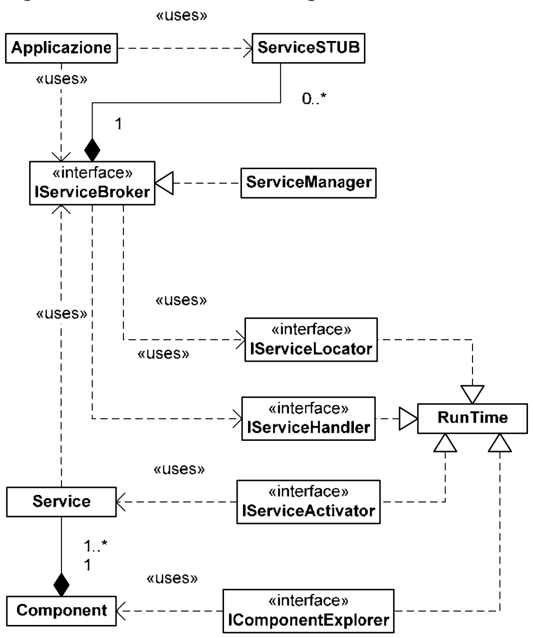


Fig. 3 –Main entities and relations

Moreover, the solution proposed is capable of propagating the service execution exceptions which could occur through any level of calling stack, without breaking neither the RunTime nor the running application. The deployment of the components which provide new services (or service updates) has to be simple, according to the philosophy of .NET framework: just copying them to a given path. In this manner, also dynamic download of new components has been easily achieved.

The RunTime is based on set of available services, and it works as a crucial service for the whole system: if the RunTime breaks, than all the services are not anymore available. Due to the dynamicity of the system the availability level of the components cannot be taken for granted. Thus, the service instance execution has to be monitored, and unexpected crashes should not stop the RunTime service in any case. Any error/exception has to be managed in order to maintain availability of the component service access functionalities. Furthermore, on the basis of the proposed delegation policy, also error information should be propagated at any system entity involved in the invocation process. With this report, many different high-level policies could take place in order to recover a service fault (e.g. instance replacement, service update).

3.2. Distributed architecture for services

This section depicts how a distributed system is deployed over the network sharing service functionalities. The same RunTime is capable to fulfill service activation and handling requests from other RunTime services. The RunTime actually covers two main roles in this distributed system:

- Unique gateway for Applications to access to all network services; it is the sole dependency of a service consumer in order to access the distributed environment;
- Peer for the service sharing network; it serves requests from other RunTimes as they were issued by any application.

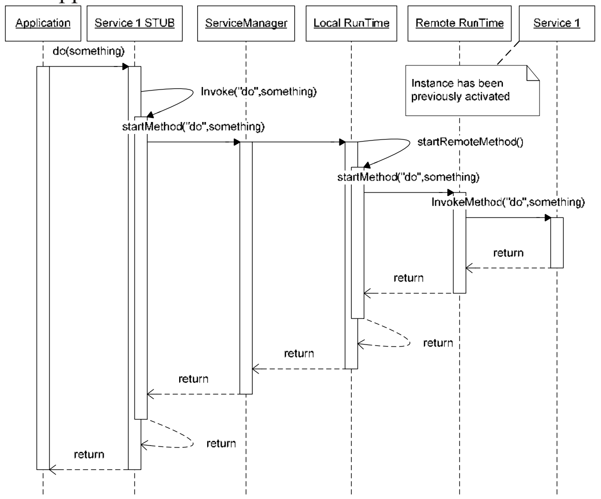


Fig. 4 – Invoking a service method on a remote platform

In Figure 4, details regarding the delegation process of serving an invocation request are shown. In this case, an Application has obtained a stub for a service instance hosted by a remote RunTime.

Please note that, the request is forwarded to the proper RunTime that finally performs the invocation on the active instance. The active service instances are located with a set of data which uniquely identify an active instance: unique ID of the instance, location i.e. the URL of the host platform (active RunTime), serviceName including its namespace. With this information any ServiceManager is capable of retrieving an active instance even if the first activation has not been commanded by it.

In Figure 5, a sequence of object diagrams is showing how a service stub, in the hands of the application code, is used as argument of another service method invocation. In Figure 5a, the starting condition is sketched: the application has already activated the instances of Service1 and Service2; the invocation is performed through Service1Stub by using the usual "dot notation" and directly passing Service2Stub as it is (i.e. *s1stub.do(s2stub)*). In Figure 5b, the ServiceManager, while handling the invocation request, extracts from

Service2Stub data regarding the localization of related Service2 instance and delivers them to the RunTime. Service2Data is the delivered information, which the RunTime can use requesting a ServiceManager to build another stub for a Service2 instance. Thus, the method implementation can also neglect that the passed argument is actually a remote active service. In fact, in Figure 5c, the RunTime is responsible of forwarding the invocation to the proper Service1 instance, and performs the opposite operation with respect to what ServiceManager did in 5b; that is: it builds a new Service2Stub (by using a ServiceManager) from the received information Service2Data and passes this stub as the actual argument of method invocation on the Service1 instance. In Figure 5d, the Service1 instance has received the Service2Stub and it will use it transparently (it has been assigned to a local reference).

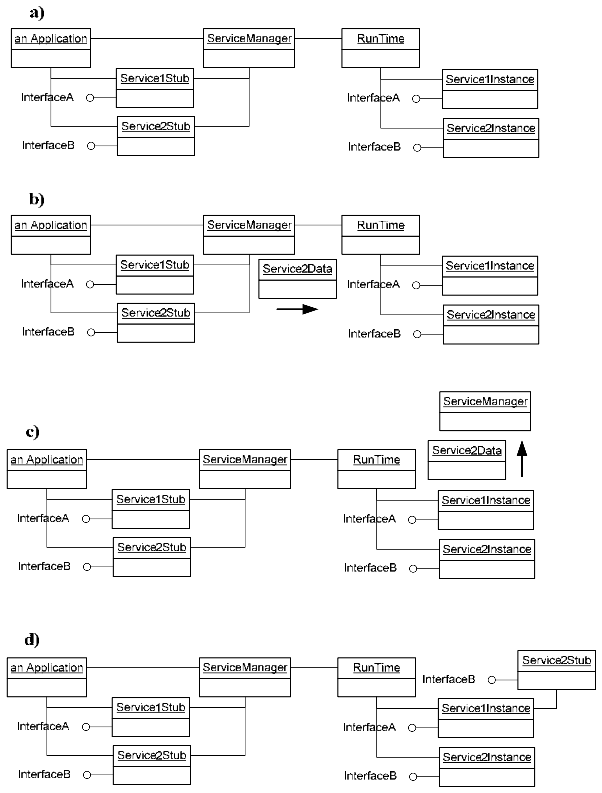


Fig. 5 – Passing a Stub as a method parameter

Please note that, this architecture enables a real transparency in term of reusing of the same application code that works for a local environment. Stubs are treated as logical references to the service instances, emulating in all aspects local references to such instances.

4. Some applications

The solution proposed has been used for enforcing dynamicity in two different middleware: .NET of Microsoft and the MPEG M3W ISO standard.

4.1. Solution Usage in .NET

The .NET Framework has been augmented with the above proposed functionalities allowing to create a more dynamic component-based middleware. The proposed implementation in .NET adds a pervasive fruition of services in a distributed environment, while .NET Remoting supports only one-to-one interactions. Components have been modeled as .NET Assemblies; they are organized at higher level for usage of their attributes. Attributes are presently used to describe exposed interfaces in terms of what has been specified by the service standardization. Also dependencies among services are reported as Assembly's attributes.

The proxy, that has been realized, decodes the invocation (method name and arguments) and subsequently forwards it to the ServiceManager.

Since .NET support "emit" feature, which is the possibility of programmatically defining a new class, it has been decided to free the application deployment from any additional dependency with respect to the used interfaces of any service. Any service stub class is generated at run-time, reading the component service description and listing the interfaces known by the application (i.e., reachable by the Application component).

In the .NET Framework, the concept of protected memory execution is defined as "application domain". Such a domain is actually managed as an independent process, while it belongs to a unique operative system process. New threads can be created inside an application domain. Since memory is protected, the communications among the application domain have to exploit network or file communication. The "execution" application Domain, where RunTime is running, is responsible of creating as many application domains as the service instances are. The communication among RunTime and its active service instance is performed with .NET Remoting. This communication support is capable to notify if an exception occurred, so that another application domain could manage the situation. In this case, the service execution never risks the RunTime for failures.

4.2. Solution Usage in MPEG M3W

The solution proposed has been also accepted by the MPEG M3W standardization activity, so the major results has been directly included in the standard edition ISO/IEC23004-3 (actually still under editing phase). Since the target platform, conceived for such standardization, is a Consumer Electronics device, many of the realization artifacts cannot be included in the defined services.

MPEG M3W includes the so called "reference software" as a compendium for understanding the information handling as depicted in the standard body text. The M3W component model is based on native binary products. The proposed concepts have been standardized in two built-in component services provides building blocks for remote invocation. Proxy service has been defined as the virtualization of the real component service on a remote platform and Wrapper service has been proposed in order to wrap any interface under a generic IReflection interface (i.e. by name invocation). DCE RPC has been used as the support communication protocol. The serialization technology has been left free in order to cope with different requirements. The standardization activity has done together with Philips and ETRI as contributors of the required technology.

4.3. An example

Among the validation tests, the following simple core experiment can be reported in the remaining document space. It has been performed on a local area network where some services have been deployed. The test has been set up on three platforms, equipped with RunTime service, with an application and two components: A component, with PrinterService implementation, is deployed on platform 2 and another, with LoggerService implementation, is deployed on platform 3.

The Application code is responsible of activating and using the required services. The sequence of the implementation instructions can be resumed as:

- gathering of the ServiceManger instance;
- activation (via ServiceManager) of a PrinterService instance;
- activation of a LoggerService instance;
- use of PrinterService instance as IPrinterSettings to specify a new text colour (*setColor(Color)*);
- use of PrinterService to print colored text (*print(coloredtext)*);;
- use PrinterService as IPrinter to print a text while logging this operation (*print(string, myLogger)*), passing LoggerService instance as second argument;
- release of active service instance by releasing local stubs (*release()*)

All the service instances are activated and managed on their platforms. Furthermore, the PrinterService instance has been a service consumer for the LoggerService, actually using another stub for the unique active instance. In Figure 7, the object diagram presents a snapshot of the three platforms status, before releasing.

5. Conclusions

The recent trend has led to the production of specific middlewares for industrial automation, home automation, multimedia, etc. In this paper, a solution to enforce in those middlewares a higher dynamism, in terms of the dynamic management, allocation and connection of remote service, deploying and updating of such components during the application lifetime, is presented. The solution has been developed on the basis of the requirements coming from MPEG M3W ISO standardization effort and to satisfy the needs of AXMEDIS IST FP6 Research & Development of the European Commission [4]. Thus, it has been accepted as valid contribution in the MPEG M3W for the definition of a component model which allows remote utilization of services and components in the standard. The solution provided is mainly based on the control and execution of services that in turn may instantiate and manage remote objects. Thus the communication among distributed objects is still possible but managed as in terms of independent services.

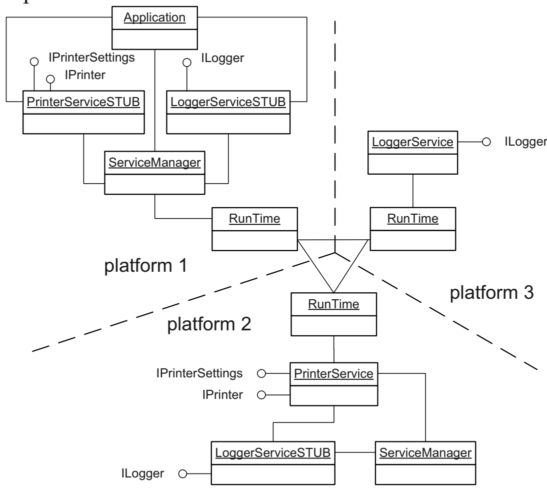


Fig. 7 – Example: platform status

6. Acknowledgements

The authors would like to thank to all MPEG M3W contributors and the AXMEDIS project partners including the Expert User Group and all affiliated members, for their contributions, funding and collaborations. A specific acknowledgment to EC DG INFSO for the partial funding.

7. References

[1] CORBA Component Model 3.0, June 2002

[2] R. Baler, C. Gran, A. Scheller, A. Zisowsky, "Multimedia Middleware for the Future Home". Proc. of the 2001 int. workshop on Multimedia middleware, Oct. 2001, Ottawa, Ontario, Canada

[3] ISO/IEC JTC1/SC29/WG11 N6835 "MPEG Multimedia Middleware: Requirements on the MPEG Multimedia Middleware V2.0", Oct. 2004.

[4] Crnkovic and M. Larsson. "Building Reliable Component-Based Software Systems". Artech House Publishers, 2002.

[5] J. Muskens and M. Chaudron. "Integrity management in component based systems". In Proc. of the 30th EUROMICRO conf., Rennes France, Aug. 2004.

[6] K. Sandstrom, J. Fredriksson, and M. Akerholm. "Introducing a component technology for safety critical embedded real-time systems". In 7th ICSE Workshop on Component-Based Software Engineering, May 2004.

[7] AXMEDIS: www.axmedis.org

[8] T. Martini, P. Nesi, D. Rogai, A. Vallotti, "A Component based Multimedia Middleware for Content Production Factory", 11th Int. Conf. on Distributed Multimedia Systems, DMS 2005, Banff Springs Hotel, Banff, Canada, Sept. 2005

Adequacy of Composite Parametric Software Reliability Models

Lance Fiondella and Swapna S. Gokhale
Dept. of Computer Science and Engineering
Univ. of Connecticut, Storrs, CT 06269
{lfiondella,ssg}@engr.uconn.edu

Abstract

Several composite parametric models have been proposed for software reliability analysis. These models enhance the classical software reliability models by explicitly incorporating the impact of testing effort on the fault detection process. The adequacy of composite models is commonly assessed using the Mean Square Error (MSE) criterion, which measures the ability of a model to explain the observed failure behavior. Many composite models perform better than the classical models on which they are based, according to the MSE. However, this goodness of fit sacrifices the parsimony of the original model and usually decreases the composite model's predictive power.

In this paper we suggest the use of the Akaike Information Criteria (AIC) to assess model adequacy. The use of the AIC is motivated by its inherent feature to penalize models based on the number of model parameters. This serves to deter against overly complicated models that may fit the observed data well, but ultimately result in poor predictions. An illustrative comparison of the different models shows that the composite models fare worse than the classical ones according to the AIC. This indicates that the predictive capability of the composite models may be poorer than the classical models. We thus conclude that several classical, simpler software reliability models still remain relevant to modeling the fault detection process during testing.

1 Introduction

The dependence of our society on the services provided by software systems continues to grow. Software systems are prevalent in several critical domains including health care, avionics, space, finance, and telecommunications, in which failures can have far reaching consequences. This heavy reliance on software systems places a significant premium on the reliable operation of these systems.

A number of models have been proposed over the past thirty years [6] for the assessment of software reliability. These models characterize the time-dependent fault detection process of a software application during its testing phase. In the recent past, some of these classical software reliability models have been enhanced to explicitly include testing effort [16, 15]. Testing effort models are based on the notion that since the number of units of effort are functionally related to the fault detection process, it may be more realistic to model the effort as a function of time and fault detection as a function of these efforts.

To apply a software reliability model, its parameters may be estimated based on the failure or/and testing effort data, either using the maximum likelihood (MLE) [7] or the least squares method [3]. The MLE method is preferred as it is asymptotically efficient at achieving the Cramer-Rao lower bound [5]. MLE works well for models with a simple form having a few parameters. However, it may face algorithmic challenges with respect to convergence for models that have a complicated form with several parameters. As a result, the MLE method has been used with reasonable success for classical models with two to three parameters. However, it cannot be applied directly to estimate the parameters of the testing effort models, since these models have at least four parameters. Thus, a hybrid approach which combines the least squares and the MLE method is followed to estimate the parameters of the testing effort models. It is from this hybrid method that the term composite[1] parametric models originates and is used to refer to the testing effort models.

The parameterized model is then used to provide quantitative estimates of different metrics such as the reliability and the failure rate. More importantly, it can be used to provide future predictions of these metrics and ultimately guide engineering decisions regarding the optimal release time. Since an important objective of model-based software reliability analysis is to guide such decisions through accurate future predictions, if a model fits the observed data well, but makes poor predictions, the utility of such a model

[1]The terms testing effort models, composite models and composite parametric models are used interchangeably in the rest of the paper.

is fairly low. Although an ideal model would both fit the observed failure data precisely and predict the future accurately, given a choice between a perfect fit and accurate predictive performance, the latter should be assigned greater importance than the former. Thus, the criteria used to assess model adequacy must emphasize their predictive capability over their ability to explain the past.

One of the prevalent criteria to assess model adequacy is the mean square error (MSE). MSE emphasizes a model's ability to fit the observed data. On the other hand, to evaluate and compare the predictive capability of the models [2], several criteria have been proposed. These include prequential likelihood ratio, u-plot, y-plot, Kolmogorov-Smirnov test, Akaike Information Criterion (AIC) [1], and predictive ratio risk [13]. Of these, the AIC is very widely accepted and has been applied in the context of software reliability engineering [17, 8]. Since the AIC penalizes bias introduced by larger number of parameters, it is considered to be an objective way to assess models with a different number of parameters. It is thus expected that AIC-guided model selection would enable better predictions.

Theoretically, the AIC may be used to evaluate only those models whose parameters are estimated using the MLE method. Practically, however, most estimation methods including the hybrid approach, share the objective of maximizing the likelihood function with the MLE method. Due to this reason, we suggest the use of the AIC to assess the adequacy of composite models whose parameters are estimated using the hybrid method. To illustrate the use of the AIC, we evaluate and compare the adequacy of two classical models, namely, the Goel-Okumoto and inflection S-shaped, against three composite models, namely, exponential, Rayleigh, and Weibull effort on two data sets. Our results indicate that the composite models fare worse than the classical models according to the AIC. The classical models may thus provide more accurate predictions than the composite models and remain relevant in modeling the fault detection process by providing valuable quantitative guidance to control and optimize the testing process.

The paper is organized as follows. Section 2 summarizes the considered models. Section 3 discusses model application. Illustrations are presented in Section 4. Section 5 offers concluding remarks with directions for future research.

2 Software reliability models

In this section we review the classical and composite parametric software reliability models used in this paper.

2.1 Classical models

An important class of classical software reliability models is based on the non-homogeneous Poisson process (NHPP) [6]. Among the several NHPP-based models [6], we considered the Goel-Okumoto [7] and the inflection S-shaped models [12] in our study for the following reasons. The GO model was chosen as it is one of the most well known and widely applied model. The inflection S-shaped model was chosen because of its ability to model a S-shaped fault detection curve which may arise due to masked faults or the learning curve of the test team [14].

The mean value function, $m(t)$, of the Goel-Okumoto model, which provides the expected number of faults detected by time t, is given by:

$$m(t) = \alpha \left(1 - e^{-\beta t}\right) \tag{1}$$

The parameters of the GO model include: α, which denotes the number of faults that will eventually be detected, and β which denotes the fault detection rate. The larger the value of β, the faster the number of faults detected will approach α, which occurs as $t \to \infty$.

The mean value function of the inflection S-shaped model is given by:

$$m(t) = N \frac{1 - e^{-\phi t}}{1 + \psi e^{-\phi t}} \tag{2}$$

where ϕ is the fault detection rate, N is the total number of latent faults, and ψ is the inflection parameter. The inflection parameter is defined for a given r as:

$$\psi(r) = \frac{1-r}{r}, \quad r > 0 \tag{3}$$

The inflection rate r provides the ratio of the number of detectable faults to the total number of faults in the application due to masking or other causes. As $r \to 1$, the inflection S-shaped model reduces to the GO model.

2.2 Composite parametric models

The general form of a composite parametric model is:

$$m(t) = a \left(1 - e^{-rW(t)}\right) \tag{4}$$

where a is the number of faults that will eventually be detected and r is the fault detection rate. $W(t)$ is the time-dependent effort function. The base model of Equation (4) is thus time dependent through the specific form of $W(t)$.

We review three composite parametric models with exponential, Rayleigh and Weibull [16, 15] testing effort functions in this section. These effort functions were chosen because they belong to the same family of models. Further, they have been used extensively in hardware reliability [10].

The mean value function when the testing effort function is exponential is given by:

$$m(t) = a \left(1 - e^{-r\alpha\left(1-e^{-\beta t}\right)}\right) \tag{5}$$

Parameters β and α are interpreted as the rate of consumption of testing effort and the total effort eventually exerted by the end of testing.

The mean value function of the composite model with the Rayleigh effort function is given by:

$$m(t) = a\left(1 - e^{-r\alpha\left(1-e^{-\frac{\beta}{2}t^2}\right)}\right) \tag{6}$$

The parameters in Equation (6) have a similar interpretation as for the exponential function in Equation (5).

The Weibull effort curve is a generalization [13] of both the exponential and Rayleigh curves and has the following mean value function:

$$m(t) = a\left(1 - e^{-r\alpha\left(1-e^{-\beta t^m}\right)}\right) \tag{7}$$

Here α is the total eventual effort, while β and m denote the scale and shape parameters respectively.

3 Model application

In this section we discuss how a software reliability model may be applied to the data collected during testing. Towards this end, we first describe the various data types, followed by the parameter estimation methods. Finally, we discuss the criteria used to assess model adequacy.

3.1 Data description

Classical software reliability models can be applied to two types of failure data, namely, time between failures data and failure count data. The former consists of occurrence times of failures, whereas, the latter consists of a count of failure occurrences in different (possibly uneven) time intervals. It is clear that the former type is more fine grained than the latter. However, when the failure data is to be augmented with testing effort, it is difficult to trace individual failures to discrete efforts without tools which systematically log effort data in homogeneous time units. Since composite models need both failure and testing effort data, to maintain consistency while applying both classical and composite models, failure count data is used here.

The failure count data consists of a vector of tuples: $\boldsymbol{t} = \{(t_1, y_1), (t_2, y_2), \ldots, (t_n, y_n)\}$, where t_i denotes the end of the i^{th} time period, while y_i denotes the number of failures observed in the interval (t_{i-1}, t_i). Furthermore, it is assumed that $y_0 = 0$ at time t_0.

To consider the testing effort explicitly, a third element is added to this vector of tuples to account for the variable testing effort. The data then is of the form $\boldsymbol{t} = \{(t_1, w_1, y_1), (t_2, w_2, y_2), \ldots, (t_n, w_n, y_n)\}$, where w_i is the effort expended in the interval (t_{i-1}, t_i).

3.2 Parameter estimation

In this section we discuss the different methods that could be used to estimate model parameters. This discussion is offered within the context of applying these methods to software reliability models.

3.2.1 Least squares method

In the least squares method, model parameters are estimated by minimizing the sum of the squares of the differences between the observed data and those computed by the model. When used to estimate the parameters of the effort function of the testing effort models, the least squares method can be mathematically represented as:

$$MSE_W = \sum_{i=1}^{n} (w_i - W(t_i))^2 \tag{8}$$

Here, MSE_W is the sum of the differences between the cumulative units of observed effort (w_i) and the number of units of effort predicted by the effort function ($W(t_i)$). Equation (8) is differentiated with respect to each parameter of the effort function. The set of equations so generated are set to zero and then solved simultaneously to produce the least squares estimates of the model parameters.

3.2.2 Maximum likelihood method

When applied to software reliability, the canonical form of the joint likelihood [6] for failure count data is given by:

$$L(\boldsymbol{\Theta}|\mathbf{t}) = \prod_{i=1}^{n} \frac{(m(t_i) - m(t_{i-1}))^{f_i}\, e^{-(m(t_i)-m(t_{i-1}))}}{f_i!} \tag{9}$$

where $\boldsymbol{\Theta}$ is the vector of model parameters, and $\mathbf{t}$ is the observed data. f_i denotes the fault count for the i^{th} interval, and is assumed to be an independent Poisson random variable with mean $m(t_i) - m(t_{i-1})$.

As the logarithm is monotonic and allows application of various identities which simplify the form of Equation (9), it is maximized for each parameter through partial differentiation of the likelihood function and solving the resulting expressions for that parameter. Typically, closed form solutions are infeasible, necessitating the use of an approximation using a numerical root finding algorithm [4], for simultaneous maximization of the joint likelihood function for all the parameters. The numerical algorithm, however, may face convergence issues especially if the expressions are complex or if the model has several parameters.

3.2.3 Hybrid method

The overview of composite models in Section 2.2 indicates that these models have four or more parameters. Thus, estimating their parameters using the MLE method is infeasible due to the convergence challenges. The following hybrid, two-step procedure is thus used to estimate their parameters. In the first step, the parameters of the effort function are estimated using the least squares method (Section 3.2). In the second step, the parameter estimates of the effort function obtained in the first step are used to estimate the parameters of the original model. Thus, by first fitting w_i with respect to t_i, it is then possible to fit y_i with respect to t_i. The rationale for this two-step procedure is that since the least squares methods provides a good fit to the effort curve it is a viable alternative to simultaneously estimating all the parameters by the MLE method.

3.3 Model assessment

Model assessment consists of evaluating how well a dataset conforms to a selected model. Of the several measures used for this purpose, we discuss the Mean Square Error (MSE) and the Akaike Information Criterion (AIC).

Minimum mean square error of the mean value function is a popular measure and is computed by:

$$MSE_{model} = \sum_{i=1}^{n} (y_i - m(t_i))^2 \qquad (10)$$

Here, MSE_{model} is the sum of the differences between the cumulative number of failures observed (y_i) and the cumulative number of failures predicted $(m(t_i))$.

An alternative method to assess adequacy is the AIC [1], given in Equation (11), which is a function of the likelihood equation and the number of model parameters, M.

$$AIC = -2 \cdot \log L(\hat{\mathbf{\Theta}}|\mathbf{t}) + 2 \cdot M \qquad (11)$$

The lower the AIC, the greater is the adequacy, and a difference of two or more in the candidate models is considered to be significant evidence that one model is better than the other in preserving the predictive power. The AIC has been used to demonstrate that the Lognormal model [8] fared better than the other software reliability models.

Since the AIC is an asymptotically unbiased estimator of the expected log likelihood, theoretically, it may be used to evaluate only those models whose parameters are estimated using the MLE method. Practically, however, most parameter estimation methods ranging from non-linear programming to genetic algorithms [11] including the hybrid method, share the objective of maximizing the likelihood equation with the MLE method. As a result, we suggest the use of the AIC to evaluate composite models, although their parameters are estimated using the hybrid method.

4 Illustrations

In this section we evaluate and compare classical and composite models based on the AIC using two data sets previously used in the context of these models. The use of prior data sets allows us to ascertain the correctness of our parameter estimates and objectively compare the results of our analysis with the earlier results [16, 15, 12, 3].

For each data set, the set of models applied are chosen based on a visual inspection of the trends in the data. The parameters of the classical and composite models are estimated using the MLE (Section 3.2.2) and hybrid methods (Section 3.2.3) respectively.

4.1 Data set 1

The cumulative number of faults detected as a function of time, shown in Figure 1, does not exhibit a S-shaped trend. Thus, it is appropriate to apply the GO model. Furthermore, as the effort function in Figure 2 only exhibits a decreasing trend on an average, the composite model with exponential effort may be suitable.

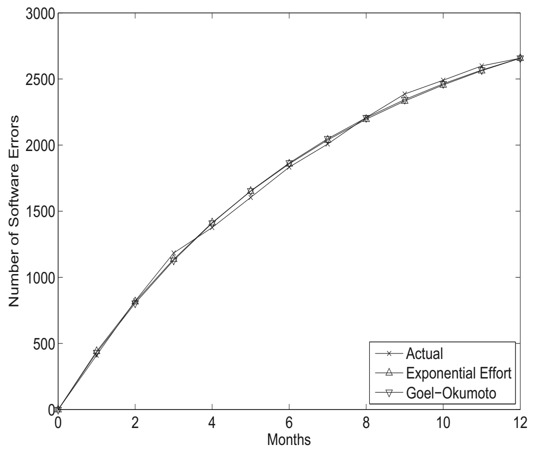


Figure 1. Faults detected vs. time (Data set 1)

Table 1 shows the MSE and the AIC of the models. The overall disagreement between the AIC for the two models is 7.46, in favor of the GO model due to its parsimony. The fluctuations in the testing effort indicate that there is alternation between testing and another activity such as bug fixing and additional test suite development. Such data describing the daily allocation of resources to various key activities would aid in the development of new models.

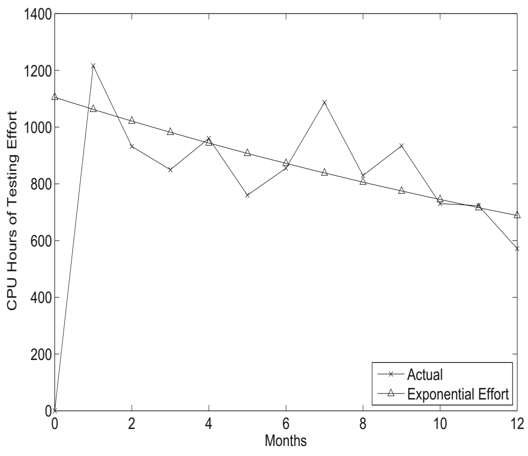


Figure 2. Testing effort vs. time (Data set 1)

Table 1. Model assessment (Data set 1)

Model	MSE	AIC
Goel-Okumoto	13354.2	168.396
Exponential Effort	13874.9	175.856

4.2 Data set 2

The fault detection curve, shown in Figure 3, exhibits the S-shaped trend, suggesting the use of the inflection S-shaped model. The effort, shown in Figure 4, initially increases and then decreases, so the composite models with Rayleigh and Weibull [13] effort functions are chosen.

Table 2 shows the MSE and the AIC for the three models. According to the AIC, the inflection S-shaped model outperforms the Rayleigh and Weibull models by 60 and 65 points respectively. The MSE of the inflection S-shaped model is the lowest. While the Weibull model has a better MSE than the Rayleigh model, the AIC unquestionably suggests that the inflection S-shaped model is closest to the underlying process. The fluctuations in the testing effort for this data set are even more pronounced than the first one.

Table 2. Model assessment (Data set 2)

Model	MSE	AIC
S-shaped	22477.1	350.537
Rayleigh Effort	63366.7	410.334
Weibull Effort	38548.2	415.248

These illustrations indicate that meaningful models cannot be built from testing effort data alone. The other key

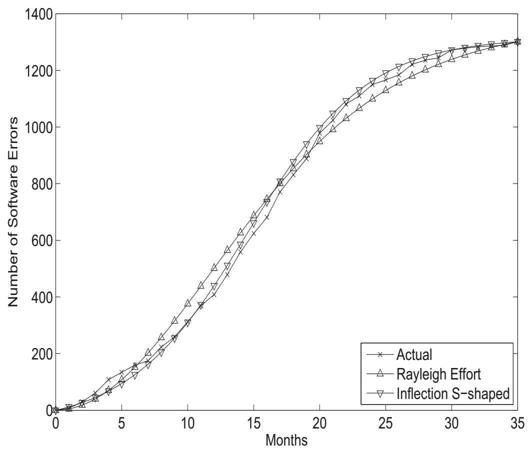


Figure 3. Faults detected vs. time (Data set 2)

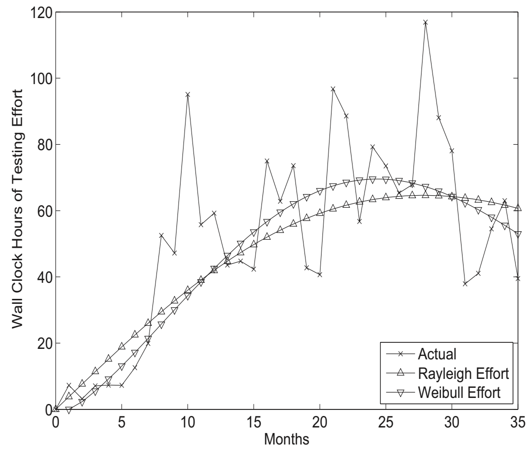


Figure 4. Testing effort vs. time (Data set 2)

activities which occur throughout testing should also be recorded. To obtain precise data, it would be necessary to implement a voluntary policy which systematically enables non-intrusive monitoring of the software process. This would enable the development and validation of simpler, more realistic models for use during the testing process. Without such detailed data, however, software reliability research must be based on overly simplified assumptions.

5 Conclusions and future research

In this paper we suggest the use of Akaike Information Criteria (AIC) to assess and compare the adequacy of composite models which incorporate testing effort into the classical software reliability models. We propose the use of the AIC since it is expected that AIC-based model selection will preserve the predictive properties of the models, because of its ability to penalize models based on the number of model parameters. Although the AIC may be theoretically applied to only those models with parameters estimated using the MLE method, we argue that from a practical standpoint it is appropriate to apply it to composite models whose parameters are estimated using the hybrid method, because the fundamental objectives of the hybrid and the MLE methods is the same, namely, maximization of the likelihood function. We demonstrate the use of the AIC in assessing and comparing composite parametric and classical software reliability models. Our results indicate that the composite models fare worse than the classical models according to the AIC. Thus, the predictions provided by the classical models may be more accurate than the composite models. The classical models thus remain relevant in modeling the fault detection process. We also provide insights into the kind of additional data that may be valuable to the development and validation of realistic software reliability models.

There exist many avenues for future research. These include: (i) considering other testing effort functions such as the log-logistic function [9], and (ii) Developing parameter estimation methods based on data mining techniques.

Acknowledgments

This research is supported in part by a CAREER award from the National Science Foundation (#CNS-0643791).

References

[1] H. Akaike. A new look at the statistical model identification. *IEEE Trans. Automatic Control*, AC-19(6):716–723, December 1974.

[2] S. Brocklehurst and B. Littlewood. *Handbook of Software Reliability Engineering*, chapter Techniques for Prediction Analysis and Recalibration, pages 119–166. McGraw-Hill, New York, NY, 1996.

[3] W. Brooks and R. Motley. Analysis of discrete software reliability models. Technical Report RADC-TR-80-84, Rome Air Development Center, NY, April 1980.

[4] R. Burden and J. Faires. *Numerical Analysis*. Brooks/Cole, 2004.

[5] G. Casella and R. Berger. *Statistical Inference*. Thomson, New York, NY, 2^{nd} edition, 2002.

[6] W. Farr. *Handbook of Software Reliability Engineering*, chapter Software Reliability Modeling Survey, pages 71–117. McGraw-Hill, New York, NY, 1996.

[7] A. Goel and K. Okumoto. Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Trans. Reliability*, R-28(3):206–211, August 1979.

[8] S. Gokhale and R. Mullen. From test count to code coverage using log normal failure rate. In *Proc. of Fifteenth International Symposium on Software Reliability Engineering*, pages 295–305, November 2004.

[9] C. Huang, S. Kuo, and I. Chen. Analysis of a software reliability growth model with logistic testing-effort function. In *Proc. of Eighth Intl. Symposium on Software Reliability Engineering (ISSRE-8)*, pages 378–388, Albuquerque, NM, November 1997.

[10] W. Kuo, V. Prasad, F. Tillman, and C. Hwang. *Optimal Reliability Design: Fundamentals and applications*. Cambridge University Press, New York, NY, 2001.

[11] T. Minohara and Y. Tohma. Parameter estimation of hyper-geometric distribution software reliability growth model by genetic algorithms. In *Proc. of Sixth International Symposium on Software Reliability Engineering*, pages 324–329, October 1995.

[12] M. Ohba. Software reliability analysis models. *IBM Journal of Research and Development*, 28(4):428–443, July 1984.

[13] H. Pham. *System Software Reliability*. Springer-Verlag, London, England, 2006.

[14] Y. Tohma, K. Tokunaga, S. Nagase, and Y. Murata. Structural approach to the estimation of the number of residual software faults based on the hyper-geometric distribution. *IEEE Trans. Software Engineering*, 15(3):345–355, March 1989.

[15] S. Yamada, J. Hishitani, and S. Osaki. Software-reliability growth with a weibull test-effort: A model & application. *IEEE Trans. Reliability*, 42(1):100–106, March 1993.

[16] S. Yamada, H. Ohtera, and H. Narihisa. Software reliability growth models with testing-effort. *IEEE Trans. Reliability*, R-35(1):19–23, April 1986.

[17] X. Zhang, X. Teng, and H. Pham. Considering fault removal efficiency in software reliability assessment. *IEEE Transactions on Systems, Man, and Cybernetics Part A*, 33(1):114–120, January 2003.

Evaluation of the OORT Techniques for Inspection of Requirements Specifications in UML: an empirical study

Tereza G. Kirner, Erik R. da Cruz
Methodist University of Piracicaba - Graduate Program in Computer Science
Rodovia do Açúcar, Km 156 - 13400-911, Piracicaba , SP - Brazil
tgkirner@unimep.br, cruz@claretianas.com.br

Abstract

The use of the Unified Modeling Language (UML) has been expanding significantly, in many companies. However, given the great amount of information generated through the diagrams, it is essential to evaluate UML models, aiming at detecting and eliminating defects that could be inserted in the system specification. One of the most efficient methods for the evaluation of software artifacts is the inspection. This work presents an empirical study on the evaluation of Object-Oriented Reading Techniques (OORTs) for the inspection of UML specifications, performed with the objective of evaluating the efficiency of such techniques as well as identifying some issues related to the professional practice.

1. Introduction

The UML (*Unified Modeling Language*) is based on concepts and notations of object orientation and its goal is to create models, supported by graphic diagrams, representative of software systems. The UML diagrams allow the production of the conceptual modeling and of the high-level system design [2]. The use of UML has been expanding, not only in the academic environment but also in companies worldwide. However, given the large amount of information generated through the diagrams and the strong inter-relationship existing among the different diagrams, it is fundamental to invest on the evaluation of UML models, in order to detect and eliminate defects that could be embedded in the system specification and design.

The main defects that may occur in UML models are inconsistencies, omissions and ambiguities, referring to a single diagram, to two or more diagrams analyzed comparatively, or to diagrams in relation to the Software Requirements Specification (SRS) document [10]. Such defects, if not identified and eliminated, will lead to failures that will impair the quality of UML models, and may result in serious damages to the quality of the developed product.

Inspection is a very efficient method for the evaluation of software artifacts, aiming at detecting defects that shall be later removed or corrected. Software inspection was initially proposed by Fagan [5], [6], being improved by other authors [7], [9]. Traditionally, the inspection process comprises meetings where groups of experts work together for the detection of defects embodied in the artifacts analyzed. Some examples of software artifacts prone to be inspected are: Software Requirements Specification (SRS) documents; requirements specifications expressed in a language or method; software coding, in a given programming language; etc.

The Object-Oriented Reading Techniques (OORTs) were proposed to inspect UML models. They include seven techniques that must be jointly utilized. Through each technique, or two UML diagrams are comparatively evaluated or one diagram is evaluated in relation to the SRS [10]. The OORT techniques have already been successfully employed (Bunde [3], Conrad [4]).

This work presents an empirical study on the evaluation of OORT techniques, performed to evaluate both the efficiency of the techniques, in terms of defect detection, and factors related to the use of such techniques by system analysts of Brazilian institutions in the region of Piracicaba, São Paulo state.

Section 2 describes the inspection process through the OORTs, presenting the main works related to the subject. Section 3 presents the empirical study. Section 4 describes the data analysis, outlining the identified results. The conclusions are presented in section 5.

2. Object-Oriented Reading Techniques

2.1. Related Work

Inspection techniques [6] have been successfully employed to identify defects in different types of software artifacts, such as SRS documents, specifications prepared through several methods (like structured analysis, object-oriented methods, etc.), system code list in different languages, etc. The inspection has shown to be efficient and effective, once it assists the detection of a great part of defects present in the artifact, with acceptable cost.

In particular, inspection techniques based on reading [1] have been used in a significant amount of projects.

Conrad [4] reports published studies on the use of these techniques, standing out several important benefits related to the reduction of project costs and increase of productivity in software development.

Presently, more and more companies are using the UML as high-level language for software specification and project. Therefore, it is of utmost importance the investment in inspection techniques adapted to reading the UML diagrams. The Object-Oriented Reading Techniques [10], make up a process of inspection for specifications modeled in UML, which already relies on some studies and experiments [3], [4].

However, there are few studies with the participation of software engineers as inspectors, once great part of the published works on inspection uses students as subjects.

An experience on the use of OORTs in the business environment was conducted at Erickson, Norway [3], using a version of the OORTs, with some adaptations considered necessary to fulfill the peculiarities of the industrial environment. The obtained results were analyzed and presented, highlighting the following conclusions: (a) the OORT techniques showed themselves efficient and assisted the detection of more than twice the number of defects than the inspection technique previously used in the company; (b) the experiment pointed out to some modifications in the OORTs that, if performed, could improve the obtained results.

The research already performed suggests the importance of developing further empirical studies, in order to evaluate the practical feasibility of the use of OORTs by software professionals in the business environment. Furthermore, it is noticed the need of improving the OORTs, aiming at getting better results from inspections of UML artifacts.

2.2. OORT Techniques

The first version of the OORTs resulted from the extension of the traditional reading techniques, to provide a process for the inspection of UML documents [12]. Later, extensions to this first version appeared [3], [4]. These works supplied the main conceptual basis for the definition of the OORTs adopted in the present research.

The inspection process through the OORTs comprises seven types of reading, classified as Horizontal Reading and Vertical Reading [10]. The Horizontal Reading aims at compare artifacts resulting from the same phase in the software development, as, for example, comparison between class diagrams and state diagrams, elaborated in the phase of logical design. The Vertical Reading aims at comparing artifacts resulting from different phases of the software development, such as requirements engineering and design. Whereas horizontal techniques are mainly directed to consistency analysis between the diagrams, vertical techniques are directed to the analysis of completeness and traceability of diagrams. Both types of reading are illustrated in Figure 1 and summarized as follows.

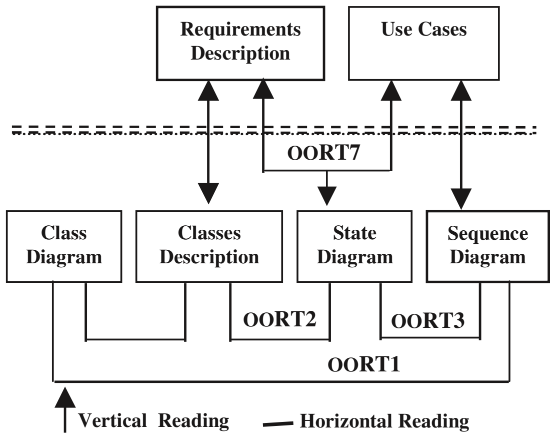


Figure 1. The OORTs and Diagrams and Artifacts related to them

- Horizontal Reading. It comprises the four readings summarized below.
  - OORT1. It analyzes a Sequence Diagram comparatively to a Class Diagram, with the goal of verifying if the Class Diagram describes the classes and their relationships, in such a way that the behavior specified in the Sequence Diagram is correctly captured.
  - OORT2. It analyzes a State Diagram in relation to a Description of Classes, with the goal of verifying if the classes are defined, so they can capture the functionality specified by the State Diagram.
  - OORT3. It analyzes a Sequence Diagram compared to a States Diagram, with the aim of verifying if every transition of state for an object can be achieved by the messages transmitted and received by that object.
  - OORT4. It analyzes a Class Diagram in relation to a Description of Classes, with the aim of verifying if the detailed description of the classes has all information needed and if the description of classes is correct.

- Vertical Reading. It comprises the three readings summarized below.
  - OORT5. It analyzes a Description of Classes comparatively to the SRS document, with the aim of verifying if the concepts and services that are described by the functional requirements are captured by the description of the classes.
  - OORT6. It analyzes a Sequence Diagram in relation to a Use Case Diagram, aiming at verifying if the

Sequence Diagram describes an appropriate combination of objects and messages that capture the functionality of the Use Case specification.

- OORT7. It analyzes a State Diagram compared to the SRS document and to the Use Case Diagram, with the goal of verifying if the State Diagram describes appropriately the states of the objects and the events that trigger changes of states, as described by the Use Case specification.

The techniques cover almost all the UML diagrams. Besides, the SRS is used to verify if the diagrams fulfill the system specification. Each technique compares at least two artifacts, aiming at detecting defects in them. It is important to highlight that the techniques presuppose that both the SRS and the Use Case Diagram are correct and free of defects.

Each technique consists on a series of steps and a checklist, which guide the inspectors in the reading of diagrams and detection of defects in them. The defects are classified in the following types: [10]:

- Omission. It occurs when one or more diagrams that should contain some concept from the software requirements do not contain a representation for that concept.
- Extraneous Information. It occurs when the design includes information that, while perhaps true, does not apply to the focused domain and should not be included in the design.
- Incorrect Fact. It occurs when a design diagram contains a misrepresentation of a concept described in the general requirements or requirements document.
- Ambiguity. It occurs when a representation of a concept in the design is unclear, and could lead the user to misinterpret or misunderstand the meaning of the concept.
- Inconsistency. It occurs when a representation of a concept in one design diagram disagrees with a representation of the same concept in either the same or another design diagram.

Considering the level of criticality, each defect is classified in:

- Serious, when it refers to a problem that impairs continuity of the reading, indicating that the system must be re-specified;
- Medium Criticality, when the defect invalidates the portion of the specification and the diagrams focused on the technique;
- Non-serious, when it does not impair the diagrams dealt with, only indicating, for example, the need for complementation.

The OORTs, as they were used in this research, are presented in detail in Cruz [8].

3. Development of the Empirical Study

The main goal of the empirical study was to answer the following questions: *Which is the level of efficiency of the OORT techniques, in terms of quantity of defects detected in software specifications prepared through the Unified Modeling Language? And which is the level of satisfaction of the specialists that inspected the UML specification through OORT techniques, regarding to important aspects involved in the application of these techniques?*

The study involved professionals with experience in software specification using UML. Given the difficulty in obtaining professionals that know and have experience in UML, the study based on a convenience sample, defined with basis on the criteria of "minimum experience of 2 years with UML". There were 7 software engineers identified who met the stipulated criteria, working in the region of Piracicaba, Brazil, which were contacted and agreed in taking part in the research.

Participants were required to inspect a Sub-System of Enrollment Control, which is part of an Academic Administration System, specified in UML. To make the consecution of the study objectives feasible, that involved the detection of defects in the UML artifacts, were injected defects representatives of all considered types, in all diagrams. In total, there were injected 47 defects, related to the types included in the adopted classification – omission, incorrect fact, inconsistency, ambiguity and extraneous information.

Each participant received the following documents for the inspection:

- Specification of the Sub-System of Academic Enrollment Control, modeled in UML, containing: Informal description of the system requirements; Use Case Diagrams; Class Diagram and Description of the Classes; Sequence Diagram; and State Diagram.
- Descriptions of the seven OORTs, each one presented through a form containing an explanation of the steps for application of the technique.
- Form for Writing down the Defects for Horizontal Reading, which allows the participants to note down: type of defect detected, keyword related to the defect, type of discrepancy, classification of defect, severity degree of defect, identification of requirement related to the defect, and additional issues related to the defect.
- Form for Writing down the Defects for Vertical Reading, which allows the participants to note down: type of defect detected, keyword related to the defect, type of discrepancy, classification of defect,

severity degree of defect, identification of requirement related to the defect, and additional issues related to the defect.

- Table of Classification of Defects: omission, incorrect fact, inconsistency, ambiguity and extraneous information.
- Report on the Use of OORT Techniques, comprising a series of questions that aimed at identifying aspects related to usability, completeness, and consistency of the OORTs.

First, a pilot test was performed with a software engineer, to verify if the documents prepared for the research were adequate. The results of such test indicated the need for performing some corrections in the documents, including: improve definitions of types of defects; and clarify the checklists included in the OORTs. Based on those results, the documents were reviewed and improved.

After the pilot test, the research was conducted. Initially, each participant was contacted by phone and, later, all of them received the research documents. After the sending of documents, individual meetings were appointed, at the workplace of the participant, for supplying of necessary clarifications about the process of inspection to be performed.

It is important to highlight that, from the 7 participants contacted, 2 did not return the forms filled in, being thus excluded from the research.

4. Analysis of Results

The data collected through the forms were tabulated and interpreted through descriptive statistics. From the 5 professionals that performed the inspection (identified as "experts"), one of them had 3 years experience in the use of UML for software specification; the other ones had more than 4 years experience. Besides that, all has already had some experience in software inspection, but none of them had used the OORT techniques.

The analysis of the results are presented as follows.

4.1. Efficiency of the OORT Techniques

The following results were identified through the performed research, including the experts who took part on it, and considering the inspected system:

- From the 47 defects injected in the artifacts, 34 (72,4%) were detected through the inspection performed by the experts.
- In three situations – using OORT1, OORT3 and OORT5, occurred the detection of 100% defects existing in the inspected artifacts.
- The higher percents of defects found occurred for classifications of "Inconsistency" (88,24%), "Ambiguity" (100%) and "Extraneous Information" (75%), also occurring a significant index for defects of "Omission" (50%). The lowest index occurred for detection of defects of the type "Incorrect Fact" (37,50%).
- A high percent of defects was detected by means of Horizontal Reading (OORT 1, OORT 2, OORT 3 and OORT 4), reaching 91,30%. The percent of defects detected through Vertical Reading (OORT 5, OORT 6 and OORT 7) was lower, reaching 54,17%.
- In a general way, the OORTs showed to be significantly efficient, once 72,4% of the defects embedded in the system specification were detected.
- Considering the seven techniques, the OORT7 was the less efficient in terms of amount of defects detected by each technique.
- Considering the five types of defects adopted, the lower index of detection of defects occurred for "Extraneous Information".
- Considering the two types of reading, the Horizontal Reading showed to be significantly more efficient than the Vertical Reading. The Horizontal Reading detected 91,30% of the defects and the Vertical Reading detected 54,17% of the defects.

4.2. Evaluation of the OORTs by the Experts

Concerning the evaluation of the characteristics of the documents and forms that made up the OORT techniques, it can be concluded that, for the performed research, including the experts that took part on it and the inspected system:

- Considering the classification of defects used, the majority (80%) of participants declared to be satisfied about the understanding propitiated by the classification. However, only 20% informed to be satisfied with the completeness of the adopted classification. Such results can suggest the need for reviewing and testing more in detail the classification, aiming at extending it in order to allow the covering of other types of defects that could occur.
- Considering the description presented to each type of defect included in the classification adopted, 80% of the participants declared to be "very satisfied", "satisfied" or with "medium satisfaction", both regarding the understanding requirement and the completeness requirement. Such result can indicate that, in a general manner, the defects are adequately defined.

- Considering the OORTs that make up the Horizontal Reading, it was evidenced that the OORT 3 (which comparatively analyzes Sequence Diagram versus State Diagram) was the one that received a lower level of satisfaction, as declared by the experts (20%, including "little satisfaction" and "no satisfaction"). Such result can indicate the need for reviewing the OORT3, aiming at improve its quality.
- Considering the OORTs that make up the Vertical Reading, it was evidenced that OORT6 and OORT7 received a lower index referring to the satisfaction of the participants, in relation to the OORT5. In a general way, the OORT7 was indicated as the one with lower index of satisfaction, by the experts. Such results can indicate the need to review and improve these OORTs, especially OORT7.
- Considering the knowledge propitiated to the experts through the participation in the research, 100% of the participants declared that their knowledge on inspection (including the use of OORT techniques) was increased. Such result is very significant and can suggest that the approach adopted in the research could be useful for the definition of training of professionals on the use of the OORT techniques.
- Considering the acceptance demonstrated by the experts in utilizing the OORT techniques in a software project, at their business environment, 40% of the participants placed themselves unfavorably to this possibility. Such result is significant and points to the need for establishing strategies of utilization of the referred techniques, in order to make the inspection process through the OORTs more acceptable. It is important to clarify that, in the research, each participant inspected all the system's specification, using the seven OORTs. A different strategy, where each professional would be responsible for the application of only one part of the OORTs, would tend to be less tiresome for the inspectors.

Additional results and discussions for the presented research are presented in [8].

5. Concluding Remarks

This work presented an empirical study on the evaluation of OORT techniques, performed with the goal of evaluating both the efficiency of the techniques in terms of detection of defects, and factors related to the utilization of these techniques by software engineers of companies in the region of Piracicaba city, Brazil.

It is important to point out that the empirical study represented an initial step for a deepest understanding of the OORT techniques, not only aiming at identifying the efficiency of these techniques in the detection of defects in specifications modeled through UML, but also aiming at identifying aspects that could be improved in these techniques.

As future work, it is important to perform a new research, involving a higher number of experts, which would make possible the analysis of results through techniques of statistical inference, which could lead to results prone to be extended to a greater scope referring to the use of OORTs.

Finally, the present work intends to give a contribution to the improvement of software quality, through the discussion of quality evaluation techniques of requirements specifications modeled in UML, and through the presentation of an empirical study on this subject, with significant results.

References

[1] Basili, V.R., Caldiera, G., Lanubile, F., Shull, F. Studies on Reading Techniques. *Proceedings of the 21th Annual Software Engineering Workshop*, Greenbelt, MD, 1996, p. 59-65.

[2] Booch, G., Rumbaugh, J., Jacobson, I. *The Unified Modeling Language User Guide*. Addison-Wesley, Reading, 1999.

[3] Bunde, G.A. *Defect Reduction by Improving Inspection of UML Diagrams in the GPRS Project*. MSc. Thesis, Agder University College, Norway, 2002.

[4] Conrad R. et al. Object-Oriented Techniques for Inspection of UML Models – An Industrial Experiment. *Proceedings of the European Conference on Object-Oriented Programming*, Darmstadt, DE, Springer-Verlag, 2003, p. 483-501.

[5] Fagan, M.E. Design and Code Inspection to Reduce Errors in Program Development. *IBM Systems Journal*, Vol. 15, Number 3, 1976, p. 182-211.

[6] Fagan, M.E. Advances in Software Inspections. *IEEE Transactions on Software Engineering*, Volume 12, Number 7, 1986, p. 744-751.

[7] Gilb, T., Graham, D. *Software Inspection*. Addison-Wesley, Reading, 1993.

[8] Kirner, T.G., Cruz, E.R., and Montebelo, M.I. Evaluation of the OORT Techniques: an Empirical Study. RTInfo - *Information Technology*, Volume 6, Number 1, 2006, p.21 – 38.

[9] Porter, A.H., Votta, L. A Review of Software Inspections. *Advances in Computers*, Volume 42, 1996, p. 40-76.

[10] Travassos, G.H., Shull, F., Fredericks, M., Basili, V.R. Detecting Defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality, *ACM SIGPLAN Notices,* Volume 34, Number 10, 1999, p. 47-56.

Adjudicator: A Statistical Approach for Learning Ontology Concepts from Peer Agents

Behrouz Far, *Member, IEEE*, Abdel-Halim Hafez Elamy, Nora Houari and Mohsen Afsharchi

***Abstract*— *We present a statistical approach for software agents to learn ontology concepts from peer agents by asking them whether they can reach consensus on significant differences between similar concepts. This method allows agents that are not sharing common ontologies to establish common grounds on concepts known only to some of them, when these common grounds are needed. The method starts with fetching positive and negative examples for a concept vaguely understood by a learner agent from the peer agents. The learner agent then uses a concept learning method to learn the concept in question. Then example objects of the candidate concept are sent back to the peer agents asking for their feedback. Peer agents evaluate the examples using two dimensional rate and weight evaluation criteria. The returned data is then tested for integrity and analyzed using analysis of variance to identify whether a statistical consensus can be achieved among peer agent with respect to the learnt concept. If such consensus exists, the learning agent can add the concept to its ontology with a higher degree of confidence. This will enhances the autonomy and improve communication and cooperation abilities among software agents.*

I. Introduction

Having a common syntax, a common semantics and a common context are necessary for communicative agents to interact and understand each other. Ontology research community tries to address issues arisen from violation or relaxation of any of the above three requirements. Conceptualization, that is identifying key concepts and their relations, is central in ontology research. Many researchers have proposed several ways of conceptualization and have devised many forms of conceptual mapping arithmetic [11] [13]. For the sake of simplicity and/or convenience many researchers in their works have assumed that it is possible to establish a common language among agents (e.g., using several variations of agent communication languages, ACL), and also the agents are provided with a complete common understanding of all the concepts they need (e.g., having a common conceptualization). In case that heterogeneity or interoperability is a requirement, many researchers assume that it is possible to have an already existing common ontology for the agents and that the agent developers can use this common ontology when designing their agents, perhaps by calling an ontology service, thus allowing for easy communication and understanding among the agents. However, the assumption of existence of a common ontology is often too strong or unrealistic. For many application domains, there is no agreement on ontology for the domain among developers. Also for many areas home brewed ontologies already exist and in many cases the potential ontologies are large, unwieldy and encompass more than what a particular agent most probably will ever need and implementing complex ontologies can also easily lead to discrepancies among implementations [1].

A recent approach is to let the agent have their individualized ontologies and provide them with learning and conflict resolution mechanisms for the concepts they need during communication. (See [6] for learning and making up a language; [13] and [1] for learning a concept). The work in [13] has focused on interactions between two agents only and single concepts and [6] was not concerned with concepts. In a previous work Afsharchi and Far have devised a methodology for having agents learn concepts from several peer agents [1]. In this method a learner agent queries peer agents by providing features (and their values) or examples that it thinks are associated with a vaguely understood concept. The queried agents provide the learner with positive and negative examples from their understanding of their own concepts (i.e. known concepts) that seem to fit the query. Then the learner agent uses a learning technique to learn the concept in question [1].

The learner agent must deal with the fact that the peer agents queried might not totally agree on which examples fit the concept and which do not. Therefore, a drawback of the above mentioned method is that there is no guarantee that the peer agents also agree upon the learnt concept. In other words, there still exists some case of misunderstanding due to the learning misses. One solution to this problem is to use a kind of voting mechanism and ask the peer agents vote on the examples for which the contradictory information is obtained. However, voting does not usually resolve the contradictions.

In this paper we have extended the work in [1] by devising a sound statistical method called adjudication that aims at resolving the contradictions among peers by questioning whether they have significantly different viewpoints regarding a concept. In the adjudication method,

This work was supported in part grants from the NSERC and IBM.

Behrouz Far, Dept. of Electrical & Computer Engineering, University of Calgary, Canada, (tel: 403-210-5411; email: far@ucalgary.ca).

Nora Houari, Dept. of Electrical & Computer Engineering, University of Calgary, Canada, (email: nhouari@ucalgary.ca).

Abdel-Halim Hafez Elamy, Dept. of Electrical & Computer Engineering, University of Alberta, Canada, (email:elamy@ualberta.ca).

Mohsen Afsharchi, Dept. of Electrical & Computer Engineering, University of Calgary, Canada, (email: mafsharc@ucalgary.ca).

after a successful learning cycle, example objects of the learnt concept are sent back to the peer agents asking for their feedback. Peer agents will evaluate the examples using a two dimensional rate and weight evaluation criteria. Their response is then tested for integrity and analyzed using analysis of variance (ANOVA) to identify whether a collective statistical consensus can be achieved among peer agent with respect to the learnt concept. If such consensus exists, the learner agent can add the concept to its ontology with a higher degree of confidence.

The structure of this paper is as follows: in Section II we give definitions for the concepts that we use throughout this paper. In Section III the concept learning mechanism is reviewed, Section IV introduces the adjudicator methodology and is followed by an example in Section V. Finally conclusions are drawn in Section VI.

II. Terminology and Definitions

In this section, we provide definitions for ontologies, concept, dimension, feature, object (example), and agent that we require for our method.

Ontology: We adopt Stumme's definition (see [12]) who defines *ontology* as a structure $O := (C, \leq_C, R, \sigma, \leq_R)$. Where C and R are two disjoint sets and the members of C are called *concept identifiers* and the members of R are called *relation identifiers*. $\leq_C$ and $\leq_R$ are partial orders on C and R, respectively called *concept hierarchy* or *taxonomy* ($\leq_C$), and *relation hierarchy* ($\leq_R$). $\sigma : R \rightarrow C^+$ is a function providing a signature for a relation.

Concept, dimension, feature, object: Many works in databases and machine learning define concepts as collections of objects that share certain feature instantiations. We assume that there exists a set of features $F = \{f_1, f_2, \ldots, f_n\}$ and a subset of F is common among agents. This means that the agents have a minimum common ground for communication. We also characterize a concept by using its dimensions and features. Therefore a concept $c \in C$ is represented by a set of dimensions, $c = \{D_1, D_2, \ldots, D_m\}$, and each dimension is comprised of certain features, $D_i = \{f_{i1}, f_{i2}, \ldots, f_{ik}\}$. Then an object (or example) $o = ([f_1 = v_1], [f_1 = v_2], \ldots, [f_n = v_n])$ is characterized by its values for each of the features. For example a concept *software development* can have dimensions such as *modeling*, *process* and *application*. And *process* itself can be comprised of features such as *architecture*, *lifecycle* and *manageability*.

In an ontology, we assign a concept identifier to each symbolic concept that we want to represent in our ontology The relation $\leq_C$ is supposed to be associated with how concepts are defined. In the literature, taxonomies are often build using the subset relation, i.e. we have $C_i \leq_C C_j$ iff for all objects $o \in C_i$ we have $o \in C_j$. This definition of $\leq_C$ produces a partial order on C and we will use this definition in the following for the ontologies that our agents use.

Agent: in this work we view an agent Ag as a quadruple $Ag = (Sit, Act, Dat, f_{Ag})$. *Sit* is a set of situations the agent can be in, the representation of a situation naturally depends on the agent's sensory capabilities. *Act* is the set of actions that *Ag* can perform and *Dat* is the set of possible values that *Ag*'s internal data areas can have. In order to determine its next action, *Ag* uses the function $f_{Ag} : Sit \times Dat \rightarrow Act$ applied to the current situation and the current values of its internal data areas. The *Sit* set usually contains parts representing observations of other agents and of the environment the agent is in. In this paper, we assume that among the observations of an agent are all messages send by other agents since the last situation an agent was in.

III. Learning New Concepts

A goal of this part of research is to develop a method how an agent can learn new concepts for its ontology with the help of other agents. This naturally assumes that the agents do not have the same ontology, otherwise learning would not be necessary. We additionally assume that there are only some base features $F_{base} \subseteq F$ that are known and can be recognized, by all agents and that there are only some base symbolic concepts C_{base} that are known to all agents by name, their feature values for the base features and the objects that are covered by them. Outside of this base common knowledge, individual agents may come with additional features they can recognize and additional concepts they know. Agents might refer to the same such features and concepts by different names and they may have features and concepts that have the same name but are not the same. While all the ontologies used by the agents will use as taxonomy the subset-relation, agents may use different other relations in their ontologies and two agents cannot rely on the same relation identifiers referring to the same relations.

Given this setting, agents will develop problems in working together, since the common ground for communication is too narrow. Our basic idea is to have an agent *learn* required concepts with the *help* of the other agents. Due to the potential differences in the ontologies of agents, objects that are positive and negative examples for a concept will play a major role in teaching an agent a new concept. We assume that the identifying name of an object is a feature in F_{base}. We do not see this as a big limitation, since it is usually not too difficult to establish a clear identification of objects. For example, if the objects are part of the environment, pointing to a particular object is sufficient to identify it.

Although we want all agents to be able to learn new concepts, for explaining our interaction scheme we designate one agent, Ag_L, as the learner agent which wants to learn a new concept and the other agents, $Ag_1, \ldots, Ag_m$, will be its peers. Ag_L has an ontology $O_L = (C_L, \leq_C, R_L, \sigma_L, \leq R_L)$ and knows a set of features F_L. Analogously, Ag_i has ontology $O_i = (C_i, \leq_C, R_i, \sigma_i, \leq R_i)$ and knows a set of features F_i. For a concept c known to the agent Ag_i, this agent has in its data areas a set pex^c_i of positive examples for c that it can use to teach c to Ag_L. Part of Act_L are actions QueryConcept, AskClassify, Learn, and Integrate, while part of the Act_is are the actions FindConcept, CreateNegEx, ReplyQuery, ClassifyEx and ReplyClass; all with appropriate arguments. These actions form our interaction scheme as:

1. Ag_L determines it needs to know about a concept c_{goal} and performs QueryConcept("c_{goal}") to inform the peer agents about this need.
2. Each agent Ag_i reacts to Ag_L's query by:
   (a) performing FindConcept("c_{goal}"), which leads to a set of candidate concepts C_i^{cand};
   (b) selecting the "best" candidate c_i out of C_i^{cand};
   (c) selecting a given number of elements out of $pex^{ci}{}_i$, thus creating p_i;
   (d) performing CreateNegEx(c_i) to produce a given number of (good) negative examples for c_i, which we call the set n_i;
   (e) performing ReplyQuery($path(c_i)$,p_i,n_i).
3. Ag_L collects the answers ($path(c_i)$,p_i,n_i) from all peer agents and uses a learner to learn c_{goal} from these combined examples (action Learn((p_1,n_1),...,(p_m,n_m))). If there are conflicts, then it resolves them with the help of the other agents using AskClassify (resp. ClassifyEx and ReplyClass by the other agents).
4. Ag_L uses the learned c_{goal} and the collected $path(c_i)$s from the other agents to construct an ontology path C_{path} leading to c_{goal} within its ontology O_L :
   (action Integrate($path(c_1)$,...,$path(c_m)$)).

The result of this learning/teaching scheme is the description of c_{goal} in terms of Ag_L's feature set F_L and an updated ontology $O^{new}{}_L = (C^{new}{}_L\ ,\leq C,\ R_L,\ \sigma_L,\ \leq R_L)$. Ag_L will also create a set $pex^{cgoal}{}_L$ in case another agent wants Ag_L to teach it c_{goal}. Details of each step are explained in [1].

IV. Adjudicator Methodology

The Adjudicator method aims at identifying and resolving conflict among peer agents (Ag_is) regarding a learnt concept (c_{goal}) using a sound procedure to verify whether the differences in the viewpoints of the peers are statistically significant or not. Figure 1 shows the flow of the method.

In the Adjudicator method, after a successful learning cycle, objects representing the learnt concept are sent back to the peer agents asking for their feedback in the form of a two dimensional rate and weight data. This is after a careful consideration of parameters such as the number of peers and number of replicas. Therefore each query is backed up by a *statistical experiment*. Then response collected from peer agents is tested for integrity and analyzed using analysis of variance (ANOVA) for each dimension of the concept to identify whether a collective statistical consensus exists among the peer agent with respect to that particular dimension of the learnt concept. The same process will be repeated for all dimensions and the goal is to seek consensus for all dimensions.

In order to obtain statistically valid results, besides specifying the evaluation data, we require:

(1) Selecting a proper number of agents to be asked for their feedback. For instance, asking 12 peer agents.

(2) Collecting several data sets (i.e. replicas) from the queried agents, for example selecting to have at least 4 data sets for each feature (or dimension) examined.

(3) Deciding upon the appropriate experiment type, for example, balanced incomplete block design (BIBD) [8] [9].

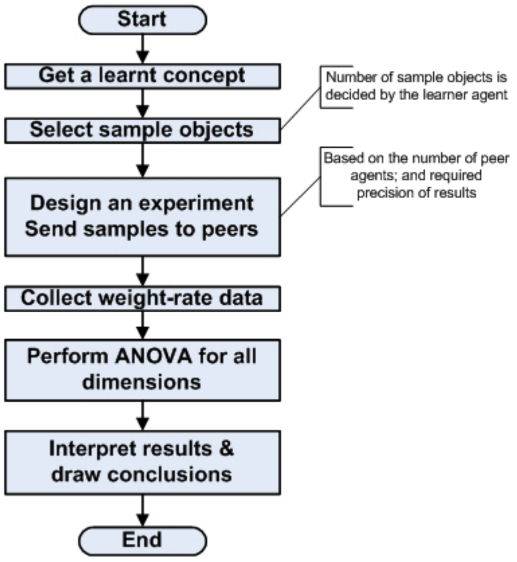

Start
Get a learnt concept
Number of sample objects is decided by the learner agent
Select sample objects
Based on the number of peer agents; and required precision of results
Design an experiment
Send samples to peers
Collect weight-rate data
Perform ANOVA for all dimensions
Interpret results & draw conclusions
End

Figure 1. Flow of the Adjudicator method

Below we will give answer to the following questions: (1) What data should be collected from peer agents? (*Section A*) (2) How many objects should be selected and how many agents should be asked to provide feedback? In other words, what shall be the target population? (*Section B*) (3) How many replicas are needed? And what shall be the appropriate statistical model? (*Section C*). Finally, we present the details of statistical validation procedure (*Section D*).

A. Evaluation model and data collection

We have devised a *Multidimensional Weighted-Attributes Framework (MWAF)* based on which the peer agents can evaluate the example objects. The idea of *MWAF* is to use the most common and important criteria (or dimensions) and their features (or attributes) of an example for a concept being evaluated.

Dimensions: the framework contains a number of dimensions, each of which represents one of the major evaluation criteria.

Attributes: are the different features pertaining to each criterion (i.e. dimension) to describe it.

Parameters: the values that are given to measure the attributes.

The reason of partitioning features into dimensions is twofold. First, it allows for hierarchical decomposition of the concept; and second, it may show common grounds for conflicts. It is possible that the all peer agent agree on certain features and disagree on a few. Analysis of dimensions will reveal which features are commonly agreed upon and which are not.

The peer agents are asked to provide two parameters for each of the features: a *rate* and a *weight*. Rate reflects the extent the feature is present in the concept. For example, whether *color* is a feature for the concept *citrus* or not. Rate is an objective parameter because it is measured according to the degree of availability or effectiveness of the examined property. Weight, on the other hand, is a subjective parameter that reflects the extent that the peer agent thinks this feature is important to the concept being evaluated. For example, whether *color* is a necessary feature for the concept *citrus* or not. The values given to these two parameters can be binary. However, for the sake of generality of the method and providing better precision we have assumed them to be numeric with the range from 0 to 10. A value of '0' implies full absence of the attribute, whereas a value of 10 reflects its maximum availability and strength.

Note that all available voting mechanisms merely collect data for the rate only. A unique feature of our method is to combine the rates and weights in order to differentiate between the *must have* and *nice to have* features of a concept.

B. Identifying sample set

Kitchenham et al [7] define a target population as the groups or individuals to whom the final results are applicable. In this work we can identify a two dimensional population: (1) a set of treatments (i.e. objects) that represent a concept; (2) and a set of peer agents Ag_is that are queried.

Ideally, a set of objects which can be evaluated by Ag_is to provide a valid set of observations are viewed as a representative subset for the learnt concept. The term "representative" is very critical, because if we do not have a representative set, we cannot claim that the final results can be generalized. Therefore identifying a proper subset of objects that represent the concept is crucial. In this work we assume that the representative sample set is selected based on the learner agent's interest, and not at random from a large number of objects. In fact, this set can be selected by identifying objects that might have caused problem during the learning phase (pessimistic approach) or the objects that are best fit for the learnt concept c_{goal} (optimistic approach).

The set of peer agents includes but may not be limited to those who were queried at the first place. The minimum number of agents to be queried depends on the type of experiment (see below) and the number of replicas.

C. Selecting variables and statistical model

Before selecting the model, or even setting up the statistical hypotheses that describe the goal, it is important to define the experimental variables and the appropriate scale of measurement [4]. In our study, the effectiveness of the features, as described by the weights and rates given to each dimension, is the dependent variable (i.e. response). In order to be tested, a dependent variable is usually quantitative and measurable. The objects and peer agents are on the other hand independent variables that influence and regulate the response; these variables are discrete in their nature and work through a nominal scale (categorical). When applying analysis of variance models, the independent variables are usually called factors or treatments.

For the sake of statistical validity each query should be tied to an appropriately designed experiment. There are several ways to construct an experiment including complete randomized design (CRD) [10], randomized complete block design (RCBD) [10] [3], balanced incomplete block design (BIBD) [8] [9]. The experiment design is decided upon based on (a) number of objects queried; (b) number of peer agents; and (c) number of replicas required.

D. Statistical validation procedure

For the statistical procedure we use the analysis of variance (ANOVA) model which is a robust and adaptable technique that is usually used to study the relationship between a dependent variable and one or more independent variables [5]. By using ANOVA, we can also identify the sources of variability among one or more potential sources. The basic underlying idea of ANOVA is to compare the variability of the observations between groups to the variability within groups. In Adjudicator method, ANOVA is used to test for the significant differences in the mean effectiveness of each dimension, D_m, among the example objects, such that:

- *Null hypothesis (H_o)*
  There is no significant difference in the mean effectiveness of the examined dimension among the evaluated objects, i.e., all the treatment effects are the same and a consensus is achieved for this particular dimension.
- *Alternative hypothesis (H_a)*
  There is a significant difference in the mean effectiveness of the examined dimension among the evaluated objects, i.e., the treatment effects are not the same and no consensus exists for this particular dimension.

ANOVA depends on the weight-rate data collected from the peer agents and in order to trust the results obtained from the ANOVA we shall check the model for aptness by running three tests, indicated in Table 1, to examine the assumptions involving the adequacy of the ANOVA procedure and to detect serious departures from the conditions assumed by the model [9]. Failure of any of the tests indicates that the collected data is not appropriate to validate the hypotheses.

Table 1. Model aptness tests

Test #	Test Type	Instrument Used
1	Outliers	a. Normal probability plot of residuals b. Individual value plot of residuals versus independent variable
2	Normality of residuals	Normal probability plot of residuals
3	Homogeneity of variances	a. Residual plots against fitted values b. Bartlett's test

The validation process is repeated for all dimensions and we can only assume a full consensus among the peer agents exist if the H_o holds for all dimensions. Unfortunately in many cases, it is hard to achieve such consensus. If the test result is significant, we can go further to carry out multiple pairwise comparisons of the objects, using Tukey's HSD (honestly significant differences) method [14] to identify which pairs of objects are significantly different from the others. In this way we can pin point the candidate objects that are causing problem and possibly remove them.

V. Example

In this example we created a test-bed and examined building conceptualization for 3 sets of software projects. We organized the entire projects documents in 3 repositories, one for each set. The sets were comprised of 110 software projects developed by software engineering students over 4 years period in the graduate and undergraduate courses. The sets were comprised of projects with concentration on software testing; concentration on object oriented analysis and design; and concentration on agent-based software development. We devised 12 agents, to serve as Ag_i peer agents, each assigned to a set of projects for one concentration in one year. We also designed an Ag_L learner agent by assigning to it random selection of projects from all three repositories. The conceptualization for each agent was arbitrary and they only shared a subset of features F and common concepts C as specified in Section III.

The learner agent Ag_L used the learning mechanism (Section III) to learn a new concept that we call it "development methodology".

The learner agent Ag_L wants to verify whether a consensus exists among 12 peers (Ag_is) queried with respect to this concept, using the Adjudicator method. The set pex^{cgoal}_L includes 9 objects that are all selected to be queried. The learnt concept has 3 dimensions (i.e., *modeling*, *process*, and *application*) and 14 features. In this experiment we decided to have at least 4 replicates which means that 4 data sets are collected for each of the 9 objects.

Giving this input set, for one-way ANOVA procedure using a CRD model requires that i=36 which means that we many need to query 36 peer agents so that each peer will receive one object at random to evaluate it, and each object will be evaluated by four peers. However, we have two limitations. We believe that there is heterogeneity among peers for many reasons that can potentially contribute to creating some sort of variability in data. Also another constraint is that we have only 12 peers. One way to overcome this lack is to make use of each peer to assess more than one object. This implies using a Randomized Complete Block Design (RCBD) where each agent should assess a complete block, i.e. 9 objects, or using the Balanced Incomplete Block Design (BIBD) model [14] in which a subset of objects (3 in this case) will be sent to each peer. In this experiment we decided to go for BIBD and query the peer agents to provide the evaluation data for 3 objects each.

In order to determine whether significant differences exist between the peers' viewpoints with respect to objects, we repeated the experiment for the 3 dimensions that characterize the 9 objects. We could classify all 14 features into one dimension but we decided to cluster them into 3 dimensions each representing a specific criterion. The following set of hypotheses describes this strategy in a statistical fashion that will be applied to all the examined dimensions.

- ***Null hypothesis,*** H_o: There is no significant difference in the mean effectiveness of the examined dimension among the evaluated objects.
- ***Alternative hypothesis,*** H_a: There is a significant difference in the mean effectiveness.

We analyzed the collected data by means of applying ANOVA procedure to the BIBD model to test the significant differences in the mean effectiveness of each individual dimension. In this way, if significant differences are ascertained, we go further to perform pairwise comparisons to identify which pairs of objects significantly differ from which ones. On the other hand, if the overall ANOVA test is insignificant, we will not apply any pairwise comparisons. In such a case, the conclusion to be made is that all the objects are statistically equal in their main effects for the examined dimension. The detailed steps of analysis were:

Step 1: Data Abstraction and Formulation

We extract the necessary data for this dimension from the collected raw data. Thus, we multiply recorded rates by the corresponding average weights of relevant attributes.

Step 2: Constructing the BIBD tableau

By adopting the BIBD arrangement we construct the BIBD tableau of this dimension.

Step 3: Testing the ANOVA assumptions

We conducted the tests depicted in Table 1 to examine the assumptions involving the adequacy of the ANOVA procedure. The results of these tests were that all of the plots did not suggest any significant departures, either from the normality of the distribution of errors or the homogeneity of error variances. Also, there was no evidence of potential outliers; all the residuals appeared to be bounded within a 95% confidence interval, and they all fell within the acceptable range of normality [9].

Step 4: ANOVA computations and hypotheses testing

We applied the adjusted formulas of the BIBD described by Yates [14], to the data arranged in the BIBD tableau and came up with the analysis of variance components. According to [14], if the calculated F statistic (F_0) is larger than its critical value (F_{crt}), F_0 falls in the rejection region. Thus, we have sufficient evidence to reject the null hypothesis at a 95% level of significance based on the available data. In this example F_0 was larger than F_{crt} for 2 out of 3 dimensions, i.e., *modeling* and *application*.

Step 5: Identifying significant differences

In the hypotheses test we carried out in Step 4, no variability was observed for the Dimension 2 (*process*) therefore no need to continue for this dimension. However, since the test was significant for the other 2 dimensions, we went further for pairwise comparison tests to identify which objects are statically different. *Christensen* [2] and Neter [10] have exhibited several methods for pairwise comparisons, such as *Tukey'*s HSD (Honestly Significant Differences, also known as the *T*-method) and *Fisher*'s LSD. In this case, we adopted the HSD method and determined that the pairs of objects which have significantly different effects were: [O1] with [O3, O4, O5, O6, O7, O8]; [O2] with [O5, O6]; and the rest were not significantly different.

Table 2 shows a binary representation of these results. It should be noted that an intersection of '**0**' in a cell implies that the corresponding two objects, as crossed by their row and column, are **not** significantly different. That is, both objects are equivalent statistically, although they may have different means of effectiveness. On the other hand, a value of '**1**' implies that the two objects are significantly different. This clearly indicates that for Dimension 1 (*modeling*) the objects O1 and partially O2 are causing trouble and there exists considerable consensus on the rest. By further reviewing the objects we found out that the O1 and O2 were instances of two popular agent based development methodologies that use rather different modeling perspective that the rest which were examples of the conventional object oriented approaches.

It may look rather odd to find that for Dimension 3 (*application*) the overall test conducted by the ANOVA was significant, while the pairwise comparisons of means conducted by the HSD test fails to reveal any significant differences among the objects in this dimension. This exceptional case occurs because the ANOVA simultaneously considers all possible contrasts involving the treatment means, and not just the pairwise comparisons. From the agent perspective, this case implies that the peer agents could not reveal significant differences among the object in the set of attributes described by Dimension 3.

Table 2: Binary representation of the evaluation results

		O2	**O3**	**O4**	**O5**	**O6**	**O7**	**O8**	**O9**
Dimension 1: Modeling	**O1**	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**
	O2		0	0	**1**	**1**	0	0	0
	O3			0	0	0	0	0	0
	O4				0	0	0	0	0
	O5					0	0	0	0
	O6						0	0	0
	O7							0	0
	O8								0

		O2	**O3**	**O4**	**O5**	**O6**	**O7**	**O8**	**O9**
Dimension 3: Application	**O1**	0	0	0	0	0	0	0	0
	O2		0	0	0	0	0	0	0
	O3			0	0	0	0	0	0
	O4				0	0	0	0	0
	O5					0	0	0	0
	O6						0	0	0
	O7							0	0
	O8								0

Step 6: Interpretation of results

Learner agent could identify the cause for discrepancy among peers. Among 14 features and 3 dimensions only 2 were the cause of trouble and pairwise analysis revealed that two objects O1 and O2 may be removed from the learned examples set. The learning (Section III) by discarding these two objects and adjudicator method (Section IV) may be run again until no further discrepancies are found.

VI. Conclusion

We presented a methodology for improving interaction and communication among agents by letting them learn ontological concepts from each other while maintaining their own individualized conceptualization. The Adjudicator method presented in this paper provides the learner agent with a conflict resolution mechanism that goes beyond a simple voting and helps achieve consensus among peer agents with regard to a learnt concept. Therefore the Adjudicator method replaces ad-hoc voting methods and is reliable because it adopts statistical procedures and validates the concluded results within certain confidence limits.

Acknowledgment

The authors are grateful to NSERC for the Discovery Research Grant and IBM for the 2006 UIMA Innovation Award.

References

[1] M. Afsharchi and B.H. Far and J. Denzinger: Ontology Guided Learning to Improve Communication among Groups of Agents, Proc. 5th AAMAS 2006, 2006, pp. 923-930.

[2] R. Christensen: Analysis of Variance and Regression. Chapman & Hall, 1996.

[3] L. Douglass, BIOM 602: Lecture Notes, Oregon State Univ., 2004.

[4] A.H. Elamy and B.H. Far: Utilizing Incomplete Block Designs in Evaluating Agent-Oriented Software Engineering Methodologies. In Proceedings of the IEEE CCECE'05, Saskatoon, Canada, May 2005, 1412-1515.

[5] R. Fisher: The Design of Experiments, 1st ed., Oliver & Boyd, Edinburgh, 1935.

[6] K-C. Jim, C.L. Giles: Talking Helps: Evolving Communicating Agents for the Predator-Prey Pursuit Problem, Artificial Life 6(3), 2000, pp. 237–254.

[7] B. Kitchenham, and S. Pfleeger: Principles of Survey Research, Part 5: Populations and Samples. Software Engineering Notes, vol. 27, no. 5, UK, 2002.

[8] M. Liu and L. Chan: Uniformity of Incomplete Block Designs, Int'l Journal Materials and Product Technology, vol. 20, no. 1–3, 2004, pp.143–149.

[9] D. Montgomery: Design and Analysis of Experiments, 6th ed., John Wiley & Sons, Inc., USA, 2005.

[10] J. Neter, W. Wasserman, and M. Kutner: Applied Linear Statistical Models, 5th ed., Irwin, USA, 1996.

[11] L. Steels: The Origins of Ontologies and Communication Conventions in Multi-Agent Systems, Autonomous Agents and Multi-Agent Systems 1(2), 1998, pp. 169-194.

[12] G. Stumme: Using Ontologies and Formal Concept Analysis for Organizing Business Knowledge, in J. Becker, R. Knackstedt (Eds.): Wissensmanagement mit Referenzmodellen – Konzepte fur die Anwendungssystem- und Organisationsgestaltung, Physica, 2002, pp. 163–174.

[13] A.B. Williams: Learning to Share Meaning in a Multi Agent System, Autonomous Agents and Multi Agent Systems 8(2), 2004, pp. 165–193.

[14] F. Yates: Experimental Design: Selected Papers. Griffin, UK, 1970.

DALICA: Intelligent Agents for User Profile Deduction

Stefania Costantini, Leonardo Mostarda, Arianna Tocchio and Panagiota Tsintza
Department of Computer Science,
University of L'Aquila, Italy
Email: stefcost,mostarda,tocchio,panagiota.tsintza@di.univaq.it

Abstract— **In this paper we are going to discuss the potential contributions that agent technology can bring into an Ambient Intelligence scenario, related to the fruition of cultural assets. The users are located in an area which is known to the agents: in the application, the users are the visitors of Villa Adriana, an archaeological site in Tivoli, near Rome (Italy). Agents are aware of user moves by means of Galileo satellite signal, i.e., the proposed application is based on a blend of different technologies. The agents, developed in the DALI logic programming language, pro-actively learn and/or enhance users profiles and are thus capable to competently assist the users during their visit, to elicit habits and preferences and to propose cultural assets to the users according to the learned profile.**

I. INTRODUCTION

The paradigm of Ambient Intelligence implies the objective of building a friendly environment where all of us will be surrounded by "intelligent" electronic devices, and this ambient should be sensitive and responsive to our needs. A multitude of sensors and actuators are already embedded in very-small or very large information and communication technologies, and a challenging task nowadays is to identify which advantages can be gained from these technology systems. Tourism for instance is a context where old and new aspects can be melted for reaching interesting results. In fact, tourism is a growing industry and it needs to evolve according to the tourists changing features. In the past, tourists were satisfied with standardized package tours. Today, with the popularization of traveling, tourists are expecting new tour experiences that are different and authentic [10]. Several interesting works have proposed a new manner of enjoying cultural places, as technology may support more dynamic and personalized methods to conceive the fruition of cultural assets. Park et al. in [8] propose a system named "Immersive tour post". It uses audio and video technology to provide improved tour experiences at cultural tour sites. This system reproduces the vision and sounds of the historical event that occurred at the particular space. Mobile applications in a mobile-environment have been experimented by Pilato et al. in [9]. Visitors are assisted in their route within the "Parco Archeologico della Valle dei Templi" (archaeological area with ancient Greek temples) in Agrigento (Sicily, Italy) by an user-friendly virtual-guide system called MAGA, adaptable to the users needs of mobility. MAGA exploits speech recognition technologies and location detection, thus allowing a natural interaction with the user. Several other proposals can be found in the literature, exploring the integration between human-computer interaction and information presentation. The system Minerva, proposed by Amigoni et al. in [1] organizes virtual museums, starting from the collections of objects and the environments in which they must be displayed, while the DramaTour methodology presented by Damiano et al. in [6] explores a visit scenario in an historical location of Turin. Visitors are assisted by a virtual spider that monitors their behavior and reactively proposes the history of the palace in detail with a lot of funny anecdotes about the people. The systems presented above have a common characteristic: they try to improve the traditional methods to inform the visitor by means of new catchy techniques for making the human-machine interface more friendly and intuitive. But, is it possible to go beyond, towards capturing the visitors desires and expectations? A particular mechanism for capturing the visitor interest for one or more cultural assets has been presented by Bhusate at al. in [2]. Each visitor receives a PDA associated to non-invasive sensors that measure "affective" context data such as the user's skin conductance and temperature. Preferences can be also catched by asking questions directly to the user before starting the visit. This method has been adopted in the system KORE [3] where parameters such as age, cultural level, preferences in arts, preferred historical period, etc., are taken into account for "tuning" the pieces of information provided, by throwing away those useless for the user (either too difficult or too easy to understand) and delivering only data which match the user profile. The architecture of KORE is based on a distributed system composed of some servers, installed in the various areas of museums, which host specialized agents. The KORE system practically demonstrates that intelligent agents can have a relevant role in capturing the user profile by observing the visitor behavior. In this paper, we present the architecture of the MAS DALICA applied to the Villa Adriana scenario for capturing the visitors interests and enhancing their profiles. Similarly to what happens in the KORE system, each DALICA intelligent agent starts its activity with the cashing of data such as the visitors' age, preferences, cultural level and so on. Then, it captures additional data about the visitor's movements and choices, elaborates them and updates the user profile. The visitor's movements are traced by means of the Galileo satellite. The learned profile allows DALICA to offer information on the cultural assets adapted to the visitor, and to proactively propose to see those assets closer on the one hand to the visitor's physical position and on the other hand to the visitor's preferences. The related items of information are provided in an appropriate customized form. As acknowledged in Section V, the DALICA system has been developed within the CUSPIS European project. In Section II we present the scenario where DALICA has been put at work and the features of the system. Section III is dedicated to the methods through which the intelligent agents are capable to capture the visitors' interest and the monitoring capabilities of the agents. Finally, we conclude in Section IV.

II. The DALICA ontology

DALICA system plays a relevant role because spies the users during their visit, captures their habits and elaborates a profile for a customized assets fruition. Villa Adriana turns out to be the greatest villa never belonged to a roman emperor, testimony of the extraordinary level of ability caught up from the roman architecture. With a perimeter of 3 Km, it combines the best elements of the architectural heritage of Egypt, Greece and Rome in the form of an 'ideal city' [11]. For a visitor, Villa Adriana is a unique wonderful place. For DALICA, Villa Adriana is a set of Points of Interest (POI's). For "POI" we intend either a specific cultural asset or a public places like restaurants located nearby. The structure of a single POI contains the following fields:

- *Identifier*: a string identifying uniquely the POI;
- *Latitude*: the latitude of the POI defined through the Galileo satellite.
- *Longitude*: the longitude of the POI defined in the same way as the Latitude.
- *Radius*: the radius of the circle that contains the POI area.
- *Keywords*: a list of the POI characteristics like, for example, 'mosaic' if the POI contains a mosaic, or 'water' if in the POI there is a fountain or a water basin. Considering that each POI can have one or more keywords, we combined each one with a number indicating its weight in the POI description. The weights are chosen according to the relative importance (expressed as a percent value) of the POI characteristics. Clearly, this information has been provided by experts.
- *Time for visit*: is an average of the time that we suppose an user will employ for visiting the specific POI.

Keywords are important because they allow to establish the possible similarities between POIs and, consequently, to discover if the visitor is interested in a particular feature which is common to them. The POIs descriptions have been collected into an appropriate ontology (developed by the group of Artificial Intelligence and Natural Language Processing at the Dept. for Computer Science, Systems and Management of the University of Rome Tor Vergata, in the context of the CUSPIS project).

III. Constructing the User Profile

The main goal of the DALICA system is that of supporting users during their visits. For this reason, when an user starts the visit, a DALICA agent is created for spying and assisting him. Agents are destined to create and upgrade the user profile according to which is proposed the information about Villa Adriana but through which steps are they able to reach the goal? The profile are created starting from an embryonal status defined by the user before starting the visit. In fact, each visitor, at the beginning of the visit, has to book the route on an Internet site where she/he can express some preferences and choices about the service fruition. The initial profile contains some data related to the visitor's name, surname, age, job and some related to the visit that she/he intends to perform (day of the visit, starting and ending time, preferences,...). Preferences express the POI characteristics that the DALICA system should take in consideration for proposing the POIs to the users. For example, if the user declares to be interested in 'mosaic' and 'plants', the system should select for him those POIs in Villa Adriana having the above keywords with a high weight value. When the visitor starts her/his route, an intelligent agent, called User Profile One, is generated. At the staring phase, it elaborates the data coming from the user-profile stored on Internet and determines an initial fruition profile. Then, it re-elaborates the fruition profile according to new data derived from the user behaviour. New enhanced fruition profile will possibly substitute the former one while the visitor proceeds in the route. At this point, it is necessary to explain through which strategies is possible to capture the visitors interests in a scenario such as Villa Adriana, where the cultural assets are arranged in an area of 300 hectares.

A. Deducing the Visitor's Interests

Intelligent agents in DALICA are reactive, pro-active and communicative. Their are capable to percept the data coming from the environment such as the satellite coordinates or the POIs chosen by the visitor and to react appropriately. While reactivity allows the agents to adopt a specific behavior in response to the external perception, pro-activity has a main role, because the reasoning process that leads to the interests deduction is based on the correlation of several data coming from the environment, from the ontology and from some basic inferential processes. Communication capabilities intervene whenever it is necessary to send data to the visitor's PDA: e.g., the explanations of what is being seen or the list of the deduced interests or the proposed other POIs to see or the warning that the visitor is entering in a restricted area. In the rest of this section we concentrate the attention on the methods used for deducing the user interests, while in next section we present the strategies for assisting her/him during the visit and for checking her/his behavior. We divide the agent deduction process into three phases: the first one represents a basic deduction level while the second and third ones elaborate the results by concatenating the previous deductions. We starts the explanation by illustrating the algorithms concerning the first phase:

Deducing the interests based on time: This algorithm is founded on the consideration that a visitor is interested in a POI if she/he observes it for a time interval "longer" than the average time of the visit for the specific cultural asset.

The meaning of “longer” can be modulated according to the current visitor’s profile. So, if a visitor has booked a visit that lasts up to six hours the time interval for the observation will be longer than that of a visitor that booked a visit lasting for two hours.

How is it possible to determine which POI the visitor is looking at? The method is based on the Galileo Satellite. Each POI, as explained in the previous section, is identified by a circle (whose center is defined by a latitude and a longitude) and by a radius. If the visitor position (expressed in latitude and longitude and coming from the PDA) belongs to the circle related to a specific POI, we can suppose that she/he is visiting that POI. If two or more POIs are close enough to determine an intersection between their circles and the visitor is located in this intersection, then the algorithm, not being able to capture the real intention of the visitor, presumes that the visitor is interested in all those POIs. Each POI which is selected according to the visitor movements is identified by a list of keywords. The algorithm elaborates the keywords of all selected POIs and then extrapolates the most frequent ones. These keywords represent the hypothetical user interests that, once deduced, will have to be confirmed both by subsequent user behavior and by other deduction mechanisms.

Deducing the interests based on the visited POIs: This algorithm considers the POIs chosen by the user and its outcome improves when several POIs have already been visited. In fact, for each POI the algorithm extracts the keywords and the most frequent ones are asserted as “deduced interest”.

Deducing the interests based on the chosen route: If a visitor decides to follow a predefined route chosen between those proposed by the system, the agent tries to capture the visitor’s interests by studying the POIs included in the route. POIs keywords most relevant for describing the route will be selected for the next step of the deduction process.

Deducing the interests by similarity: This algorithm employs a similarity measure. In particular, the interests expressed by the visitor in the web site are matched with those in the ontology. Those in the ontology which look to be similar enough are selected as deduced interests.

Deducing the interests according to some questions: Another strategy for capturing the visitor’s interests is centered on some occasional questions about the POIs located near the visitor. The agent observes the POIs around the PDA, chooses one of them and asks the visitor’s opinion on it. A positive response such as (“Yes, I like the Odeon”) will trigger the interests deduction process.

Deducing the interests according to cultural questions: The last strategy for deducing the visitor’s interests takes into consideration the cultural level of the visitor. Some questions such as “Do you like the ancient art? Do you know what is a cavea?” are useful to determine the information level to submit to the visitor. Moreover, some parameters such as the visitor’s job and age are involved in the process. The agent compares the data acquired via the questions and via the other parameters and elaborates them in order to determine the appropriate degree of the information. We have identified for now three degrees.

Basic: It is related to a basic information level where the user prefers a superficial information on the POIs combined with details on the ancient people’s life. This level usually fits primary and secondary students and occasional visitors.

Medium: Provides more technical data on POIs and particular attention is reserved to their structure. This level fits people fond of art.

Specialized: Provides the visitor with a detailed information on POIs combined with information about the materials and techniques used to manage the cultural assets. This level is tailored to specialized students, technical people, researchers and so on.

The second deduction phase captures the results of the previous deduction algorithms and tries to compare them, with the aim of reaching a more precise user profile definition. In particular, those interests coming from the previous phase and confirmed by this second one are involved in a process that selects only the most frequent ones. In fact, each deduced interest is involved in a *interests updating process*. More precisely, each interest/keyword is associated to a weight (priority) N. For a specific deduced interest K, we have define a global evaluation function computed on the weights. In this manner, the system takes in account not only the interests more frequently deduced but also their ’relevance’ in the deduction process. Then, these interests are sent to the visitor’s PDA in order to be confirmed by her/him. Precisely, this second phase is based on the following algorithms:

Filtering the deduced interests according to the time: This filter combines the deduction of the interests based on the permanence near a certain POI and the moment when the deduction itself has been reached. In particular, this step has the objective of understanding whether a visitor remained in a specific area because interested in a POI or for some other reasons (e.g., she/he was sitting on a lawn eating a sandwich).

Combining the deduced interests: The interests deduced by the previous algorithms based on time, on visited POIs, on the chosen route and according to some questions are crossed in order to obtain a more reliable user profile definition. The interests which are confirmed will be involved in the *interests updating process*.

Using similarity for confirming the deduced interests: Reliability of the interests deduced in the previous phase is checked according to the similarity degree with those inserted in the visitor’s profile in the web site. If the similarity is greater than a prefixed threshold, the interest will be involved in the *interests updating process*.

The third phase delivers data related to the elicited interests to the visitor’s PDA. When the visitor receives the interests list, she/he can confirm either all interests or a subset of them. The selected interests are managed by the agent for updating the user profile. Moreover, the agent communicates them to a central system that manages the information for the visitor in order to propose (through the agent) data and POIs closer to her/his desires and expectations.

B. Monitoring Visitor's Behavior

Intelligent agents in DALICA are also used for monitoring the users behavior with a fixed frequency. The situations where the reactive and proactive capabilities of the agents are put at work for this kid of monitoring are at least the following.

Checkinging the forbidden areas: In Villa Adriana there are areas where visitors cannot enter. These areas are defined in the ontology and an agent monitors from time to time the visitors' movements in order to guarantee that no one violates the rules.

Monitoring the visitors route: The agent has the ability to follow the visitor that has chosen a predefined route along her/his visit. For instance, the agent is able to make the itinerary shorter or longer (by either removing or adding POIs) according to the user pace, so that the user can complete the itinerary in time.

Creating a list of POIs: When the visitor has finished the visit, the agent collects all POIs that she/he has visited and puts them in a file with texts and images. This allows the visitor to keep a reminder of his visit to Villa Adriana.

C. The DALICA Architecture

The DALICA architecture involves a MAS and a central external system. This system on the one hand acts as a "router" between the MAS and the PDA's: in fact, the MAS is presently too heavy to be directly installed on the PDA's. Thus, the MAS resides on a more powerful machine and uses the central system to exchange data with the PDA's. It receives messages from/to the agents and delivers them from/to to the PDAs of the visitors. On the other hand, the central system collects and stores data about visitors and visits for future use. In the DALICA MAS, several intelligent agents cooperate in order to support the users during their visit. The three most important agents composing the MAS are the following.

Generator Agent: The role of this agent is to automatically generate the User Profile agents when a user starts a visit. The generation process happens when PDA sends a positioning message related to a new visitor.

User Profile Agent: Acts as described before in this section. They deduce the visitors interests and monitor their behaviors.

Output Agent: Manages communications between the DALICA MAS and an external central system.

DALICA agents have been implemented in the DALI language,[4] [5] [12], an Active Logic Programming language for executable specification of logical agents. DALI is a prolog-like logic programming language with a prolog-like declarative and procedural semantics [7].

IV. CONCLUSIONS

We conclude this paper by making some considerations about our work. It is not so easy to find an application where intelligent agents are put at work in a real scenario but it is even less frequent to find intelligent logical agents at work. In the light of these considerations, the DALICA MAS is a novelty. This also because DALICA exploits the signal of Galileo Satellites to deduce the Users Profiles. DALICA at work in the area of Villa Adriana practically demonstrated that logical agents can be applied successfully for capturing the visitors habits and preferences. Our system cannot be compared with platforms such as MAGA and DramaTour where the main goal is to offer information to the visitors via specialized interfaces. DALICA mainly deduces the visitors interests and leaves the job of presenting the information to an external component. KORE is the system closer to DALICA because it uses agents for managing the information through the study of the User Profile. KORE does not use the Galileo signal and its agents are not logical. Moreover, DALICA is more centered on the deduction profile process while KORE mainly filters the information according to the User Profile characteristics. As future developments, the system reasoning capabilities that are presently quite basic can be improved. Also, previous experience can be better exploited. Different agents managing different visitors might communicate so as to cooperate in improving their performance and enhancing the services they offer.

V. ACKNOWLEDGEMENTS

This work has been partially supported by the project CUSPIS (GJU/05/2412/CTR/CUSPIS) "A Cultural Heritage Space Identification System".

REFERENCES

[1] F. Amigoni, S. D. Torre, and V. Schiaffonati, "Yet another version of minerva: The isola comacina virtual museum," in *Proc. of the First European Workshop on Intelligent Technologies for Cultural Heritage Exploitation, at The 17th European Conference on Artificial Intelligence*, 2006, pp. 1–5.

[2] A. Bhusate, L. Kamara, and J. Pitt, "Enhancing the quality of experience in cultural heritage settings," in *Proc. of the First European Workshop on Intelligent Technologies for Cultural Heritage Exploitation, at The 17th European Conference on Artificial Intelligence*, 2006, pp. 1–13.

[3] M. Bombara, D. Cal, and C. Santoro, "Kore: A multi-agent system to assist museum visitors," in *Proc. of the Workshop on Objects and Agents (WOA2003), http://citeseer.ist.psu.edu/708002.html*, 2003.

[4] S. Costantini and A. Tocchio, "A logic programming language for multi-agent systems," in *Logics in Artificial Intelligence, Proc. of the 8th Europ. Conf.,JELIA 2002*, ser. LNAI 2424. Springer-Verlag, Berlin, 2002.

[5] ——, "The dali logic programming agent-oriented language," in *Logics in Artificial Intelligence, Proc. of the 9th European Conference, Jelia 2004*, ser. LNAI 3229. Springer-Verlag, Berlin, 2004.

[6] R. Damiano, C. Galia, and V. Lombardo, "Virtual tours across different media in dramatour project," in *Proc. of the First European Workshop on Intelligent Technologies for Cultural Heritage Exploitation, at The 17th European Conference on Artificial Intelligence*, 2006.

[7] J. W. Lloyd, *Foundations of Logic Programming (Second, Extended Edition)*. Springer-Verlag, Berlin, 1987.

[8] D. Park, T. Nam, C. Shi, G. Golub, and C. V. Loan, *Designing an immersive tour experience system for cultural tour sites*, in chi '06 extended abstracts on human factors in computing systems ed. Montral, Qubec, Canada, April 22 - 27: ACM Press, New York, NY, 1193-1198, 2006.

[9] G. Pilato, A. Augello, A. Santangelo, A. Gentile, and S. Gaglio, "An intelligent multimodal site-guide for the parco archeologico della valle dei templi in agrigento," in *Proc. of First European Workshop on Intelligent Technologies for Cultural Heritage Exploitation, at The 17th European Conference on Artificial Intelligence*, 2006.

[10] A. Poon, "The new tourism revolution," *Tourism Management,vol.15, no.2*, 1994.

[11] U. site, "Villa adriana," http://www.villa-adriana.net.

[12] A. Tocchio, "Multi-agent sistems in computational logic," Ph.D. Thesis, Dipartimento di Informatica, Universitá degli Studi di L'Aquila, 2005.

An Approach to Multimodal Input Interpretation in Human-Computer Interaction

Fernando Ferri, Patrizia Grifoni, Stefano Paolozzi
Institute of Research on Population and Social Policies, National Research Council, Italy
{fernando.ferri, patrizia.grifoni, stefano.paolozzi}@irpps.cnr.it

Abstract

Human-to-human conversation remains such a significant part of our working activities because of its naturalness. Multimodal interaction systems combine visual information with voice, gestures and other modalities to provide flexible and powerful dialogue approaches. The use of integrated multiple input modes enables users to benefit from the natural approach used in human communication. However natural interaction approaches introduce interpretation problems. In this paper is presented an approach to interpret user's multimodal input. Starting from the analysis of the different types of modalities' cooperation we take into account the user's input behavior in order to better approximate the resultant multimodal input sentence with the user's intention. This multimodal sentence is transformed in a natural language one and we provides an algorithm to calculate the exact/approximate interpretation according to the sentence similarity level with sentence templates stored in a predefined knowledge base.

1. Introduction

People can communicate with great efficiency and expressiveness adopting natural interaction approaches. Perhaps this is the main reason why face-to-face conversation remains such a significant part of our working activities despite of the availability of a great number of communication technologies. Nevertheless, there is a great interest in research about natural interaction approaches in order to develop technologies to facilitate them [2], [6], [14] and to improve the interpretation by the computer side.

From this perspective multimodality has the potential to greatly improve Human Computer Interaction combining harmoniously the different communication methods . A user can use voice, handwriting, sketching and gesture to input information., The system can use icons, text, sound and voice (output)to present information.

This paper uses the concept of multimodal language defined as a set of multimodal sentences [4], by the extension of the definition of Visual Language given in [3]. A multimodal sentence contains atomic elements (glyphs/graphemes, phonemes and so on) that form the Characteristic Structure (CS). The CS is given by the elements that form functional or perceptual units for the user. A multimodal sentence is defined, similarly to [5], as a function of: 1) the multimodal message, 2) the multimodal description that assigns the meaning to the sentence, and 3) the interpretation function that maps the message with the description, and the materialization function that maps the description with the message.

One of the most relevant problems of the multimodal dialog consists of the fact that naturalness of communication is directly proportional with the complexity of the interpretation of messages.

The goal of this paper is to propose a new approach to understand how different modalities cooperate each other, taking into account the user's behavior that can alter the multimodal input recognized by the system. The resulting multimodal input sentence is matched with a template stored in a knowledge base to provide an interpretation of the sentence. The sentence can precisely match the template or approximates it.

We consider the speech modality as the prevalent one because, generally, users explain their intentions by speech and use other modalities to "support" the speech and eventually to resolve ambiguities.

For this reason we have chosen the system can map concepts involved in the multimodal sentence using a natural language sentence. Each multimodal sentence corresponds to a natural language one. The system returns the interpretation of the multimodal sentence if the template that exactly maps with the corresponding natural language sentence is available. When the user interacts with the system, the corresponding multimodal sentence must refer to a stored template in the knowledge base in order to be interpreted. If the corresponding sentence, expressed in natural language, doesn't match with any of the stored templates than the system can interpret those sentences that approximate the matching considering templates similar to the first one. Because of some multimodal sentences (with different templates) can have very close interpretations (some times they can have the same meaning), this paper proposes to calculate the templates' similarity starting from the semantic similarity of the natural language sentences corresponding to the Multimodal one.

For this purpose it is possible the association of a sentence and its template with the most similar template computing semantic similarity between natural language sentences. In this way we can provide an interpretation of the multimodal sentence also in case of non-perfect matching.

The natural language sentence can be represented as a semantic network of objects and binary relations among them, where each object corresponds to one node.

The rest of the paper is structured as follow: section 2 briefly presents a short literature of related works; section 3 provides a running example, in section 4 is illustrated our approach to interpret the multimodal input basing on modalities interaction's type and user's behavior, section 5 is devoted to the evaluation of sentence similarity in the multimodal context, section 6 concludes and describes some future research.

2. Related Work

The problem of multimodal system interaction has been studied for several years and still remains an active research field.

Several studies have examined multimodal interaction and interpretation addressing in particular sketch-speech interaction. For example [8] discusses an integration technique in order to solve the problem of interpretation of multimodal queries constituted by multiple speech and sketch inputs. In [1] is underlined the importance of the study of the user's behavior in order to correctly interpret multimodal inputs combining sketching with speech, enabling a more natural form of communication. In [11] and [4] are proposed interesting approaches for studying multimodal interaction showing how the use of integrated multiple input modes enables users to benefit from the natural approach used in human communication.

3. A Motivating And Running Example

In order to explain our approach we consider the following example: the user draws an Entity-Relationship scheme assigning a label to each construct. Without loss of generality we assume that the input is given by only two modalities: speech and sketch. This scenario is represented in Fig. 1.

Suppose that the user sketches the diagram as shown in Fig. 1a, and he/she speeches the sentences shown in Fig. 1b. The system has to interpret the multimodal sentence and then has to materialize it as shown Fig. 1c.

For the sake of simplicity we only consider the creation of the Professor entity. The user says: "*The first rectangle is the entity Professor*", at the same time the user sketches the figure of a rectangle (in his/her intention).

If the figure is properly interpreted by the sketch recognizer as a rectangle (Fig. 2a), we are in the typical situation of redundant information given by different modalities (i.e. the concept "rectangle" is expressed both by speech and sketch modalities) The Natural Language sentence associated to the multimodal one will be: "*The first rectangle is the entity Professor*" and it represents all the concepts expressed by speech and sketch.

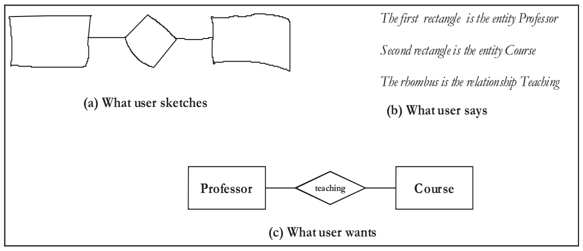

The first rectangle is the entity Professor
Second rectangle is the entity Course
The rhombus is the relationship Teaching
(a) What user sketches
(b) What user says
Professor
teaching
Course
(c) What user wants

Fig. 1. An example of multimodal interaction.

Given such a sentence, a corresponding template to the sentence in the knowledge base, must be found. The associated template (in lexical form) is:

DT + JJ + **rectangle** + VB + DT + **entity** + NN

where DT refers to a determiner, VB a verb, JJ an adjective, NN a singular noun.

The situation is more complex if we have some ambiguities problem. That is for example, if a user draws a figure that in his/her intention is a rectangle, recognized by the system as a square. So the system is unable to identify that the given input is redundant and can produce an incorrect multimodal sentence interpretation. In Fig. 2 both situation are illustrated with the proper timeline.

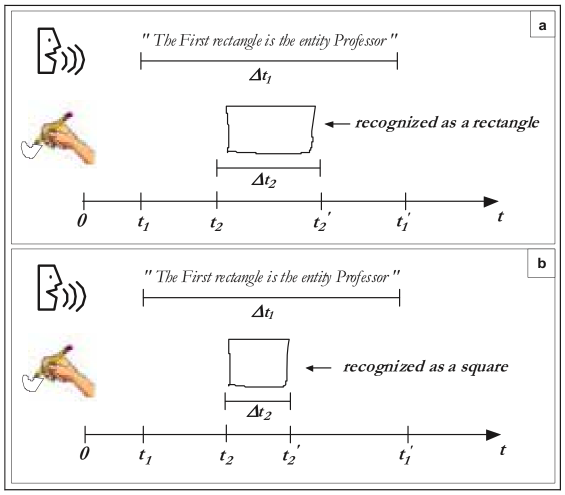

a
" The First rectangle is the entity Professor "
Δt1
recognized as a rectangle
Δt2
0 t1 t2 t2' t1' t
b
" The First rectangle is the entity Professor "
Δt1
recognized as a square
Δt2
0 t1 t2 t2' t1' t

Fig. 2. Example of multimodal input (by speech and sketch) with timeline.

From the user point of view both situation must reproduce the same multimodal sentence, but due to ambiguities in sketch recognition the system doesn't produce the same sentences and they are not associated to the same template. In order to avoid these problems we propose an approach that take into account the user's behavior to reproduce multimodal sentences as near as possible to the user's input will.

4. Our Approach

4.1. Multimodal interaction

The most important problem to solve is to understand how different modalities cooperate and what is the template that the multimodal sentence matches according to the cooperation modality.

Six type of cooperation between modalities have been distinguished (for a more formal definition see [11]):

- *Complementarity*: different chunks of information composing the same command are transmitted over more than one modality.
- *Concurrency:* independent chunks of information are transmitted using different modalities overlapping in time.
- *Equivalence*: a chunk of information may be transmitted using more than one modality.
- *Redundancy*: the same chunk of information is transmitted using more than one modality.
- *Specialization*: a specific chunk of information is always transmitted using the same modality.
- *Transfer*: a chunk of information produced by one modality is analyzed by another modality.

Let us consider the interaction of two modalities (in particular we address speech and sketch modalities) M_1 and M_2 that transmit information in ΔT_1 and ΔT_2 time intervals respectively. The possible time intervals relationship between M_1 and M_2 are summarized in Fig. 3 and are:

- *Sequential*: The transmission of the second modality starts after the first one.
- *Disjoint*: The transmissions of the two modalities take place in two separated time intervals.
- *Overlap*: The transmission of a modality partially overlaps the transmission of the other one.
- *Contains*: The transmission of a modality is self contained in the transmission of the other one.

Evaluating time intervals for the involved transmission, we analyze the possible combination of different input events. Multimodal input events can either be interpreted independently, or they can be merged.

Let us refer to the example given in the previous section. Firstly the system individually recognizes the concepts or the chunk of information for each modality involved in the input event. In this case the system must perform a speech and a sketch recognition in order to capture the initial information. Then each input modality is associated with its own time interval of transmission. The next step is to interpret these unimodal input on the base of the transmission time and the information recognized in order to extract a multimodal sentence representing the whole input event. The system's flow for this example is illustrated in Fig. 4.

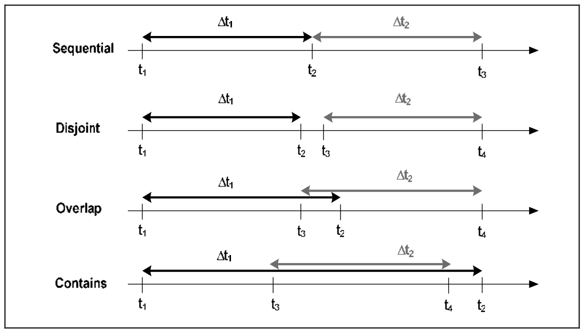


Fig. 3. Time intervals relationship in multimodal interaction.

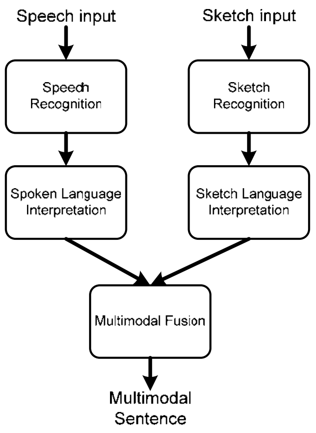


Fig. 4. The system's architectural flow.

For our purposes, it is important to address two type of cooperation: Complementarity and Redundancy. Considering the aforementioned time intervals, Sequential, Overlap and Disjoint transmissions can denote Complementarity interaction as Contains and Overlap transmissions can denote Redundancy interaction. In Fig. 5 this relationships are presented.

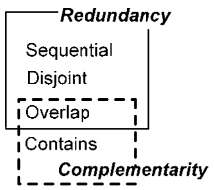


Fig. 5. Time intervals and type of interaction.

In this paper we are interested in Redundancy interaction, because it has been observed that a redundant multimodal input involving speech and sketch enables a more natural form of communication.

As stated before each modality transmission represents a finite number n of concepts.

Let us suppose that modality M_1 transmits the concept C_1 in the time ΔT_1 and the modality M_2 transmits the concept C_2 in the time ΔT_2. ΔT_1 partially overlaps time ΔT_2. In our example M_1 is the speech modality and M_2 is the sketch modality. We have to identify the template for the multimodal sentence in order to correctly interpret it. That is, this "match" requires interaction between modalities has to be recognized. A redundant interaction C_1 and C_2 must be the same concept. However, the modality used to express one concept (for example concept C_1) can introduce some imprecision and approximations. It can be happen in a situation in which the user's will is to draw a rectangle, but he/she draws a square instead. At the same time the concept expressed by speech mode is "the rectangle". In this case the two concepts C_1 and C_2 are *similar* concepts. The question is: what similarity measure has to be considered between two concepts in order to have a redundant interaction between modalities?

This measure car vary among different users. The system has to acquire knowledge on the user's behavior during the whole interaction process.

In particular we have considered a sample of 20 different users (10 men and 10 women) and we asked them to draw a rectangle, alternatively with other figures, with our sketch interface, for 20 times. This permits to take into account the user's behavior, that is used to better calculate the similarity between concepts.

The sketch recognizer will identify, for each sample, a measure of similarity between user's sketch and the required figure (in this case a rectangle).

This measure represents a fundamental element for the computation of concept similarity. More formally, if C_i is the i-th sample drawn by the user, and n is the total number of the samples, the user behaviour approximation A_{ss} (for sketch-speech modalities interaction) is:

$$A_{ss} = \frac{1}{n}\sum_{i=1}^{n} C_i \ , \qquad A_{ss} \in (0,1) \tag{1}$$

The concepts are stored in the concepts database as natural language terms. We adopt an extended words similarity algorithm in order to calculate semantic similarity between concepts are using the WordNet lexical database [15].

4.2. Evaluating Concepts Similarity

The taxonomical structure of the WordNet knowledge base is important in determining the semantic distance between words. In WordNet, terms are organized into synonym sets (synsets), with semantics and relation pointers to other synsets.

One direct method for similarity computation is to find the minimum length of path connecting the two words [12].

For example, the shortest path between student and schoolmate in Fig. 6 is student-enrollee-person-acquaintance, the minimum path length is 4, the synset of person is called the subsumer for words of student and schoolmate, while the minimum path length between student and professor is 7. Thus, we could say that schoolmate is more similar to student than professor to student.

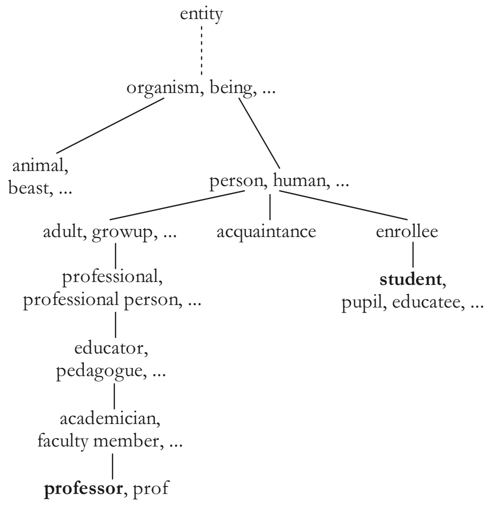


Fig. 6. A portion of the Wordnet's semantic network.

However, this method may be not sufficiently accurate if it is applied to a large and general semantic net such as WordNet. For example, the minimum length from student to animal is 4 (see Fig. 6), less than from student to professor; however, intuitively, student is more similar to professor than to animal. To address this weakness, the direct path length method must be modified as proposed in [10] utilizing more information from the hierarchical semantic nets. It is important to notice that concepts at upper layers of the WordNet's hierarchy have more general semantics and less similarity between them, while words that appear at lower layers have more concrete semantics and have a higher similarity. Therefore, also the depth of word in the hierarchy should be considered. In summary, we note that similarity between words is determined not only by path lengths but also by depth (level in the hierarchy). Moreover we take into account the user behaviour considering the user behaviour approximation A_{ss}. The proposed algorithm is an extensions of the one proposed in [10] for words similarity.

Given two concepts, c_1 and c_2, we need to find the semantic similarity *$s(c_1,c_2)$*.

Let be *l* the shortest path length between c_1 and c_2, and *h* the depth of subsumer in the hierarchical semantic nets, the semantic similarity can be written as:

$$s(w_1, w_2) = f_1(l) \cdot f_2(h) \cdot B_{ss} \tag{2}$$

B_{ss} represents the user's behaviour contribute and its value depends on the value of the product between *$f_1(l)$* and *$f_2(h)$*. If this value is higher than a threshold value (δ), B_{ss} is equal to *$(A_{ss})^{-1}$*, otherwise B_{ss} is equal to 0. More formally:

$$B_{ss} = \begin{cases} 0 & \text{if} \quad f_1(l) \cdot f_2(h) < \delta \\ \dfrac{1}{A_{ss}} & \text{if} \quad f_1(l) \cdot f_2(h) \geq \delta \end{cases} \tag{3}$$

The threshold value used in our experiments is 0.4.

The path length between two concepts, c_1 and c_2, can be computed according to one of the following cases:

1. c_1 and c_2 belong to the same synset,
2. c_1 and c_2 do not belong to the same synset, but their synsets contains one or more common words,
3. c_1 and c_2 neither belong to the same synset nor their synsets contain any common word.

First case implies that c_1 and c_2 have the same meaning, then we assign the value 0 to the semantic path length between c_1 and c_2. In the second case we can notice that c_1 and c_2 partially share the same features, then we assign the semantic path length between c_1 and c_2 to 1. For case 3, we must count the actual path length between c_1 and c_2. By the above considerations we can considered the function to be a monotonically decreasing function of l:

$$f_1(l) = e^{-\alpha l} \tag{4}$$

where $\alpha \in [0,1]$.

In order to calculate the depth contribution, according to the fact that at upper layers of the WordNet's hierarchy words have more general semantics and less similarity between them, while words that appear at lower layers have more concrete semantics and an higher similarity, function *$f_2(h)$* should be a monotonically increasing function with respect to depth h:

$$f_2(h) = \frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}} \tag{5}$$

where $\beta \in (0,1]$.

The whole formula for words similarity computation with the contribution of (4) and (5) is:

$$s(w_1, w_2) = e^{-\alpha l} \cdot \frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}} \cdot B_{ss} \tag{6}$$

The values of α and β depend on the used knowledge base. For WordNet the optimal parameters are α=0.2 and β=0.45 as explained in [9].

5. Evaluating Multimodal Sentence Similarity

To compare the user's input multimodal sentence with the stored templates the system has to proceed as follow:

let be T_1 the template associated with the user's input sentence T_1^S and T_2 the template associated to the stored sentence T_2^S.

According to [12] we must construct a vector V that contains all the distinct elements from the associated sentences T_1^S and T_2^S.

The semantic similarity between two sentences is defined as:

$$S_s = \frac{s_1 \cdot s_2}{\|s_1\| \cdot \|s_2\|} \tag{7}$$

where s_i (i=1,2) are the semantic vectors. The value of an element of a semantic vector is defined as follows:

$$\sigma_j = \hat{s}_j \cdot I(w_j) \cdot I(\hat{w}_j) \tag{8}$$

Where w_j is a word in the joint word set and $\hat{w}_j$ is its associated word in the sentence and $\hat{s}_j$ is the j-element of the lexical vector $\hat{s}$ (below introduced).

Function $I \in [0,1]$ is based on the notion of information content introduced by Resnik. The information content of a concept c quantifies as informativeness decreases when probability of the concept c increases [13]. Note that this approach has been chosen because various results in the literature demonstrate that it has a higher correlation with human judgment than the traditional edge counting approach [7].

It has been shown that words that occur with a higher frequency (in a corpus) contain less information than those that occur with lower frequencies:

$$I(w) = 1 - \frac{\log(n+1)}{\log(N+1)} \tag{9}$$

where N is the total number of words in the sentence, n is the frequency of the word w in the sentence (increased by 1 to avoid presenting an undefined value to the logarithm).

The lexical vector $\hat{s}$ is determined by the semantic similarity of the corresponding word in the joint word to a word in the sentence. Each entry of the semantic vector corresponds to a word in the joint word set, so the dimension is equal to the number of words in V. Take T_1^S as an example:

- if w_j appears in the sentence, $\hat{s}_j$ is set to 1.
- if w_j is not contained in T_1^S, a semantic similarity score is computed between w_j and each word in the sentence T_1^S, using the method similar to the one illustrated for concepts similarity in the previous section..

Thus, the most similar word in T_1^S to w_j is that with the highest similarity score.

6. Conclusion

Multimodal human-computer interaction, in which the computer accepts input from multiple channels or modalities, is more flexible, natural, and powerful than unimodal interaction with input from a single modality. However naturalness of communication is directly proportional with the complexity of the interpretation of messages. In this paper is presented an approach to interpret multimodal input sentences, we have implemented a multimodal system based on speech and sketch modalities that use the aforementioned methodology to better understand user's input, also considering the user's way of inputting. Future work will address other input modalities, such as gesture, that could also help disambiguate the sketches and correctly simulate the user's ideas.

References

[1] Adler, A., Davis, R.: "Speech and sketching for multimodal design", in *Proc. of the 9th Int. Conf. on Intelligent User Interfaces*, ACM Press, 2004, pp. 214–216.

[2] Binot, J. L., Falzon, P., Sedlock, D., Wilson, M. D.: "Architecture of a multimodal dialogue interface for knowledge-based systems", in the *Proc.of Esprit'90 Conference*, Kluwer Academic Publishers: Dordrecht, 1990, pp. 412-433.

[3] Bottoni, P., Costabile, M. F., Levialdi, S., Mussio, P.: "Formalizing visual languages", in *Proc. of IEEE Symp. Vis. Lang. '95*, IEEE CS Press, 1995, pp. 334-341.

[4] Caschera, M. C., Ferri, F., Grifoni, P.: "Multimodal interaction systems: information and time features", in *Int. Journal of Web and Grid Services 2007* - Vol. 3, No.1 pp. 82 - 99, to be published.

[5] Celentano, A., Fogli, D., Mussio, P., Pittarello, F.: "Model-based Specification of Virtual Interaction Environments" in *Proc. of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing* (VLHCC'04) - Volume 00, 2004, pp. 257-260.

[6] Johnston, M., Cohen, P. R., McGee, D., Oviatt, S. L., Pittman, J. A, Smith, I.: "Unification-based Multimodal Integration", in *Proc. of the 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, Madrid, 1997, pp. 281-288.

[7] Lee, J. H., Kim, M. H., Lee, Y. J.: "Information retrieval based on conceptual distance in is-a hierarchies", in *Journ..of Documentation*, Vol.49, n.2, 1989, pp. 188-207.

[8] Lee, B-W., Yeo, A.W.: "Integrating sketch and speech inputs using spatial information", in *Proc. of the 7th Int. Conf. on Multimodal interfaces*, ACM Press, 2005, pp.2-9.

[9] Li, Y., McLean, D., Bandar, Z. A.: "An Approach for Measuring Semantic Similarity Using Multiple Information Sources", *IEEE Trans. Knowledge and Data Eng*., vol. 15, no. 4, 2003, pp. 871-882.

[10] Li, Y., McLean, D., Bandar, Z. A., O'Shea, J. D., Crockett, K.: "Sentence Similarity Based on Semantic Nets and Corpus Statistics" in *IEEE Trans. Knowledge and Data Eng*, vol. 18, n. 8, 2006, pp. 1138-1150.

[11] Martin, J.C.: "Toward intelligent cooperation between modalities: the example of a system enabling multimodal interaction with a map", in *Proc. of Int. Conf. on Artificial Intelligence (IJCAI'97) Workshop on "Intelligent Multimodal Systems"*, Nagoya, Japan, 1997.

[12] Rada, R., Mili, H., Bichnell, E., Blettner, M.: "Development and Application of a Metric on Semantic Nets", *IEEE Trans. System, Man, and Cybernetics*, vol. 9, no. 1, 1989, pp. 17-30.

[13] Resnik, P. "Using information content to evaluate semantic similarity in a taxonomy", in *International Joint Conference for Artificial Intelligence*, 1995, pp. 448-453.

[14] Wahlster, W.: "User and discourse models for multimodal communication", in *Readings in intelligent user interfaces*, Morgan Kaufmann Publishers Inc, San Francisco, 1998, pp. 359-371.

[15] WordNet 2.1: A lexical database for the English language. http://www.cogsci.princeton.edu/cgi-bin/webwn, 2005.

Sketch Style Recognition in Human Computer Interaction

Danilo Avola, Fernando Ferri, Patrizia Grifoni
Institute of Research on Population and Social Policies, National Research Council,
Via Nizza 128, 00198 Rome, Italy
{danilo.avola, fernando.ferri, patrizia.grifoni}@irpps.cnr.it

Abstract

Sketch-based interaction enables users' simple communication and it is used to represent concepts and commands in human-computer interaction. This communication approach can be used in different contexts with different devices. The ink style through which the user performs a sketch is a critical component in the recognition and interpretation processes. In particular, the different users' styles adopted to perform the sketch can introduce over-tracing and/or cross-hatching phenomena that are respectively represented in the sketch like bold style or dashed style. The paper provides an approach to recognize the different ink styles performed by a user during his/her sketch activity. More specifically an approach to recognize both a single stroke style and the whole sketch style is presented.

1. Introduction

People use the free hand drawing to express concepts and to represent ideas in an immediate and intuitive simple way. Therefore, the use of sketch-based interfaces to interact with different devices provides a natural convenient way to carry out concepts and commands.

The user interaction is influenced by the individual drawing style. Sometimes, tools used to interact with the different devices (such as: smart phone, PDA, palmtop, tablet-PC, and so on) and/or specific contexts can influence the individual ink style too. As shown in Figure 1, different users can draw the same object/concept using different styles. This can happen without specific reasons because a sketch action is intrinsically an informal and messy human expression. Therefore, the over-tracing and/or dashed-tracing phenomena can frequently occur. In this paper discussion is focused on recognition of sketch drawing styles. Figure 1 shows the same object/concept (circle) drawn by four different sketches obtained using/combining the three drawing styles (solid, bold, dashed described in section 2). The different drawing styles make any interpretation of the sketch very hard.

The purpose of this paper is to propose and discuss an approach to recognize some of the different styles performed by the user during her/his sketch activity. More specifically, the approach recognizes both a single stroke style and the whole sketch style.

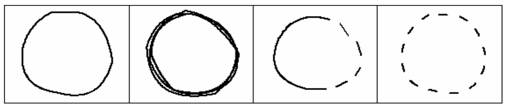


Fig. 1. Four different circle representations.

There is an extensive body of related work on sketch style recognition in Human Computer Interaction. In [1] [2] [3] authors suggest a fast, simple and compact approach to recognize sketches drawn with a stylus on a digitizing tablet. The approach enables the system to identify shapes (such as: triangles, lines, rectangles, circles, arrows, crossing lines, diamonds and ellipses) of different size and with different orientation in the space, drawn with continuous, dashed-tracing or over-tracing strokes. This approach is based on the computation and ratio of particular geometrical features of shapes in the sketch (such as: convex hull, the largest triangle and the largest quadrilateral inscribed in the convex hull, the smallest enclosing rectangle of the convex hull). The over-tracing action, considered as a user imprecision, is discussed in [7]. The system generates geometric approximations for single stroke shapes that are over-traced. According to this approach the system has to match input strokes with geometric primitives producing fits for several shapes (such as: lines, arcs and circles) by computing model's parameters that minimize the least squares fitting error. Also in [9] the problems that involve the interpretation of over-tracing freehand sketch are faced. In this approach over-tracing strokes are interpreted according to the following steps: 1) the strokes are first classified into lines and curves by a linearity test, 2) after this a strokes grouping process that handles lines and curves separately is performed. Afterwards, the grouped strokes are fitted with 2D geometry and further tidied-up with endpoint clustering and parallelism correction. Finally, the in-context interpretation is applied to detect incorrect stroke interpretation based on geometry constraints and to suggest the most probable correction based on the overall sketch context. In [5] the authors present an approach that, unlike the previous two, deals with both the over-tracing and hatching sketch on the 3D sketch. With this approach the strokes are divided into core strokes (strokes that touch the characteristic curves of the object), and hatching strokes (strokes that are mapped to the faces of the object). This approach allows the interpretation of the user's sketch that presents over-tracing strokes, by grouping strokes into bundles. Another approach to recognize the free hand sketches drawn using different styles (with over-traced, dashed or continuous strokes) is given in [8]. The key advance of this approach, is an integrated sketch parsing and recognition model designed to enable natural and pen-based computer interaction. With this approach, the stream of pen strokes is firstly examined to identify delimiter patterns called "markers". These then anchor a spatial analysis, which groups the remaining strokes into distinct clusters, each representing a single visual object. Finally, a shape recognizer is used to find the best interpretations of the clusters.

Indeed there are several approaches that face the over-tracing or (less frequently) cross-hatching phenomena. But there are not many works that face the problems from the stroke style point of view. This paper proposes an original approach to recognize the different drawing styles. It aims to solve the ink style problems while the user is performing the stroke (on-line recognition). The approach does not need to know the features of the geometric shapes in the sketch. It works only on each single stroke. For this reason the approach can be adapted for sketches made up of one/multi strokes. Besides, the approach allows the recognition of all styles that make up a single object/shape (made up of more than one stroke) in the sketch.

The paper is organized as follows. Section 2 describes the common ink styles with which usually a user carries out a sketch. Section 3 describes the approach to recognize the different ink styles. Finally, Section 4 concludes the paper.

2. Sketch Styles in the Drawing Activity

In this section the most common styles with which a user carries out a sketch are discussed. Sketch-based interaction is an intuitive and simple communication method used to represent concepts and communicate in different contexts. Sketches represent abstract ideas, concepts or commands using symbolic graphical elements and spatial constraints among them. A user conveys a concept or an idea by graphical ink individual style, and he/she does not follow precise rules. Besides this, the complexity of the sketch layout may depend on the specific context. In fact a sketch performed in a specific context (such as: Data Flow Diagram or Entity Relationship Diagram, and so on) usually is "less complex" than one performed in another specific context such as complex architectural designs. This more complex sketch context can require to use more ink styles than the simplest one. However, proportionally to the complexity of the sketch in the different contexts, there is the necessity to interpret correctly the ink style performed by the user. Usually a user performs a sketch by three defined styles [4]: solid, bold, dashed. The first style (solid), as shown in Figure 2-a (simple electrical scheme), is commonly used to represent a considerable part of a sketch (or the whole sketch). The second style (bold), as shown in Figure 2-b (simple example of a binary tree), is generally used to represent an object in a particular contexts, or to emphasize a particular area or object (or its part) in a sketch. Otherwise it can be a personal user expression style.

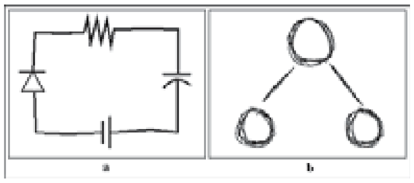


Fig. 2. An example of (a) solid style, (b) bold style.

The third style (dashed), as shown in Figure 3, can have different uses. For example, a user could want to "hide" some parts of a sketch that are useful to understand the context of the sketch, but not important to understand the core meaning of it, otherwise a user could want to distinguish different objects or to highlight different groups of objects by using different dashed ink patterns. Figure 3-a shows an example of elements extracted from a sub-database contained in a main database, while Figure 3-b shows several 2D geometric figures. In Figure 3-a the dashed style is used to focus on the context in which the elements extraction from the sub-database occurred, while in the Figure 3-b different dashed pattern styles are used to distinguish the three groups of Figures (elliptical group; quadrilateral group; triangular group).

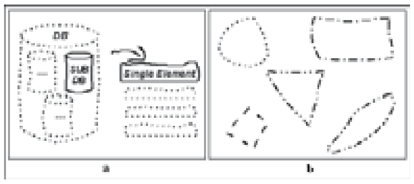


Fig. 3. An example of (a) sketch with dashed style, (b) different patterns of dashed style.

In this context, a stroke is a drawing action defined by the sequence: pen down, pen movement, pen up.

The next section introduces the proposed approach to recognize the different ink styles conveyed by the user during sketch drawing.

3. An Approach to Recognize Sketch Styles

This section discusses the approach proposed to detect the different ink styles in the sketch. In the previous section it has been observed that usually there are three styles to draw a sketch (solid, bold, dashed). Indeed every stroke of a sketch can be drawn only in solid or bold style. In fact, usually, a dashed style is obtained composing solid and/or bold strokes. Figure 4 shows that a dashed style is a repetition of one pattern. In Figure 4-a there are four examples of rectangles drawn with different dashed styles. Because of the given stroke definition each rectangle in Figure 4-a is drawn with more than one stroke (we assume that every stoke represented in Figure is in solid style). Looking at Figure 4-b, it is possible to observe that every rectangle in the Figure 4-a has been drawn through the repetition of a precise pattern. In particular the first and the third rectangle have been drawn using a pattern made up of a single solid stroke, while the second and the fourth rectangles have been respectively drawn through a pattern made up of two and three solid strokes.

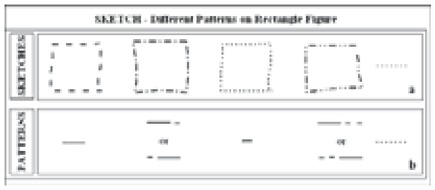


Fig. 4. Representations of the (a) same rectangle, and (b) related patterns.

The pattern used to draw the second and the fourth rectangle highlights the importance, from sketch-based context, of the pattern invariance to detect a dashed style. In fact, potentially, every single stroke can be different from the other; therefore every dashed shape (which is composed of pattern repetition) is drawn by a repetition of similar (but not identical) patterns. This aspect has to be taken into account during dashed style recognition (as shown in the second part of this section).

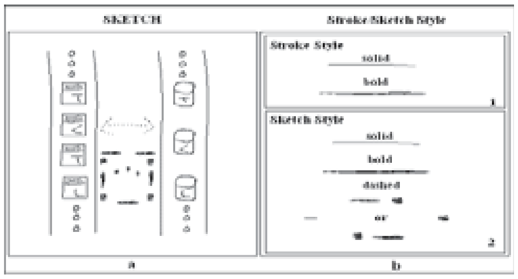


Fig. 5. Stroke style and sketch style.

The first part of this section concerns the approach used to detect the stroke styles (solid, bold), the second concerns the approach to detect the style of every ink used in the sketch (solid, bold, dashed). The last two concepts are shown in Figure 5. In Figure 5-a a sketch is represented. In Figure 5-b-1 there are the two different stroke styles. In Figure 5-b-2 all the recognized different styles in the sketch are represented. In particular in Figure 5-b-2 the three different dashed styles are represented. They respectively belong to (in the middle of the Figure 5-a): 1) the bi-directional arrow, 2) the rectangle and the row in the rectangle.

The first step in detecting stroke style consists of the analysis of the spatial and temporal sequence with which the pixels belonging to the stroke are drawn by the user during sketch activity. The aim of analysis is to detect the possible direction change conveyed by the user during stroke drawing. As shown in Figure 6, this analysis can be easily performed by studying the behavior of a single space component, of the stroke, in function of the time.

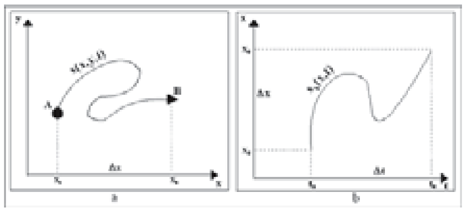


Fig. 6. (a) The stroke s, (b) the projection, on x axes, of the stroke s.

As shown in Figure 6-a, a stroke (s) is a function of both spatial coordinates (x, y) and temporal coordinate (t). The labels A and B show respectively the start point (at time $t=t_s$) and the end point (at the time $t=t_e$) of the drawn stroke.

Considering only one of the two spatial coordinates of each pixel (for instance x) and the time (t) in which the pixels have been drawn, it is possible to obtain a reliable bi-dimensional transposition of the stroke drawn by the user. In this way, as shown in Figure 6-b, a function (s_x) depending on both the chosen axis (x) and the time (t) can be considered. This function represents the projection on x axes of the stroke (s). The Δx shows the variability range of the x values during the drawing of the stroke, the x coordinates (x_s and x_e) show respectively the x spatial start point (at time time $t=t_s$) and the x spatial end point (at the time $t=t_e$) of the drawn stroke. Finally Δt shows the drawing stroke time interval. The projection function (s_x) allows highlighting the direction change performed by the user during stoke drawing (s). In fact, as shown in Figure 7, these changes (on stroke s) are conveyed in proximity to the local and/or absolute maximum and minimum points belonging to the projection function (s_x).

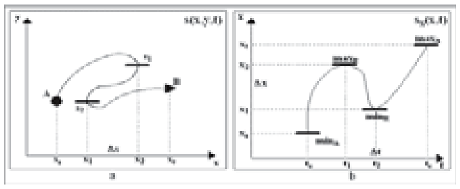


Fig. 7. (a) Stroke with two direction changes, (b) local and/or absolute maximum and minimum points.

It is important to observe that it is the chosen axis (in this case the x axis) that determines the meaning of direction change. In fact, in this context, the direction change of the stroke (s), is defined as the transition of x values from non-increasing monotone to non-decreasing monotone (and vice versa). In Figure 7-b the projection function (s_x) clearly shows four maximum and minimum points, and that is an absolute maximum point max_A (coordinates (x_e, t_e)), an absolute minimum point min_A (coordinates (x_s, t_s)), a relative maximum point max_R (coordinates (x_2, t_1)), and a relative minimum point min_R (coordinates (x_1, t_2)). In particular the points max_R and min_R identify the direction changes (respect x axis) of the stroke (s), while the other two points represent the start (min_A) and the end (max_A) points of the stroke (s). These four points univocally fix the related coordinates of the stroke (s). In fact, as shown in Figure 7-b, the points min_A, max_R, min_R and max_A are respectively related to the points A, s_1, s_2 and B of the Figure 7-a.

In this way it is possible to recognize the direction changes on stroke (s). In Figure 7-a the stroke (s) shows two direction changes according to the points (s_1 and s_2). These changes allow for the detection, in the stroke (s), of three different sub-strokes. The first sub-stroke is shown by segment [A, s_1], the second one is shown by segment [s_1, s_2], and the last one is shown by segment [s_2, B]. The stroke (s) can thus be subdivided in several strokes, where every identified sub-stroke has a unique direction. It is important to observe that the identification of the single sub-stroke is independent from the speed with which the user has drawn it. The identification entirely depends on the behavior (transposed on the x axis) performed by the user during the stroke drawing. That is, if the user draws the stroke in Figure 7-a two times with a different speed, the approach will identify the same sub-strokes, as shown in Figure 8.

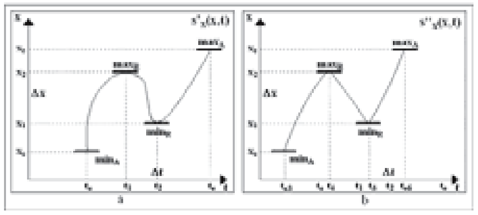


Fig. 8. Same stroke with two different temporal sequences.

The user has drawn the stroke in Figure 8-b faster than the stroke in Figure 8-a. Besides the user has kept a more constant speed during stroke drawing shown in Figure 8-b than the one shown in Figure 8-a. In spite of this, the approach has detected (spatially on x axis) the same coordinates for the four maximum and minimum points (even if the functions $s'_x(t)$ and $s''_x(t)$ are obviously different). Therefore the approach will consider the same sub-strokes shown in Figure 7-a.

In this context it is important to observe that during sketch activity a stroke can cross itself one or more time (for example when a user performs a bold stroke). In this way various pixels can be over-traced. In spite of this, every time that a pixel is over-traced the related time has to be considered (obtained always a continuous function).

The continuity of the generic projection function (s_x) is necessary to consider the possible relative maximum and minimum points drawn upon a pixel already drawn.

When all the sub-strokes have been detected the second step of the approach will be performed.

The second step to detect stroke style has to take into account every sub-stroke detected in the previous step. An enclosing rectangle for every sub-stroke can be considered, as shown in Figure 9.

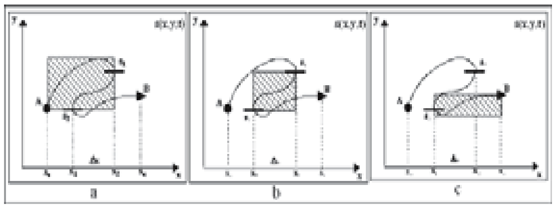


Fig. 9. Enclosing rectangles on: (a) first sub-stroke, (b) second sub-stroke, (c) third sub-stroke.

In Figure 9 the three sub-strokes in the relative enclosing rectangles are shown. In particular Figure 9-a shows the enclosing rectangle for the sub-stroke [A, s_1], Figure 9-b shows the enclosing rectangle for the sub-stroke [s_1, s_2], and Figure 9-c shows the enclosing rectangle for the sub-stroke [s_2, B]. The enclosing rectangle represents the smallest rectangle that contains the stroke segment.

The last step of the approach to discriminate the stroke style (bold or solid) is to analyze the relationship among all the enclosing rectangles. Usually a low level overlap among all the enclosing rectangles indicates that the stroke has been drawn with solid style, on the other hand a high level overlap indicates that the stroke has been drawn with bold style. A stroke drawn in a solid style usually has all its enclosing rectangles not completely overlapped, as shown in Figure 10-a and 10-b, while, a stroke drawn in a bold style usually has all its enclosing rectangles strongly overlapped, as shown in Figure 10-c and 10-d. As it is possible to observe, in Figure 10-b, there are two wide areas (A_1 and A_2) that are covered only by two different enclosing rectangles. Exactly, the area A_1 belongs only to the enclosing rectangle considered for the sub-stroke [A, s_1], and the area A_2 belongs only to the enclosing rectangle considered for the sub-stroke [s_2, B].

In Figure 10-d, the areas (A_1 and A_2) that are covered only by two different enclosing rectangles are smaller than in the previous example.

We have observed that in a solid stoke the area intercepted by the union of areas covered by A_1 and A_2 (in Figure 10-b, $A_1 \cup A_2$) is a larger part (among the 46% to 100%) than the area made up by union of all the areas of the rectangles in Figure 10-a. The bold stroke has, on the same involved areas, a smaller part (among the 0% to 37%) than the union of areas of Figure 10-c.

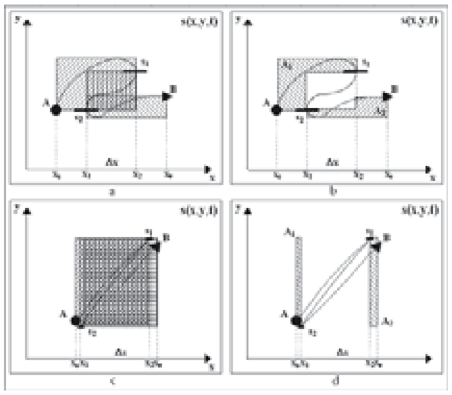


Fig. 10. Enclosing rectangle on: (a)(b) solid stroke, (c)(d) bold stroke.

Given the approach to detect the stroke style, it is possible to introduce the approach to detect the sketch ink style. Now, with the aim to detect dashed style in the sketch (if present), it is necessary to analyze the relationships among the recognized strokes. In order to explain better this second approach we introduce the example of Figure 11.

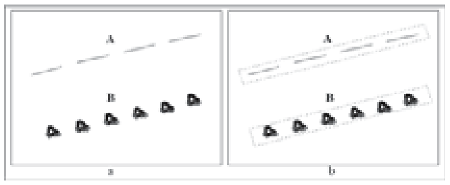


Fig. 11. (a) Ten strokes, (b) two dashed strokes.

The stroke style approach recognizes, in the Figure 11-a, ten strokes. More exactly, the approach recognizes two different groups of strokes. The first one (in Figure 11-a-A) made up of four solid strokes and the second one (in Figure 11-a-B) made up of six bold strokes. The best interpretation for the sketch in Figure 11-a, as shown in Figure 11-b, is the one that recognizes two dashed strokes. The first one made up of four strokes belongs to the first group, and the second one made up of six strokes belongs to the second group.

The approach to reach this new level of recognition (sketch inked style) takes into account the spatial and temporal relationships among strokes.

The first step of this approach, as shown in Figure 12-a, is to consider an enclosing rectangle for each stroke previously recognized. The further step, as shown in Figure 12-b, is to determine the barycentre of each enclosing rectangle. After that a neighbourhood relationship has to be identified among the barycentres' rectangles. The neighbourhood relationship is based on simple distance measure among barycentres' rectangles.

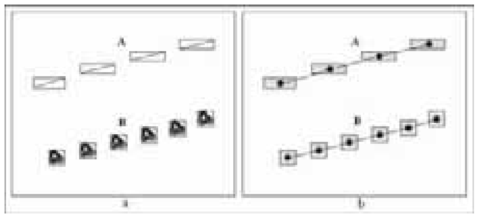


Fig. 12. (a) Enclosing rectangles, (b) enclosing barycentres' rectangle.

As shown in Figure 13-a the barycentres' rectangles are considered according to the order in which the related strokes have been drawn. Consequently, as shown in Figure 13-b, c and d, the distances among each ordered couple of barycentres' rectangles are considered. Since each calculated distance is smaller or equal than the fixed threshold (thr) the involved strokes can be considered belonging to a new stroke (dashed stroke) made up of the original strokes.

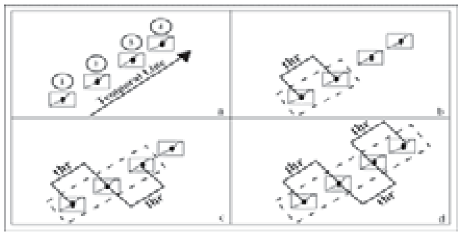


Fig. 13. A stroke made up of four sub-strokes.

As it is possible to observe in Figure 14, when two rectangles centers are too far, taking into account the considered fixed threshold, more dashed strokes have to be considered.

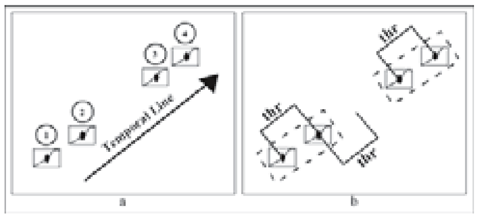


Fig. 14. (a) Four strokes, (b) two dashed strokes.

Figure 14-a shows four strokes drawn in a solid style. Figure 14-b shows two dashed strokes each one made up of two original solid strokes.

A dashed stroke can be made up of complex patterns. Through the approach introduced to detect the sketch ink style it is also possible to identify the type of pattern that composes a dashed stroke.

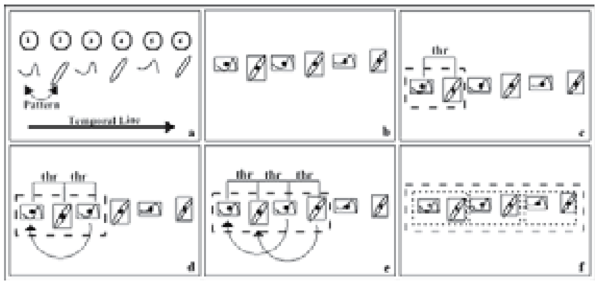


Fig. 15. Complex dashed pattern.

Figure 15-a gives a simple example of dashed stroke made up of a pattern that involves two solid strokes.

Figure 15-b shows the main step of the approach to detect the sketch ink style, that is the detection of the enclosing rectangles.

In Figure 15-c the first couple of rectangles' barycentres is considered, and a dashed stroke is recognized. In Figure 15-d a new solid stroke is considered, and a similarity relationship is detected between the first and the third solid strokes. In Figure 15-e another solid stroke is considered and another new similarity relationship is detected between the second and the fourth solid strokes. Now, there are two similarity relationships that cover all the four solid considered strokes, for this reason it is possible to split the set of solid strokes in two sub-sets, which have two solid strokes each. In this way, as shown in Figure 15-f, a dashed stroke can be recognized, besides the pattern that composes the dashed stroke can be identified.

Finally, as added value, taking into account the temporal information on the strokes that composes a pattern, it is also possible to identify the verse of a pattern.

There are several ways to detect a similarity relationship between strokes contained in enclosing rectangles. In this case the measures of entropy (adapted to binary image), average and variance of the simple distribution of the pixels in the enclosing rectangles, as reported in [6], have been sufficient to evaluate the similarity of two strokes.
The pattern recognition approach proposed in this paper to identify the dashed pattern works when all the strokes that make up the pattern are different according to their entropy measure. Indeed, this kind of reasoning can be expanded to allow us to recognize more complex patterns. The approach to follow in order to reach this result is to group strokes according to their similarity. This process has to take into account the verse of the dashed pattern, moreover a certain decision can be taken only when all the strokes reached by the threshold have been analyzed. The proposed approach discussed for the x axis can be extended to the other spatial coordinates.

4. Conclusion

The spread of sketch-based interfaces to support the different devices is mainly due to the fact that they make communication intuitive and spontaneous. In this way a user can easily represents concepts, commands and ideas in different contexts building a powerful interaction language. The ink style through which the user performs a sketch is a critical aspect. Due to this aspect, recognizing and interpreting steps are very hard tasks.

The selection of a specific ink style is driven by a personal ink style. Sometimes, it also depends on the interaction tools and/or the specific contexts. This paper proposes a strategy to detect the sketch ink style in human-computer interaction. The approach is subdivided in two main steps. In the first one it detects the solid or bold style of every stroke that composes the sketch. In the second it researches temporal, spatial and similarity relationships among the detected stokes to identify the dashed strokes. In this way every ink in a sketch is recognized as solid, bold or dashed stroke. The shown approach takes into account only the single stroke and it does not need to know the shapes features contained in the sketch. In this way it is possible to recognize also the different ink styles that can compose the different sides of the same object/shape.

References

[1] M.J. Fonseca, J.A. Jorge, "Experimental Evaluation of an on-line Scribble" Pattern Recognition Letters, Vol. 22 (12), pp. 1311–1319, 2001.

[2] M.J. Fonseca, J.A. Jorge, "Using Fuzzy Logic to Recognize Geometric Shapes Interactively" fuzz IEEE, the 9th IEEE International Conference on Fuzzy System (FUZZIEEE), San Antonio TX USA, Vol 1, pp. 291-296, 2000.

[3] M.J. Fonseca, J.A. Jorge, "A simple approach to recognize geometric shapes interactively. In Proceedings of the Third Int. Workshop on Graphics Recognition (GREC'99), Jaipur India September, 1999.

[4] M.J. Fonseca, J.A. Jorge, "Sketch-Based Retrieval in Large Sets of Drawings" Ph.D. Thesis Department of Information Systems and Computer Engineering, Technical University of Lisbon Superior Technical Institute July, 2004.

[5] J. Mitani, H. Suzuki, F. Rimura "3D sketch: sketch-based model reconstruction and rendering" Kluwer Academic Publishers, pp. 85–98, 2002.

[6] W.K. Pratt "Digital Image Processing. PIKS Inside, Third Edition Copyright © John Wiley & Sons, 2001.

[7] T.M Sezgin, R. Davis "Handling Overtraced Strokes in Hand-Drawn Sketches" In Proceedings of the AAAI Spring Symposium Series: Making Pen-Based Interaction Intelligent and Natural, Washington DC, pp. 21-24, October 2004.

[8] L.B. Kara "Automatic parsing and recognition of hand-drawn sketches or pen-based computer interfaces" Ph.D. Thesis Mechanical Engineering Department Carnegie Mellon University, September 2005.

[9] D.C Ku, S.F. Qin, D.K Wright "Interpretation of Overtracing Freehand Sketching for Geometric Shapes" The 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG ' 2006), January 30 - February 3 2006.

Human-Computer Interaction for a Novel Arm-wrestling Robot

Chul-goo Kang, and Ho-yeon Kim
Department of Mechanical Engineering
Konkuk University, Seoul, Korea
cgkang@konkuk.ac.kr

Abstract. *A peculiar arm-wrestling robot recently developed in our laboratory is basically comprised of a mechanical arm and a control computer. The arm-wrestling robot detects the maximum arm-force of the user in the early stage of the match, generates a different game scenario each time, and executes force feedback control to implement the scenario. This paper presents the configuration of the arm-wrestling robot, implementation of real-time control, design of the software structure, and human-computer interaction. The validity of the proposed system is verified through experimental studies, and is demonstrated at the Future Tech Korea Exhibition.*

I. INTRODUCTION

Presently, a novel arm-wrestling robot called Robo Armwrestler is under development for senior health care in our Intelligent Control and Robotics Laboratory (intro-bot.konkuk.ac.kr), which is supported by the Korean Government. Our vision is to realize humanoid robots that have entertaining functions such as arm wrestling and chess playing, as well as service functions such as errands, and help the elderly and the disabled in the near future.

Several years ago, Y. Bar-Cohen issued a challenge to build a robot using muscles of electrically activated polymers that could arm-wrestle a human [1]. As a result of challenge, a few interesting arm-wrestling robots were built using electroactive polymers (EAP) [2]. However, the primary object of these arm-wrestling robots is to demonstrate the potential of EAP technology, and thus these robots do not have a broad range of functions related to arm wrestling skills. Another effort relating to a humanoid robot arm has been in the field of prosthetic devices, such as the Utah Artificial Arm [3]. However, it does not appear that prosthetic devices are suitable for arm wrestling, in which strong arm-force is required.

Several practical arm wrestling devices have been patented as amusement units or as units for developing and strengthening wrist and arm muscles [4]-[7]. These arm wrestling devices may be classified roughly into three types according to the methods of generating reaction force against player's arm-force.

The first type of device uses a spring force; a typical example is U.S. Patent 3,947,025 [4], in which the arm wrestling device is comprised of a helical coiled spring that has adjustable stiffness. The second type of device uses pneumatic or hydraulic cylinders, which is better than the spring type from the viewpoint of force manipulability. However, a disadvantage is that the system becomes complicated and bulky because of supplementary device for pneumatic or hydraulic pressure generation. An example of the hydraulic device is U.S. Patent 5,842,958 [5]. The third type of arm wrestling device uses electric motors instead of springs or pneumatic/hydraulic cylinders in order to generate resistive force against the user. Examples of this type are Japan Patent 6-315544 [6] and Japan Patent 2002017891 [7], in which torque motors are used for generating arm-force, and sensor plates and photosensors are used for detecting arm speed in order to prevent a throw fracture of the player.

These patented devices are invented for playing simple arm wrestling games or practicing strength training, in which they usually generate fixed force levels. If the player generates a stronger force than the arm wrestling device, then he will win; otherwise he will lose the game. Therefore, the device has a deficiency in that the player is soon bored after a few trials.

The Robo Armwrestler has salient features to improve the above deficiency, namely, it generates automatically the force level appropriate to each person after sensing the human arm-force at the early stage of the match; its generated force profile varies with each match in order for the person to play for a long time without being bored; the winning average of the robot is determined randomly, but the person's will to win during the match influences the winning average of the robot. The robot recognizes a human's approach and human's sitting on the chair, and begins to talk and guide. The facial expression of the avatar changes synchronously according to arm wrestling situations.

This paper presents the configuration of the arm-wrestling robot, implementation of real-time force control, design of the software structure, and human-computer interaction.

II. Configuration of The System

The arm-wrestling robot, Robo Armwrestler, is basically composed of a mechanical arm and a control computer. The mechanical arm comprises a servo motor, a position/velocity sensor, a speed reducer, a torque sensor, three inclinometers, and an adapter with a mechanical stopper as shown in Fig. 1. Two ultrasonic sensors are attached at the right and left sides on the front of the table and detect a human's approach within a prescribed range of angles near the arm-wrestling robot. One photoelectric sensor using infrared rays detects a human sitting on a chair. In order to guide the player, the display monitor and speakers are prepared at an appropriate position of the table.

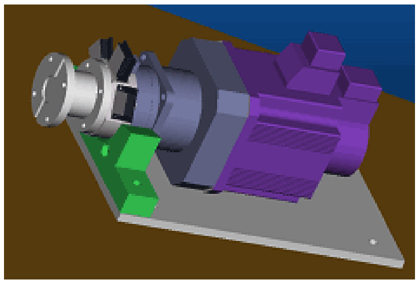


Fig. 1. Mechanism for arm-force generation.

The control system of the robot is comprised of a CPU, a memory part, an amplifier part, a logic circuit part, a pulse generation part, and output ports. The CPU produces motor control input using the control program and the feedback signals, and produces voice and image signals. Voice speakers and a display monitor are driven by the CPU through output ports.

Torque sensor, inclinometer, photoelectric sensor, and ultrasonic sensor signals are converted to digital signals through A/D converters, and transmitted to CPU. Encoder signals are transmitted to CPU directly through digital input channels. Motor driving signals are converted to analog voltage through a D/A converter, and transmitted to the motor driver.

When the CPU is down, the D/A converter can still output the last signal of the motor control input, and thus a dangerous situation can occur if the electric power is applied to the motor. To resolve this problem, the CPU transmits an initialization completion signal to the motor power control part through a D/A converter, and sends 0 value to the motor driving part through the D/A converter when the initialization procedure is completed. The motor power control part turns on the mechanical relay (MR) to supply the electric power to the motor according to the output signal of the sold state relay (SSR), which in turn is actuated by the initialization completion signal.

User safety is thus guaranteed even if the motor power switch is turned on before completing the initialization procedure or at abnormal conditions of the control system since the electric power is not transmitted to the motor.

III. Real-time Force Control

To make the arm-wrestling game attractive, we add force control functions to the system in such a way that force command is generated intelligently, and that real-time based tracking control can follow the given force command. Linux and RTAI (real-time kernel) have been adopted for the operating system of the robot. When Linux and RTAI were implemented at desktop PC with Pentium IV 2.8GHz, timing errors less than ±0.02 ms occurred for generating 5 ms timer interrupts. Window XP had roughly ±1 ms latency when we executed the same program with the same PC platform.

A block diagram of the force feedback control scheme is shown in Fig. 2. The control system receives feedback signals of actual position, velocity, and torque, and calculates torque command using feedback signals and scenarios. The system then controls the mechanical arm by means of generating the motor control input using force control logic. The force control logic is basically a PID type, but it uses velocity and position information together with force information.

Force control performance is mainly dependent on the accuracy of feedback signals, and real-time control capability, including the accuracy of sampling time, and the force feedback control logic itself. Force feedback control plays a key role in arm-wrestling the robot, but position feedback control is also necessary for rotating the mechanical arm to a starting position, and setting the initial absolute angle of the mechanical arm before a match.

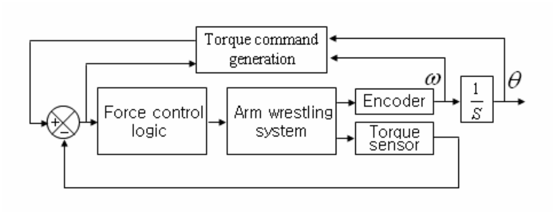


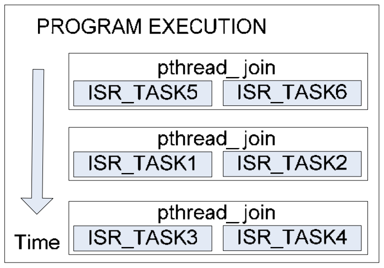
Fig. 2. Simplified block diagram of the force control system.

When using the incremental encoder as the position/velocity sensor, we initially set the absolute zero degree angle of the mechanical arm using the mechanical stopper and velocity feedback control. Initial setting of the absolute arm angle can also be accomplished redundantly using multiple inclinometers, but in this case the arm-force generation mechanism becomes more complicated, and possibly more expensive.

IV. Software Structure

The software for the arm-wrestling robot is programmed with C language using multithreading function, *pthread*. The whole program is composed of 6 tasks, and each task corresponds to each interrupt service routine. Six tasks are defined as in Fig. 3, in which task 1 and task 2 run in parallel, task 3 and 4, and task 5 and 6 run in parallel, too. Task flow with respect to time is shown in Fig. 3.

Arm wrestling is not strictly a strength sport, as people often think, because technique and speed are both very important. Therefore, a key element of the arm-wrestling robot is to create an intelligent game scenario online. In the Robo Armwrestler, the inference engine generates an appropriate force profile for each match, which considers a human's maximum force, a human's force pattern, time duration, a human's will to win, and randomness. Force control logic enables the robot arm to follow the generated force profile as smoothly as possible. In the developed system, winning or losing is not predetermined, but varied online according to the pattern of the game progression.



ISR_TASK1 : Wait a person.
ISR_TASK2 : Detect a person.
ISR_TASK3 : Output voices and images.
ISR_TASK4 : Generate scenarios and control the motor.
ISR_TASK5 : Determine the absolute initial angle of the arm.
ISR_TASK6 : Output warning messages.

Fig. 3. Task definitions and task flows

For a few seconds in the early stage of the game, the inference engine detects the human's maximum force according to the procedure. The system increases force up to the specified value with a parabolic fashion for a short time, and then the robot measures arm velocity at every 0.1 second during the next few seconds. If the velocity is positive, then the robot increases the force until the velocity becomes negative, and memorizes the force value.

To realize unpredictable and intelligent game patterns, we adopt a random number and a value called *will point* that quantifies the will of the arm-wrestler to win the match. If the will point is near 100, the user is considered to have a strong desire to win the match. If the will point is near 0, the user is considered to have a weak desire to win the match. The will point is calculated by

will point = (average arm-force during one sub-scenario)/(maximum arm-force of the user) x *100.*

The game scenario is composed of several sub-scenarios. Each sub-scenario has a different rotation angle limit of robot arm within 150 degrees, and is divided into three classes; win sub-scenarios, draw sub-scenarios, and defeat sub-scenarios. If a sub-scenario is selected by means of random number generation, the robot decreases or increases the force during randomly determined intervals of time. During these intervals of time, human force is measured and averaged in order to calculate the will point. As soon as the execution of a sub-scenario is completed, the next sub-scenario is immediately prepared. This sub-scenario may be generated online at that instant, or may be selected among many sub-scenarios prepared in advance.

Arm wrestling progression is affected by the will point and the prespecified probability. For example, if the obtained will point is 86, then the class of win, draw, or defeat scenario is determined according to winning probability with 8 %, drawing probability with 90 %, and defeat probability with 2 %. This class determination is conducted using a random number 0 ~ 99, that is, the generated random numbers 98 and 99 imply the defeat class, random numbers 8 to 97 imply the drawing class, and random numbers 0 to 7 imply the winning class.

Using another random number, we select a sub-scenario randomly within sub-scenarios of the determined class. If the selected sub-scenario is a drawing one, then the will point is recalculated after the sub-scenario ends, and the above procedure is repeated. If the selected sub-scenario is a win or a defeat one, then the win or defeat sub-scenario is progressed, and the human wins or defeats, and the arm wrestling ends. Finally, the arm-wrestling system is initialized for the next match.

V. Human-Computer Interaction

In an idle situation, the computer plays music, and waits for a person. As a person approaches the robot, the computer automatically detects his approach, greets him, and encourages him to arm-wrestle. If the person sits down, the computer guides him to start the game.

The human-computer interaction of the Robo Armwrestler is composed of four steps. The first step includes initializing the arm-force generation mechanism and the control system, transmitting an initialization completion signal to the motor power control part, applying power to the motor driving part, and setting an initial absolute angle of the mechanical arm.

The second step includes detecting a human's approach to the robot through ultrasonic sensors, guiding the human with voice and image messages if the human is detected, or repeating the process if not detected, detecting the human's sitting on the chair, and guiding the human, or repeating the process if not detected.

The third step includes increasing the torque acting on the mechanical arm up to a specified value, determining whether the velocity of the mechanical arm is positive or not, increasing the torque acting on the mechanical arm with a specific rule if the velocity is positive, decreasing the torque acting on the mechanical arm if the velocity is negative, and repeating the above process during a specified time interval, and determining the user's maximum arm-force.

The fourth step includes selecting sub-scenarios among winning, drawing, and losing sub-scenarios, calculating the will point of the player, selecting the next sub-scenario according to the will point, and repeating or ending the match according to the selected sub-scenario.

The fourth step can be implemented differently from the above by generating sub-scenarios online, which are characterized by three parameters, i.e., force increment, rising time, and maintaining time. All three values are determined using a random number and the will point. The force increasing or decreasing in a sub-scenario is achieved by polynomial curves.

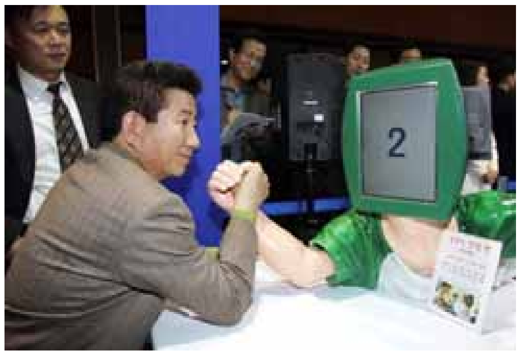


Fig. 4. An arm-wrestling match between the President of Korea, Roh Moo-hyun, and the Robo Armwrestler at the Future Tech Korea 2005 Exhibition.

Operation experiences at the laboratory and the Future Tech Korea Exhibitions in October 2005 and 2006 held in Seoul, Korea, have revealed that the Robo Armwrestler generates intelligent scenarios unpredictably and reliably, and controls the robot arm-force properly so that the human arm-wrestler feels as if he is arm-wrestling against a human. Fig. 4 shows a scene that the President of Korea, Roh Moo-hyun, arm-wrestles with the Robo Armwrestler at the Future Tech Korea 2005 Exhibition.

When a 72-year-old woman arm-wrestled with the robot, she won one time and lost another time. Also, the results showed that force patterns generated by the arm-wrestling robot and the elapsed time of arm wrestling are different from match to match even if the same person plays.

Another experiment was conducted as follows. As soon as a youth with strong arm force around 50 N·m finished a match, a 10-year-old boy with around 20 N·m arm-force began another match. In this case, the arm-wrestlings proceeded smoothly without changing any parameters of the arm-wrestling robot.

When one user played arm wrestling 26 times with the Robo Armwrestler, the elapsed time varied each time (average = 17 sec) and winning average was 63%.

VI. Conclusion

In this paper, we have presented the configuration of the arm-wrestling robot, implementation of real-time force control, design of the software structure, generation of intelligent arm-wrestling scenarios, and human-computer interaction. The validity of the proposed system is verified through experimental studies, and is demonstrated at the Future Tech Korea 2005 and 2006 Exhibitions. Although the robot works as expected with the designed autonomy and reasonable control performance, we plan to further pursue research in order to build the arm-wrestling robot capable of recognizing facial expressions of the human using a webcam, and thus emotionally communicating with the human player. Moreover, we plan to add more degree-of-freedom for more human-like motion, and to eventually integrate arm-wrestling function into a humanoid robot.

References

[1] Y. Bar-Cohen, “Electric Flex,” IEEE Spectrum, vol. 41, no. 6, pp. 28-33, 2004.

[2] G. Kovacs, and P. Lochmatter, “Arm wrestling robot driven by dielectric elastomer actuators,” in Proc. of SPIE, San Diego, CA, 2006, vol. 6168, pp. 1-12.

[3] E. Iversen, H.H. Sears, and S.C. Jacobsen, “Artificial arms evolve from robots, or vice versa?” IEEE Control Systems Magazine, vol. 25, no. 1, pp. 16-20, 2005.

[4] John M. Hobby, Jr., "Arm wrestling unit," US patent no. 3947025, 2004.

[5] Fernando P. Rufa, "Arm wrestling device," US patent no. 5842958, 1998.

[6] Taihei Giken Kogyo KK, “Hand wrestling game machine,” Japan patent no. 6-315544, 1994

[7] Shimizu Toshio, “Muscular force strengthening device for arm wrestling,” Japan patent no. JP2002017891, 2002.

Managing XML Versions and Replicas in a P2P Context

Deise de Brum Saccol1[1,2], Nina Edelweiss[2], Renata de Matos Galante[2,4], Carlo Zaniolo[3]
[2]*Instituto de Informática - Universidade Federal do Rio Grande do Sul (UFRGS)*
Av. Bento Gonçalves, 9500, Porto Alegre, RS, Brazil
{deise, nina, galante}@inf.ufrgs.br
[3]*Computer Science Department – University of California (UCLA)*
405 Hilgard Avenue, Los Angeles, CA, United States
zaniolo@cs.ucla.edu

Abstract

Peer-to-Peer (P2P) systems seek to provide sharing of computational resources, which may be duplicated or versioned over several peers. Duplicate resources (i.e. replicas) are the key to better query performance and availability. On the other hand, multiple versions can be used to support queries on the lineage of resources and the evolution of history. However, traditional P2P systems are not aware of replicas and versions, which cause complexity at the logical level and inefficiency at the physical level. To solve these problems, we propose an environment for detecting, managing and querying replicas and versions of XML documents in a P2P context. We also show that the proposed environment can also be used for plagiarism detection, web page ranking, and software clone identification.

1. Introduction

P2P systems refer to a class of applications that use distributed resources to perform tasks in a decentralized context. Each participant acts both as client and server, providing access to resources through direct and decentralized communication [1]. Their usability is mainly dependent on techniques used to find and retrieve results. The results quality may be measured by metrics such as the result set size, query satisfaction, and processing time [2].

However, searching for resources in P2P systems must deal with two important issues: the first is the existence of replicas and the second is the presence of multiple versions of a resource. Replicas (i.e. multiple representations) are important for performance optimization: when the user poses a query then the results must be returned from peers that best satisfy performance and fast response time requirements. To take advantage of resource replication it is necessary to detect these replicas; otherwise, redundant results at a high processing cost are returned to the user.

The second problem arises from the evolving behavior of some resources, which is a fundamental aspect in persistent information systems. This feature is even more evident in XML domain, with frequent structure and content changes. The evolution aspect must be managed to allow historical analysis for dynamic resources.

The applications of the version concept are many and diverse, for instance the management of the co-authoring software, as studied in [3]. However, past approaches focus on centralized management and truly P2P distributed management still represents a difficult challenge. In P2P systems, versioning techniques must consider that versions and replicas may be spread over several peers. In such context, detecting duplicates and versions is mainly useful for query optimization. To address this issue, our paper proposes *DetVX*, an environment for the detection, management and querying of XML replicas and versions.

The main contributions of this paper are:

- A brief environment specification to detect, manage and query replicas and versions in a P2P environment;
- A replica and linear version detection mechanism based on *hash* functions and document similarity;
- A temporal XML model, based on *diff* algorithms and timestamps, for representing versioned resources and supporting basic temporal queries.

The paper is organized as follows: Section 2 presents related works. Section 3 briefly describes the proposed environment. Section 4 discusses the replica and version manager module; a similarity function is presented for version detection in content and structural evolutions. Query capabilities are presented in Section 5. Section 6 highlights other applications that may use our mechanism. Section 7 presents conclusions and future work.

[1] This work has been partially supported by CNPq under grant No. 142396/2004-4, Capes under grant No. 1451/06-5, PERXML under grant No. 475.743/2004-0 and DIGITEX - CTInfo under grant No. 550.845/2005-4.

[4] This work has been partially supported by CNPq under grant No. 481516/2004-2 (Edital Universal) and Fapergs under grant No. 0412264 (Auxílio PROAPP).

2. Related Work

There has been some recent works on temporal XML models [5][6], extensions to its query languages [7], temporal libraries [8] and version control [4]. However, version control systems model files as text line sequences, storing the last version and using reverse editing scripts to retrieve previous versions [21]. These systems do not preserve the logic structure of the original file and do not support complex queries, and thus are inadequate to support XML versions. These gaps are addressed in some works, such as [9][10] and [11][12], respectively.

Previous works focus on version management rather than version detection (i.e. the creation of a new version from an old one). However, version detection is essential in our motivating application, since the anonymity/distributed nature of P2P environments prevents users from identifying resources from which the new version or replica is being created. Moreover, existent replica detection proposals focus on identifying multiple representations of the same object in the real world [13], which may have content or structure differences. However, our work considers a replica as an identical copy of a XML file.

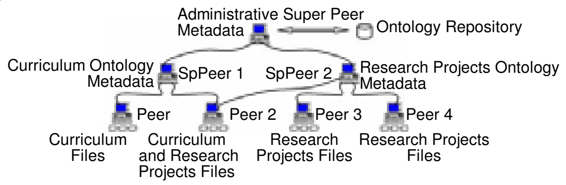
To address this issue, we propose a detection mechanism based on file similarity. There is some research on change detection that can be used as a basis for measuring similarity. Some approaches use *diff* algorithms to detect differences between files [14][15]. Another possibility is to analyze their ordered tree representations by calculating the *edit distance,* i.e. the minimum cost to transform one tree into another tree using basic operations [16][17].

Diff algorithms can be used to detect differences and, in a certain way, a similarity value between files. However, *diff* results are a delta script with no semantic information regarding the similarity between documents. Also, the tree edit distance results do not contain valuable information related to the similarity level that could be used to detect resource versions. Our work focuses on this gap and proposes an environment for detecting and managing replicas and versions of XML documents in a P2P context.

Many applications may use version detection mechanisms. For plagiarism detection, comparing file checksums is enough for detecting exact replicas, but insufficient for partial copies [22][23]. By considering partial copies as versions, such plagiarism can be detected. The web page ranking process can also take advantage of the detection mechanism by ranking new versions of existent top-ranked pages [25]. At last, the software clone problem that arises during the development of systems may have a negative impact on their maintenance [24]. The proposed mechanism can help to detect such clones.

3. DetVX Environment

DetVX is an environment for detecting and managing replicas and versions of XML documents in a P2P context [26]. *DetVX* is based on a super peer architecture [19]. Super peers are responsible for receiving the query and resending it to aggregated peers and other super peers. Peers must (re)connect in super peers in order to share their files. Shared XML files are related to a knowledge domain, used as a peer grouping criterion in super peers. An ontology is used to represent the knowledge domain [18]. Super peers are managed by the administrative super peer, as depicted in Figure 1.



Administrative Super Peer Metadata
Ontology Repository
Curriculum Ontology Metadata
SpPeer 1
SpPeer 2
Research Projects Ontology Metadata
Peer
Peer 2
Peer 3
Peer 4
Curriculum Files
Curriculum and Research Projects Files
Research Projects Files
Research Projects Files

Fig. 1. *DetVX* Environment

Files may be duplicated or versioned over the super peer network. To provide the functionalities for replica and version detection, this work proposes the following modules. The *peer manager* is responsible for (re)connecting peers and periodically verifying modifications in shared files. The *ontology manager* maintains the ontology repository and associates ontologies to super peers. The *replica and version manager* identifies and manages document replica and versions. The *query processor* is responsible for verifying the query domain and rotating queries to peers. Metadata play a fundamental role and are detailed in Section 3.1. In this paper, we do not detail the peer and ontology managers. More details may be found in [26].

3.1 Document and Metadata Representation

The term *file* refers to a physical representation stored in a peer; *document* refers to the representation of an object in the real world. In other words, one document can be stored as many files, either because it is replicated or versioned. A file has a registering and a modification time. The modification time is considered to define the file order over time. Local (`fileID`) and global identifiers (`GFID`) are used to identify a file in a peer and in a specific location in the network, respectively [26]. Documents also have identifiers (`docID`) and they are used to identify versions and replicas of the same object in the real world.

To manage identifiers and other relevant information, the approach relies on the extensive use of metadata. Metadata are represented as XML files and classified in two levels, as shown in Figure 1. In this paper, only super peer metadata are considered. Super peer metadata basically specify the available versions and replicas in a specific super peer (`superPeerId`), and the corresponding timestamps for each element (`timeStart: TS, timeEnd: TE`) that is found in certain file (`fileID`) in a peer (`peerID`), as shown in Listing 1.

```
<Metadata superPeerId="SP1">
  <document docID=”D1” fileID="F7" HDoc=’’YES”>
    <version versionID="1" peerID="P1" registeringTime="10/10/2005"
       modificationTime="08/08/2004" duplicate="no"
       hashResult="d49622ddab3733549e54749755fd52b5">
      <element name="author" TS="08/08/2004" TE="10/15/2004"/>
      <element name="address" TS="08/08/2004" TE="10/15/2004"/></version>
    <version versionID="2" peerID="P2" registeringTime="11/20/2005"
       modificationTime="10/16/2004" duplicate="yes"
       hashResult="7c00bb062edc60fa548729a3d55c04fd">
      <locationDuplicate>Peer 3</locationDuplicate>…</version>
</Metadata>
```

Listing 1. Super Peer Metadata

Each element has two timestamps inferred from the modification time of the file in which the element is contained. Super peer metadata information is updated whenever a new file is registered into a peer and is extensively used during querying process.

4. Replica and Version Manager

This module is responsible for detecting replicas and versions and representing the history in a new structure, called *H-Doc* file.

4.1 Detection Mechanism

To solve the detection problem, a first approach is to look for replicas and versions in the local peer. If they are not found, the detection is executed in the next peer of the super peer network. The detection mechanism is executed whenever a file is registered or updated in a peer. When a file is removed, only the metadata need to be updated. A peer modification checking service is responsible for periodically watching the peer and notifying its super peer whenever a change is detected.

The replica detection mechanism aims to verify if a file is a copy of any other file stored in any peer belonging to the same super peer network. In our work, a duplicate (or replica) is defined as an identical copy of a XML file. The replica detection is done by comparing the file hash result with all the hash results already stored in its super peer metadata. Two files *f1* and *f2* are replicas if:

```
HashFunction(f1)=HashFunction(f2)
```

The version detection mechanism aims to verify if a modified file is a version of any other file stored in any peer belonging to the same super peer network. Since this work assumes the linear versioning approach, this activity will compare the candidate file only with the last file versions available in the super peer network.

There are two types of evolution that are considered:

- Content: <x>A St, 7</x> $\Rightarrow$ <x>B St, 8</x>
- Structure and content: <x>A St, 7</x> $\Rightarrow$ <y>B St</y> <z>8</z>

In this proposal, version detection is based on file similarity. The general idea is that two files with high similarity are considered two versions of the same document; two different documents, otherwise.

Let's first consider the content evolution type.

4.1.1 Content Evolution

Suppose two files, *f1* and *f2*, shown in Listing 2.

```
<employee>                              <employee>
  <name>Marcos</name>                     <name>Marcos</name>
  <hiringDt>10/10/03</hiringDt>           <hiringDt>10/10/03</hiringDt>
  <job>engineer</job>                     <job>manager</job>
  <salary>3700</salary>                   <salary>4900</salary>
  <address>7 St</address>                 <address>7 St</address>
  <phone>65982541</phone>                 <phone>65982541</phone>
</employee>                             </employee>
```

Listing 2. XML Files

In order to evaluate the similarity between these files, some features are observed:

- ***Diff* results:** the root element in both files has six child elements. Using a *diff* algorithm, the differences between the files are detected. As Listing 3 shows, the content of the elements *salary* and *job* do not match in the second file. In other words, 67% of the original elements kept unchanged in the second file.

The assumption here is the following: the bigger percentage of matched elements, the bigger chance the files are versions of the same document.

```
<delta> <Deleted update="yes"  pos="0:0:3:0">3700</Deleted>
        <Deleted update="yes" pos="0:0:2:0">engineer</Deleted>
        <Inserted update="yes"  pos="0:0:2:0">manager</Inserted>
        <Inserted update="yes" pos="0:0:3:0">4900</Inserted> </delta>
```

Listing 3. *Diff* result[2] for files *f1* and *f2*

- **Matched and unmatched elements:** We consider the term *matched* to refer to an element that has the same content in both files (for example, *name*); *unmatched*, otherwise (for example, *salary*). Let's take a look at the unmatched elements *salary* and *job*. Using a (combination of) string similarity function(s), we calculate a value that demonstrates how similar the unmatched elements are. The more similar the respective unmatched elements, the bigger chance the files are versions of the same document.
- **Element change relevance:** Another important issue is the relevance of individual changes. Some domain concepts can change more frequently than others. Let's suppose that we have an *address* element. Two different addresses can easily refer to the same person; however, two different birthdates suggest that we are analyzing two different objects in the real world. In other words, the change relevance is differently weighted for different concepts. We assume different weights, such as *high* (1), *medium* (0.5) and *low* (0). The average of weighted relevances is used to calculate file similarity. The smaller change relevance they present, the bigger chance the files are versions of the same document.

Based on the previous discussions, the similarity function *simC* between two files *f1* and *f2* is defined as:

$$simC(f1,f2) = (w_1*F_1 + w_2*F_2 + w_3*F_3 + ... + w_n*F_n)$$

Where w_n is a factor that weights the importance of a specific feature F_n. A factor may be positive or negative (if it influences the similarity growth or reduction, respectively). Considering $w_x, w_{x+1},...w_y$ as positive factors and $w_z, w_{z+1},...w_q$ as negative factors, we assume that $w_x + w_{x+1} +...+w_y = 1$ and $0 <= w_z + w_{z+1} +...+w_q <= 1$.

In our approach, three features are considered to produce the following content evolution similarity function:

$$simC(f1,f2) = w_1*P + w_2*S + w_3*R$$

Where: *P* is the percentage of matched elements, *S* is the mean similarity of the unmatched elements and *R* is the average of domain relevances of the unmatched elements (defined by the system administrator). *P* and *S* factors (w_1 and w_2, respectively) are positive values (the greater these values, the more similar the files) and *R* factor (w_3) is a negative value (the smaller this value, the less relevance the change and the more similar the files). The factors ($w_1, w_2, ..., w_n$) must be defined based on the importance of the three features in

[2] We are currently using *XyDiff* implementation [14], but the architecture allows changing to other *diff* algorithms.

specific applications/domains and recall/precision measures [30].

The intervals of the defined variables are defined as: `{P|P ∈ [0,1]}, {S|S ∈ [0,1]}, {R|R ∈ [0,1]`. Analyzing the minimum e maximum values of *P*, *S* and *R*, and the sum restrictions for positive and negative factors, we conclude that the similarity function produces a value *simC* that ranges from -1 to 1, i.e. `{simC|simC ∈[-1, 1]}`.

To calculate *P*, we use a function *calcP* that returns the percentage of matched elements based on the *diff* result. *S* is calculated by using a (combination of) string similarity function(s) *(StrSim())* and it is defined as the average of unmatched elements (*ue*) similarity values. The function is defined in more details as follows:

$$simC(f1,f2) = w_1{*}calcP(diff(f1,f2)) + w_2{*}\frac{\sum_{x=1}^{t} StrSim(ue1_x,ue2_x)}{t} - w_3{*}\frac{\sum_{x=1}^{t} R(ue_x)}{t}$$

As depicted in Figure 2(a), the similarity function values are not uniformly distributed. To uniformly distribute the values, we sort and map the *m* similarity function results into *n* classes. The mapping, represented in a transformation table, categorizes *m/n* members in each class. Since we have 100 different similarity values, this transformation generates *0.01*m* members in each class.

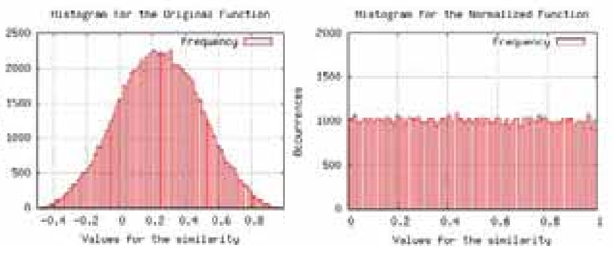


Fig. 2. Similarity Function Values (a) and (b)

Figure 2(b) shows the distribution of the mapped uniform transformation. We generated 1.000.000 values according to the original similarity function, using 0.5, 0.5 and -0.5 as the weight values, and grouped them into 100 classes. These classes were mapped to values `∈ [0,1]`, in order to uniformly distribute the function values. To ensure that the mapping is correct, we generated more 100.000 values and mapped them to this table.

After producing the similarity values, a threshold is used to detect versions based on them. The threshold generation is an ongoing work and it is not detailed in this paper. Further study is still needed to assess which threshold is better respect to precision and recall.

4.1.2 Structure and Content Evolution

Suppose two files, *f3* and *f4*, shown in Listing 4.

```
<employee>                                  <employee>
  <name>Marcos</name>                         <name>Marcos</name>
  <hiringDt>10/10/03</hiringDt>               <salary>4500</salary>
  <job>engineer</job>                         <address>7 St</address>
  <salary>3700</salary>                       <phone>65982541</phone>
  <phone>65982541</phone>                   </employee>
</employee>
```

Listing 4.XML Files

In order to evaluate the similarity between these files, the discussions about *diff results* and *element change relevance* in the last sub-section are still valid. Another feature is also observed:

- **Added and removed elements:** using a *diff* algorithm, the differences between the files are detected. Analyzing the files and the *diff* results, we can see that the *f4* has added one element (*address*) and has removed two elements (*job* and *hiringDt*). Let's refer *added* to the elements in the first situation and *deleted* to the elements in the second situation. These concepts are similar to the ideas presented in [29], which consider *plus*, *minus* and *common* elements for measuring similarity between a document and a DTD.

We consider the term *matched* to refer to an element that has the same structure and content in both files (for example, *name* and *phone*); *unmatched*, for those elements that the content has changed (for example, *salary*). Similar to the ideas presented for the content evolution, the following features are considered to produce the structure evolution similarity function:

$$simE(f3,f4) = simC(f3,f4) + w_4{*}A + w_5{*}D$$

Where: *simC* is the content similarity value, *A* is the percentage of added elements and *D* is the percentage of deleted elements. *A* and *D* factors (w_4 and w_5, respectively) are negative values (the smaller these values, the more similar the files).

The intervals of the defined variables are defined as: `{A|A ∈ [0,1], {D|D ∈ [0,1]`. Analyzing the minimum e maximum values of *simC*, A, *D*, and the sum restrictions for positive and negative factors, we conclude that the similarity function produces a value *simE* that `∈[-3, 2]`.

To calculate *A*, we use a function *calcA* that returns the percentage of added elements, based on the *diff* result. To calculate *D*, we use a function *calcD* that returns the percentage of removed elements, based on the *diff* result.

$$simE(f3,f4) = simC(f3,f4) - w_4{*}calcA(diff(f3,f4)) - w_5{*}calcD(diff(f3,f4))$$

The similarity values are not uniformly distributed. Similarly, the process detailed in the previous section is applied on the results to uniform these values. Also, the threshold process presented in Section 4.1.1 is still valid.

Whenever a new version or replica is detected, the timestamps described in the super peer metadata need to be updated. Metadata updating is described in [26].

4.2 A Consolidated Historical Representation

After detecting the versions, the system stores them in a new physical file, which contains the entire history of a document. The document history is named *consolidated historical representation* and represented in *H-Doc* files. *H-Doc* files are stored in the respective super peer where the original versions are registered. Timestamps are responsible for validating data in specific versions. *H-Doc* representations are generated only for frequently accessed and evolved files. The goal is to provide faster query processing for queries that ask historical retrieval.

The *H-Doc* generation process is detailed in [27]. Listing 6 shows the *H-Doc* file generated for Listing 4. Consider that *f3* and *f4* have *01/01/2004* and *01/01/2005 as m*odification times, respectively.

```
<employee TS="01/01/2004 TE=NOW">
        <name TS="01/01/2004" TE="NOW">Marcos</name>
        <hiringDt TS=01/01/2004 TE="12/31/2004">10/10/03</hiringDt>
        <job TS="01/01/2004" TE="12/31/2004">engineer</job>
        <salary TS="01/01/2004" TE="12/31/2004">3700</salary>
        <salary TS="01/01/2005" TE="NOW">4500</salary>
        <phone TS="01/01/2004" TE="NOW">65982541</phone>
        <address TS="01/01/2005" TE="NOW">7 St</address>
<employee>
```

Listing 5. *H-Doc File*

In *DetVX* environment, the generation of the *H-Doc* file is done by *XVersion* tool, a currently implementation work [27], based on *diff* algorithms and timestamps.

5. Query Processor

After detecting replicas and versions, temporal queries may be posed on the original files located in the peers or on the historical representation stored in the super peers.

5.1 Querying the Original Files

To evaluate which files must be accessed to answer a query, our approach relies on metadata described in Section 3.1. The query submission works as follows: the user poses a query in a specific peer (named *querying peer*). This query belongs to a specific domain. Looking at the super peer metadata, it is possible to see how to access the history or versions of an element or document.

Considering the super peer metadata described in Listing 1, some temporal retrieving examples are described below:

1. Retrieve the version v_i of an element e_j – for instance, get the first version (`versionID="1"`, line 3) of the element *author* (`element name="author"`, line 6). By searching the version number represented in metadata, the system can verify that the first version of the queried element is found in *peer* 1 (`peerID="P1"`, line 3) located at *super peer* 1 (`superPeerId="SP1"`, line 1). Thus, the system must access this file and return the results.
2. Retrieve the history of an element e_j – for instance, get the history of the element *address*. To answer this query, the system searches the metadata, looking for all the versions (`versionID`) of the element *address* (`element name="address"`). The last version of this element is represented by `TE=now`. Another possibility for this query is to check if there is a generated *H-Doc* representation for this file (attribute `HDoc="YES"`, line 2). In this case, the system can access this file in the super peer, as described in the next section.

5.2 Querying the H-Doc File

Consider a document *D* as a *n-tuple* $D = (root, e_1, e_2,..., e_n)$ and an element *e* in this document as a 3-tuple $E=(TS, TE, <content>)$, where *TS* and *TE* denote the timestamps. Temporal restrictions are applied based on a specific date *x* or on an interval *x* and *y* ($x<y$). Some temporal clauses are:

1. *Select_Before (E, x)*: returns the elements *e* that are valid in *H-Doc* file before *x* (elements whose $TS < x$);
2. *Select_After (E, x)*: returns the elements *e* that are valid in *H-Doc* file after *x* (elements whose $TE>x$);
3. *Select_Between (E, x, y)*: returns the elements *e* that are valid in *H-Doc* file between *x* and *y* (elements whose $TS<=y$ and $TE>=x$);
4. *Select_Now (E)*: returns the elements *e* that are valid in *H-Doc* file in current time (elements whose *TE=now*);

The same clauses are defined for retrieving entire documents, such as *Select_Before (D, x)*, *Select_After (D, x)* and others. Query capabilities based on *XQuery* language [20] have been implemented in our tool named *XVersion.* This tool generates the *H-Doc* document and allows basic temporal queries over the historical file. More details about *XVersion* may be found in [27].

6. Other Applications

This paper focuses on version and replica detection problem in P2P systems. Although this is the motivating scenario for our system and experiments, we expect that our proposal can be used in other applications, such as:

▪ **Web page ranking**: ranking methods usually involve the location and frequency of keywords in a web page. Search engines verify if the searched keywords appear close to the page top (headline or in the first few paragraphs). Frequency is also considered by analyzing how often keywords appear in relation to other words in a web page [25]. Another factor that may be considered for ranking is the incoming link degree (i.e. the number of links that point out to a page *p*). However, new *p* versions may have a small incoming link degree, mainly because of the pages that were pointing to *p* are not aware of the new version. In such context, version and replica detection may be useful for ranking new versions even if they have low incoming degrees.

▪ **Plagiarism detection:** Digital files may be easily copied, either partially or completely. One way to detect plagiarism is by comparing file checksums, which is simple and suffices for reliably detecting exact copies. However, detecting partial copies is more complicated [22]. By using the mechanism proposed in this paper, similar files are identified. The threshold definition must be in accordance to such application. For instance, partial copies must be identified with a low threshold, whereas complete copies must be detected with a higher threshold.

▪ **Software clone identification**: replicated code can arise during the development and evolution of software systems and it has a negative impact on their maintenance. The detection gets difficult mainly because of small differences, such as reformatting, code and variable name changes [24]. Existent detection mechanisms usually rely on the use of a parser, but this approach is dependent on the programming language syntax. The classical plain-text representation of code is convenient for programmers but requires parsing to uncover the deep structure of the program. Representing code in a structured format, as XML documents [28], permits easy specification of numerous software-engineering analyses by leveraging on the abundance of XML tools and techniques. In this context, the proposed mechanism may be used for software clone detection.

7. Concluding Remarks

This paper focused on detection and management of XML replicas and versions in P2P contexts. The relevance of such problem is quite evident in many scenarios, such as plagiarism detection, web page ranking, software clone identification, assuring link permanence in Web documents, and enhancing search in P2P systems. To increase efficiency and effectiveness in such systems, this paper briefly described the proposed architecture and functionalities of the *DetVX* environment.

We have proposed a simple structure for representing metadata which can be used for managing and querying the available files. A document similarity function used as the basic idea in the detection mechanism was also described. The proposal requires no intervention by the user. The user is only requested to update the document and register the file; the system detects prior versions or duplicates, generates identifiers and manages all the related metadata.

The current state of the project is as follows. We have already implemented *XVersion*, a tool for representing and querying document history. Basic retrieval capabilities have been implemented, allowing simple temporal queries over the historical representation. As future work, we are going to incorporate the detection mechanism in *DetVX* environment. The completion of the detection mechanism will allow us to measure improvements on selected testbeds**,** including JXTA [31]. Results will be presented in the conference.

References

1. Aberer, K. and Hauswirth, M.. An Overview on Peer-to-Peer Information Systems. Workshop on Distributed Data and Structures, Paris, France, 2002.
2. Yang, B. and Garcia-Molina, H.. Efficient Search in Peer-to-Peer Networks. In: Proceeding of the Intl. Conf. on Distributed Computing Systems, Vienna, Austria, 2002.
3. Westfechtel, B., Munch, B. P., and Conradi, R. A Layered Architecture for Uniform Version Management. IEEE Trans. Software Eng., 27(12):1111–1133, 2001.
4. Chien, S-Y., Tsotras, V. J., Zaniolo, C. (2001). XML Document Versioning. SIGMOD Records, Vol. 30 Number 3, Sept.
5. Su, H., Kramer, D., Chen, L., Claypool, K. T., Rundensteinrer, E. A.. XEM: Managing the Evolution of XML Documents. Proc. of 11th Intl. Work. on Res. Issues in Data Engineering, Heidelberg, 2001.
6. Grandi, F. and Mandreoli, F.. The Valid Web: an XML/XSL Infrastructure for Temporal Management of Web Documents. Proc. of Advances in Information Systems, 2000.
7. Gao, D. and Snodgrass, R.T.. Temporal Slicing in the Evaluation of XML Queries. Proc. of Very Large Database Systems, 2004.
8. Wang, F. and Zaniolo, C.. Representing and Querying the Evolution of Databases and their Schemas in XML. In Workshop on Web Engineering, SEKE, San Francisco, USA, 2003.
9. Chien, S.; Tsotras, V.; Zaniolo, C. and Zhang, D.. Storing and Querying Multiversion XML Documents using Durable Node Numbers. Proc. of the 2nd Intl. Conf. on Web Information Systems Engineering, 1, 232-241, vol.1, 2001.
10. Grandi, F., Mandreoli, F., Tiberio, P.. Temporal Modeling and Management of Normative Documents in XML Format. Data & Knowledge Engineering, v. 54, n. 3, p. 327-354, Sept., 2005.
11. Vagena, Z. and Tsotras, V.. Path-Expression Queries over Multiversion XML Documents. Proc. of Intl. Workshop on the Web and Databases, 49-54, 2003.
12. Wang, F. and Zaniolo, C.. An XML-Based Approach to Publishing and Querying the History of Databases. World Wide Web: Internet and Web Information Systems, 2005.
13. Weis, M. and Naumann, F.. Detecting Duplicates in Complex XML Data. Proc. of the 22nd Intl. Conf. on Data Engineering, 2006.
14. Cobena, G., Abiteboul, S. and Marian, A.. Detecting Changes in XML Documents. Proc. of 18th Intl. Conf. on Data Engineering, 41-52, 2002.
15. Wang, Y., DeWitt, D. J., Cai, J. (2003). X-Diff: An Effective Change Detection Algorithm for XML Documents. Intl. Conf. on Data Engineering, 519-530.
16. Chawathe, S.S.. Comparing Hierarchical Data in External Memory. Proc. of the 25th Intl. Conf. on Very Large Data Bases, Morgan Kaufmann Publishers Inc., 90-101, 1999.
17. Wan, X. and Yang, J.. Using Proportional Transportation Similarity with Learned Element Semantics for XML Document Clustering. WWW '06: Proc. of the 15th Intl. Conf. on World Wide Web, ACM Press , 961-962, 2006.
18. Peres, A., Lopes, M., Corcho, O.. Ontological Engineering: with Examples from the Areas of knowledge Management, e-Commerce and Semantic Web. Springer, 1st edition, 2004.
19. Schollmeier, R.. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architetures and Applications. Proc. of the 1st Intl. Conference on Peer-to-Peer Computing, 27-29, Linköping, Sweden. IEEE Computer Society 2001.
20. XQuery 1.0: An XML Query Language. W3C Proposed Recommendation. Available at: http://www.w3.org/TR/xquery.
21. CVS: Concurrent Versions System. Available at: http://www.nongnu.org/cvs.
22. Schleimer, S., Wilkerson, D., Aiken, A.. Winnowing: Local Algorithms for Document Fingerprinting. Proc. of the ACM SIGMOD Intl. Conf. on Management of Data, San Diego, California, p. 76-85, 2003.
23. Chen, X., Francia, B., Li, M., McKinnon, B., Seker, A.. Shared information and program plagiarism detection. IEEE Transactions on Information Theory, v. 50, n. 7, p-1545-1551, 2004.
24. Ducasse, S., Niertrasz, O., Rieger, M.. On the effectiveness of clone detection by string matching. Journal of Software Maintenance and Evolution: Research and Practice, v. 18, n. 1, p. 37-58, 2006.
25. Baeza-Yates, R., Castillo, C.. Relating Web Characteristics with Link based Web Page Ranking. Proc. of the 8th Intl. Symposium on String Processing and Information Retrieval, 2001.
26. Saccol, D.B., Edelweiss, N., Galante, R.M.. Detecting, Managing and Querying Replicas and Versions in a Peer-to-Peer Environment. In: 1st IEEE TCSC Doctoral Symposium, in conjunction with the 7th IEEE Intl. Symposium on Cluster Computing and the Grid, Rio de Janeiro, 2007 (to appear).
27. Saccol, D. B.; Giacomel, F. S.; Galante, R. M.; Edelweiss, Nina.. Grouping and Querying XML Document Versions in a Peer-to-Peer Environemnt (in Portuguese). In: Actas do XATA-XML: Aplicações e Tecnologias Associadas, Lisboa, 2007.
28. Badros, G. J.. JavaML: A Markup Language for Java Source Code. In Proc. of the 9th Intl. Conf. on the World Wide Web, Amsterdam, 2000.
29. Bertino, E., Guerrini G., Mesiti, M.. A Matching Algorithm for Measuring the Structural Similarity between an XML Document and a DTD and its Applications. Information Systems, v. 29, n. 1, Special issue on web data integration, p. 23-46, 2004.
30. Baeza-Yates, R.A., Ribeiro-Neto, B.. A. Modern Information Retrieval. ACM Press / Addison-Wesley, 1999.
31. Gong, L.. JXTA: A Network Programming Environment. IEEE Internet Computing, 5(3):88–95, May/June 2001.

Knowledge sharing through a simple release planning method for web application development

Sven Ziemer and Ilaria Canova Calori

Norwegian University of Science and Technology, NO-7491 Trondheim, Norway
E-mail: {svenz|canovaca}@idi.ntnu.no

Abstract

Web application development is – under circumstances such as a strong emphasis on time-to-market – characterised by the use of informal and ad-hoc development practices and a lot of tacit knowledge. A recently proposed release planning method, that has been developed for the use in these environments, aims at bringing stakeholders together to share their knowledge and to decide on a configuration for the next release of a web application that satisfies all stakeholders. An initial experiment to evaluate the release planning method indicates that the method does well on supporting a consensus and stakeholder satisfaction, but it does not contribute to knowledge sharing between stakeholders in small projects. To study the effect that learning and experience from using the method several times has on knowledge sharing and understanding among members of a small development team, and to study the effect that a changed communication pattern has on knowledge sharing and understanding.

1 Introduction

Web applications that are developed with a strong focus on short time-to-market use development practices that can be characterised as informal, chaotic, rush-to-market and ad hoc [7] [11]. As a result of these practises, the knowledge that is available to make a decision regarding the development activities, is spread between all stakeholders, and is tacit, informal, and in many cases qualitative and based on the stakeholders opinion and beliefs. To make a decision that takes all available knowledge into consideration, it is necessary to bring the stakeholders together and to make them share their knowledge.

Release planning is an important decision for a software project. Release planning for web application development must balance important factors such as a short time-to-market, that is important to give an early return on the investments that have been made into the web application, and innovative and attractive functionality that aims at attracting the attention of potential users. In a recent paper we proposed a release planning method that brings together all available knowledge from the stakeholders and make them reach a consensus based decision on the next release [10].

Release planning is an important decision for a software project [2]. Moreover release planning for web-application development must balance important factors such as a short time-to-market, that is important to give an early return on the investment, and innovative and attractive functionalities that aims at attracting the attention of potential users. Many works refer to empirical studies on planning of software releases [4] [5] but there are not specific works about web application release planning. In a recent paper a release planning method has been proposed to bring together all available knowledge from the stakeholders and make them reach a consensus based decision on the next release [10].

To study the effect of this method on small development teams, we conducted an experiment with students at our university [9]. Based on the analysis of the results from this experiment we decided to repeat the experiment two times in two different settings, in order to study the effect of experience and learning on knowledge sharing and understanding and the the impact of changing the instructions on the communication pattern of a method with respect to knowledge sharing and understanding.

This paper describes the details of the experiments and reports the results and lessons learned from them. The paper is organised as follows: We give a more detailed background for our motivation to repeat the initial experiment in section 2. The experiment planning is given in section 3, and the results are shown in section 4. A discussion of relevant topics is presented in section 5, and conclusions are given in section 6.

2 Background

To make good decisions, the decision makers must have access to the knowledge that is available about the issue at

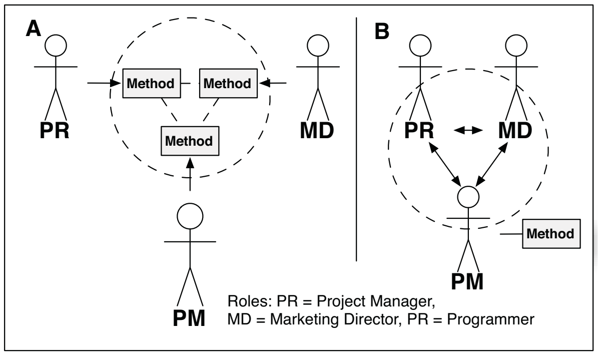

A
Method
Method
Method
PR
MD
PM
B
PR
MD
PM
Method
Roles: PR = Project Manager,
MD = Marketing Director, PR = Programmer

Figure 1. Two communication patterns

hand. When the knowledge leaves out certain aspects, this will be reflected in the decision and in its results. When a lot of the knowledge that a decision has to be based on is tacit, we need a different technique to share knowledge and create a common understanding than we need when the knowledge is explicit.

Web development projects involve often tacit knowledge, that is spread between all stakeholders. A recently proposed release planning method, aims at bringing together stakeholders, make them share their knowledge and create a common understanding, and to make a decision on the next release for a web application development project. To do so, the stakholders should perform a qualitative assessment on the expected return on value for every requirement and candidate configuration. They should use a a simple scale with five points, where ”1” stands for no value and ”5” for very high value. In addition, each stakeholder should give a short justification for his assessment. Based on the assessment requirements and candidate configurations are prioritised, and a final decision on the next release is made [10].

We decided that the proposed release planning method should be evaluated and validated experimental. We therefor planned an experiment to observe the effects the proposed method had when used by small development teams. The factors we decided to study were: knowledge sharing, understanding, prioritisation, reaching a consensus, and stakeholder satisfaction. Due to limited resources the experiment was planned as an off-line, student experiment, with one factor and with two treatments. The students were divided into a treatment group and a control group. Both groups had to participate in a role play and solve a release planning task. The treatment group used the proposed release planning method whereas the control group solved the task in an ad hoc manner. After the experiment all students had to fill in a post-experiment questionnaire that we used to analyse the effect of the release planning method on the aforementioned factors. The results showed that the treatment group did perform better on prioritisation, reaching a consensus and stakeholder prioritisation, whereas the control group performed better on knowledge sharing and understanding.

The results were not as expected and we could therefor not reject the null hypothesises stating that the release planning method did not perform different for knowledge sharing and understanding. When analysing and discussing the results we found two possible explanation for the unexpected results:

- When introducing a new method, the focus of the developers is on understanding the method and to use it correctly. The problem to be solved will consequently receive less focus, and the communication about the problem between the team members will be less intensive. With more experience in using the method, the understanding of the method is growing, and the developers will move their focus to the problem. The communication between the team members will also focus more on the problem again. The control groups had not to learn a new method and could use all their attention on the task.
- Another explanation is based on the communication pattern that was introduced with the instructions on how to use the release planning method and the handouts supporting the use of the method. Every subject on the treatment groups received a form where he could write down his notes on the problem during the experiment. This was done for the team members convenience, but seemed to have the unintended side-effect of disabling the communication between group members (see figure 1 A).

Our conclusions were that we would expect the treatment groups to perform better on knowledge sharing and understanding when running the experiment again and observing the effect (1) of the team members experience in using the method, and (2) of changing the communication pattern that is introduces to the treatment groups by the instruction they receive. The changed communication pattern is shown in figure 1 B, and aims to enable as much oral communication between the group members as possible.

We decided to investigate these explanations further, and performed four new experiments. The initial experiment is denoted E1 (see figure 2). Two experiments – E 2.1 and E 2.2 – are replications of the first experiment. Their results will – together with the results from the first experiment E1 – let us study the effect that learning and experience has on knowledge sharing and understanding. For the two other experiments – E 3.1 and E 3.2 – we will change the communication pattern in our instructions. Only the group leader will receive a form for taking notes, and the group members

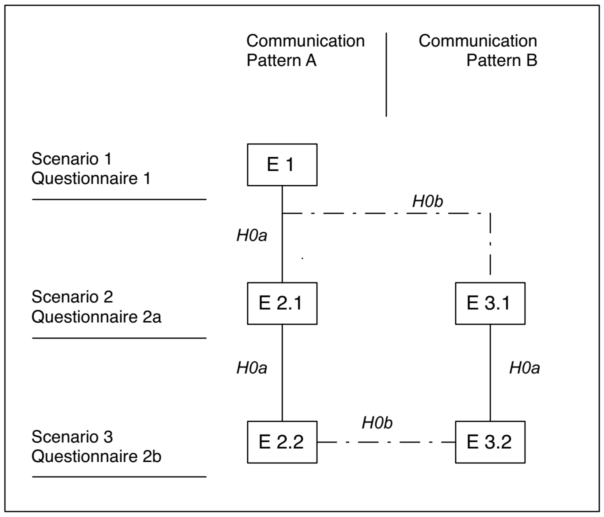

Communication Pattern A
Communication Pattern B
Scenario 1 Questionnaire 1
Scenario 2 Questionnaire 2a
Scenario 3 Questionnaire 2b
E 1
E 2.1
E 3.1
E 2.2
E 3.2
H0a
H0b

Figure 2. The organisation of experiments.

have to give their comments to the group leader orally. The result of this experiment can be compared to the results of E 1, E 2.1 and E 2.2, to study the effect that a changed instrumentation have on knowledge sharing and understanding.

3 The experiments

We planned the four experiment together. Figure 2 shows how the experiments are related and the factors that changed during the experimentation: The communication pattern, the scenario and the questionnaire.

Experiment definition The experiment definition is (using the template from [8]):

Object of study: The release planning method [10].
Purpose: Study the effect of learning and a changed instrumentation on knowledge sharing and understanding.
Quality focus: The stakeholders increased shared knowledge and understanding.
Perspective: The development teams point of view **Context:** Experiment with industrial economy and computer science students in their 3rd year of study, forming small groups of 3 or 4 members. The study is conducted as Multi-test within object study.

Experiment planning The four experiments have been planned as replications of the initial experiment E1. We refer to [10] for a detailed description of the experiment setting. However, even when the experiments are planned as replications of E1, we had to make some small changes to be able to reach the goal for these experiments:

- As indicated in section 2, we changed the communication pattern that we introduce in our instructions on how to use the release planning method and the handouts that each group receives. These changes apply to experiments E3.1 and E3.2. We recruited also a different student group for these two experiments. The students were new to the experiment and the proposed release planning method. Experiments E 2.1 and E2.2 are unchanged from E1, and where run with the same communication pattern, handouts, and student group.
- We created two new scenarios to be used in the role plays that take place in each experiment. Experiment E2.1 and 3.1 used the first scenario, and experiment E2.2 and 3.2 used the second scenario. Both scenarios are from the development of web applications. The first version of the new applications have to be deployed within three weeks, and the task of the groups is to find a release that satisfies all stakeholders.
- We used the same post-experiment questionnaire as in the first experiment. Based on the leasons-learned from the fist experiment, we added four questions in a new group of questions on communication, to get a better impression on the communication style in each group. In addition we changed the order of the questions in the questionnaire for experiments E2.2 and E3.2, and changed the wording of approximate 30 % of all questions to a negative wording. Otherwise, the questionnaire remained unchained.

Hypothesis formulation The hypothesises of the experiments are as follows:

H0a: There is no effect on knowledge sharing and understanding from learning when the users gain experience in the release planning method.
H1a: There is a effect on knowledge sharing and understanding from learning when the users gain experience in the release planning method.
H0b: The changed communication pattern used in the release planning method has no effect on knowledge sharing and understanding.
H1b: The changed communication pattern used in the release planning method has an effect on knowledge sharing and understanding.

4 Results

The experiments collect data by means of a post-experiment questionnaire, filled in be every participant or every experiment he participated in. The questionnaires were divided into six groups – knowledge sharing, understanding, prioritisation, consensus reaching, stakeholder

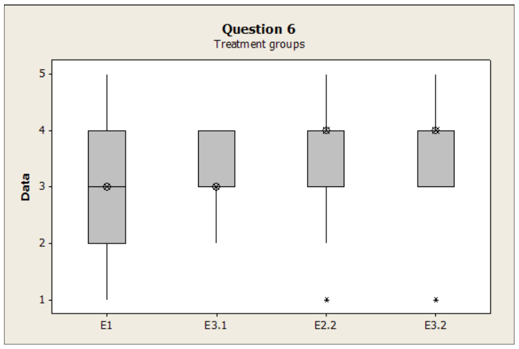


Figure 3. Boxplot for question 6 for E1, E3.1. E2.2 and E3.2

satisfaction and communication – and contained a total of 28 issues. For each issue, a statement is given together with a five-point Lickert scale. Each respondent used the Lickert scale to express his degree of agreement with the statement, by checking of one of the following: strongly disagree, disagree, neutral, agree and strongly agree.

The questionnaire were analysed group for group, and each group were analysed question by question. The null hypothesis was tested for each group of questions. To reject it we required that the results for one sample were significantly better for at least 50 % of the questions than the results of the sample they compared to. We considered the results to be significant for p-value less than 0.1.

4.1 Hypothesis A – Learning

What is the effect on knowledge sharing and understanding when the subjects in the treatment groups gain some experience in using the release planning method in small teams, and start to learn from previous experience? Are there any differences between the treatment groups using the proposed release planning method and the control groups solving the task in an ad hoc style. To answer these questions we analyse the differences between the results from experiments E1–E2.1, E1–E2.2 and E3.1–E3.2.

Knowledge sharing – Six questions

- **E1 – E2.1** The results for the treatment groups in E2.1 are better for 6 out of 6 questions compared with the results in E1. For one question, the difference is significant. For the control groups the differences are really small and sometimes even negative – i.e. the results in E2.1 are worse than in E1.
- **E1 – E2.2** The results for the treatment groups in E2.2 are better in 4 out of 6 questions compared with the results in E1. For two questions, the difference is significant. For the control groups the results are worse on 5 out of 6 questions, and this difference is significant in two cases.
- **E3.1 – E3.2** The same observations are valid for the results in E3.1 and E3.2. The treatment groups show better results in 4 out of 6 questions, while the control groups show better results only in 1 out of 6.

The results are not strong enough to reject the null hypothesis with the criteria we have defined for this experiment. A boxplot for the results of question 1 for the treatment groups from these experiments are shown in figure 3.

The results show that the treatment groups improved their performance on knowledge sharing as they gain experience in using the proposed method and start to learn from previous experiences. The control groups do not improve their performance on knowledge sharing. While the control group did perform better on knowledge sharing on E1, this has now changed, and the treatment group performs better in both E2.2 and E3.2 In our opinion, using the release planning method in this situation helps the group to organise and share their knowledge, and help them to learn from it.

We consider the results as a strong indication that the proposed release planning method is enabling and supporting knowledge sharing, and conclude that a little experience and learning is needed to take advantage of the release planning method to improve knowledge sharing over the level that exists when working ad hoc style.

Understanding – four questions

- **E1 – E2.1** The results for the treatment groups in E2.1 are better in 1 out of 4 questions compared with the results in E1, and in this case the difference is significant. For the control group the results in E2.1 are better in 2 out of 4 questions, but the difference is not significant.
- **E1 – E2.2** The results for the treatment groups in E2.2 are better in 1 out of 4 questions compared with the results in E1, and in this case the difference is significant. For the control group the results in E2.2 are worse in 4 out of 4 questions, and the difference is significant in 3 of them.
- **E3.1 – E3.2** The results for the treatment groups in E3.2 are worse in 4 out of 4 questions compared with the results in E3.1, and on 3 of them the difference is significant. For the control group the results in E2.2 are worse in 4 out of 4 questions, and the difference is significant in 4 of them.

The null hypothesis can not be rejected based on the results. The results show that understanding is not affected by experience and learning. The results are the same for both the treatment groups and the control groups. Our conclusions is that understanding is influenced by other factors that experience and learning. The results in E2.2 and E3.2 might be affected by the fact that 3 out of 4 questions where negatively rephrased (see also section 5) and it seems that participants tend to be more neutral in this case.

4.2 Hypothesis B

What is the impact of changing the communication pattern for the treatment group on knowledge sharing, understanding and communication? We compare the results from experiments E1–E3.1 and E2.2–E3.2. The subjects of experiment E2.2 are participating in their third experiment, whereas the subjects of experiment E3.2 are participating in their second experiment. Hence, the subjects of experiment E2.2 are more experienced than the subjects of E3.2. However, comparing the results from E3.2 with E2.1 introduces other problems with respect to the questionnaire and the scenario. This will be discussed in a section 5.

Knowledge sharing

- **E1 – E3.1** The results for the treatment groups in E3.1 are better in 5 out of 6 questions on knowledge sharing compared with the results of E1. For two questions, the difference is significant. For the control groups, there is almost no difference in the results of E3.1 and E1, and none of the difference found have significance.
- **E3.2 – E2.2** The results for the treatment groups in E3.2 is slightly better for 5 out of 6 questions. However, the difference is not significant for any question. The control group has similar results with 4 out of 6 questions with a slightly better result for E3.2 compared with E2.2. However, the treatment group has better results for E3.2 than the control groups.

None of the results are so strong that we can reject the null hypothesis. The results show that the treatment group is improving their performance on knowledge sharing at both occasions. We conclude from the results that the improvements stemming from a changed communication pattern decline with more experience on how to use a new method. In other words, to support the communication when introducing a new method in a context where communication is critical, it is important to promote communication. After the method has been internalised this is not necessary any more.

Understanding

- **E1 – E3.1** The results for the treatment groups in experiment E3.1 are better then for E1, with two of the differences being significant. The control groups have better results in E3.1 then in E1 for 3 out of 4 question, with one significant difference.
- **E3.2 – E2.2** The results for the treatment groups are better in 3 out of 4 questions (with one significant difference). The control groups have the same result.

The results show that the null hypothesis can be rejected for the treatment groups on E1 – E3.1. Understanding can be improved by instructing group members to communicate more when a new method is introduced. When the group members gain some experience in the methods, there seem to be other factors that influence understanding more than learning (see above) and a more explicit communication scheme.

5 Discussion

There are some issues about the experiment that have to be considered in more detail.

The first issue is the questionnaire that was used for experiments E2.2 and E3.2 (shown as questionnaire 2b in figure 2). To avoid that the respondents reuse their response from the previous experiment and try to be conform by giving the same answer, we decided to regroup the questions and change around 30 % of the questions into a negative form. The original statement ”The release chosen by my group was my preferred release” was changed into ”The release chosen by my group was not my preferred release”. To compare the results of these negative worded questions with the results of the corresponding positive worded questions, we had to change the value of these questions, assuming that the response ”Disagree strongly” on a negative question is the same result as ”Agree strongly” on a positive question. However, when we compare the results for questions with a negative worded variant in questionnaire 2b, we see that the differences between the questions are bigger than for other questions. It seems that it is more easy for respondents to say that they agree than to say that they do not agree. In cases where the respondents feel that they do not agree, it seems that they prefer to choose ”Neutral” or ”Disagree”, but not ”Strongly disagree”. This effect – unknown to us at the time we planned the experiment – has been described – among others – in [6].

This effect disturbs the comparison between experiments E1–E2.2 and E2.1–E2.2. We assume that without this effect, some of the differences would have been stronger, and maybe even significant. We have decided not to remove these questions, as they still show how the difference between the treatment and control groups changes over time. This difference has a positive tendency for both knowledge sharing and understanding for the treatment groups.

Another issue are the impact of the scenarios on the results. The scenarios had some variations, such as the total development effort that could be included into a configuration, the total number of requirements in a scenario, and the ”size” of the requirements, i. e. the estimate of development effort. These factors have an impact on how easy a group of stakeholders think it is to find a final release. In an interview with 10 students that participated in experiments E3.1 and E3.2, they told us that each decision had its own challenges, and that they found difficult aspects in both scenarios.

In the comparison with experiment E2.2–E3.2 the students of E2.2 are more experienced as this is their third experiment, while it is only the second experiment for the students of E3.2. The reason for comparing these two experiments is that it involves two experiments with the same scenario and questionnaire. In our opinion it is better to compare two experiments with the same scenario and questionnaire – and accept the difference in experience – than to compare two results that were collected with different questionnaires.

Our last issue is the validity threat posed by the use of students in the experiments. Students are not as experienced as professionals, and the experiment may therefore not be as realistic as it would be if professionals were used. However, other studies using both students and professional indicate that the relative difference in the results between students and professionals for two or more experiments are close [1] [3]. Also, it would be very resource demanding to acquire a sample of professionals of the same size as used in our experiment. The experiments reported in this paper could, however, be expanded with one or more case studies using the proposed release planning method in a more realistic context, i.e. with professionals.

6 Conclusions and further work

In this paper we have presented the results of four experiments that we conducted to further investigate a release planning method [10]. An initial experiment [9] gave us some answers about the use of the method, but also some new questions. In the experiments described in this paper, we addressed two issues: what is the effect of experience and learning on factors such as knowledge sharing and understanding, and how will a changed communication pattern impact these factors.

The results of the experiments show that knowledge sharing is positively influenced by both experience and changed communication pattern. Understanding is improved little by changing the communication pattern and not from experience. The results indicates that the communication pattern introduced in this experiment will accelerate the improvement in the performance on knowledge sharing.

References

[1] E. Arisholm and D. Sjøberg. Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software. *IEEE Transaction on Software Engineering*, 30(8):512–534, 2004.

[2] P. Carlshamre. Release planning in market-driven software product development: Provoking an understadning. *Requirement Engineering*, 7(3):139–151, 2002.

[3] J. Carver, L. Jaccheri, S. Morasca, and F. Shull. Issues in using students in empirical studies in software engineering education. In *Proceedings of the Ninth International Software Metrics Symposium (METRICS'03)*, 2003.

[4] G. Du, J. McElroy, and G. Ruhe. A family on empirical studies to compare informal and optimization-based planning of software releases. In *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering*, pages 212–221, 2006.

[5] J. Momoh and G. Ruhe. Release planning process improvement – an industrial case study. *Software Process: Improvement and Practice*, 11(3):295–307, 2006.

[6] H. W. O'Neil. Response style influence in public opinion surveys. *The Public Opinion Quarterly*, 31(1):95–102, 1967.

[7] B. Ramesh, J. Pries-Heje, and R. Baskerville. Internet software engineering: A different class of processes. *Annals of Software Engineering*, 14:196–195, 2002.

[8] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers, 2000.

[9] S. Ziemer and I. C. Calori. An experiment with a release planning method for web application development. In *Submitted to Eurospi 2007*, 2007.

[10] S. Ziemer, P. Sampaio, and T. Stålhane. A decision modelling approach for analysing requirements configuration trade-offs in time-constrained web application development. In *Proceedings of SEKE 2006*, 2006.

[11] S. Ziemer and T. Stålhane. Web application development and quality - observations from interviews with companies in norway. In *Proceedings of Webist 2006*, 2006.

Distributed BPEL Processes

Luciano Baresi*, Andrea Maurino+, and Stefano Modafferi*
*Dipartimento di Elettronica e Informazione - Politecnico Milano
Piazza L. da Vinci 32 – 20133 Milano (Italy) baresi|modafferi@elet.polimi.it
+Dipartimento di Informatica Sistemistica e Comunicazione - Università di Milano Bicocca
Via Bicocca degli Arcimboldi 8 – 20126 Milano (Italy) maurino@disco.unimib.it

Abstract

BPEL only supports a strictly centralized and coordinated execution of Web service compositions, but this solution is clearly not the best option in many concrete cases. The conceptually monolithic BPEL process should be executed in a distributed setting, where each peer is only responsible for a fraction of the whole process and for the coordination with the other fragments. These considerations lead us to propose a formal approach to support the distributed execution of BPEL processes. The partitioning is customizable and easily intertwines with the adoption of special-purpose middleware infrastructures. In particular, the paper shows how we pair the distributed execution with a tuple-based infrastructure, to support distributed execution in mobile contexts, and with a publish and subscribe middleware, to support dynamically changing scenarios.

1 Introduction

BPEL (Business Process Execution Language [12]) is the de-fact standard for workflow-based compositions of Web services, but their centralized approach to composition and coordination is often a too heavy constraint. For example, if we think of business processes that involve different branches of the same organization, or also different companies federated into virtual organizations, we tend to conceive the global process as a single entity to identify the interdependencies among the parts and the overall synchronization among them. However, its centralized execution is only a technological choice: the parts that pertain to the different partners may be executed in a distributed setting, where each peer is responsible for a particular part of the process and coordinates with the others to keep the overall structure of the execution. This is an example of the more general case of cooperations where the different actors can proceed autonomously and only need loose synchronization with the other parts.

We can also consider all those cases where the computing infrastructure is based on a set of devices with limited capabilities, which can execute part of the process, but are not powerful enough to take care of the whole execution. Emergency situations [8] are good examples for this scenario: oftentimes a reliable computing infrastructure does not exist —or has been damaged— and it must be mimicked through mobile ad-hoc networks set by using portable devices with limited capabilities. Similarly, nomadic users may be in charge of fragments of more complex processes that need to exploit temporary connections to the network to exchange information and synchronize with the other parts of the process.

All these examples highlight some clear requirements for the execution of BPEL processes. A conceptual monolith BPEL process must be executable in a distributed setting, but the execution must preserve the behavior of the centralized one. BPEL engines are still good candidates for executing the different parts, but we need to move from one single BPEL engine to a set of federated engines. The different needs require that the distributed execution be based on suitable middleware infrastructures that offer the basic communication primitives to support the interaction and synchronization of the different parts. Different scenarios require that the middleware provide different guarantees: frequently disconnected partners may ask for persistency of synchronization messages to support the asynchronous communication with the others, while highly flexible distributions may require multi/broadcast mechanisms to be able to exchange messages with a dynamically defined number of partners.

These considerations lead us to propose a formal approach to support the *distributed execution* of BPEL processes. The approach, whose first formulation was presented in [2], exploits metamodeling techniques and graph transformation systems [1] to provide a formal means to partitioning a BPEL process into a set of coordinated subprocesses. The BPEL activities required to synchronize the control and data flows with the other subprocesses are added automatically. The transformation and slicing process is customizable and thus can exploit different middleware infrastructures. For example, if the communication framework supports the multicast delivery of information, the BPEL activities responsible for sending data to the other processes can avoid to consider the actual number of partners. We also present the explicit support for two different well-known middelware technologies: *LIME* [9], as representative of tuple-based systems for mobile devices, and *ReDS* [6], to address publish and subscribe infrastructures.

The rest of the paper is organized as follows. Section 2 exemplifies the approach for automatically partitioning BPEL processes on a simple step-by-step transformation. Section 3 introduces the tools developed both to support the automatic partitioning and to let obtained subprocesses interact with the selected communication infrastructure. Section 4 surveys some related proposals and Section 5 concludes the paper.

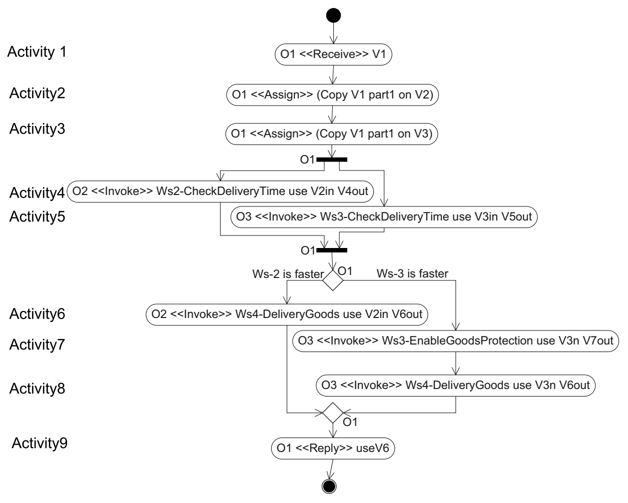
2 Workflow partitioning

The approach for partitioning BPEL processes is based on graph transformation [1]. The choice of representing *partitioning rules* as productions of a graph transformation system relies on the fact that the interpretation of workflows as graphs is widely diffused and thus workflow partitioning can be easily interpreted as a transformation of such graphs. In this paper, we do not explain how *partitioning rules* are defined[1], but we prefer to explain how they work.

A simple example process allows a broker to deliver goods to its clients. After receiving a request (from a client), the process checks two alternatives —two different Web services— to estimate the time needed to deliver ordered goods and always chooses the fastest way. This means that in one case, the process simply invokes an external shipping service, while in the other case it uses another Web service to enable a specific protection on ordered goods and then invokes the same service as for the first case. In the end, it replies the client with the estimated delivery time.

Besides showing the process, Figure 1 also introduces a possible division among three orchestrators. In this scenario, the different orchestrators can be seen as different branches of the same company: `O1` is the broker, `O2` is in charge of the simple delivery process, and `O3` manages the "protected" delivery process. External services and variables have simple names (`WSn` and `Vn`, where `n` is an integer) and operations have self-explaining names. Notice that each variable is used by more than one operation and therefore we need to consider them in the partitioning process since a variable is used by operations invoked by different orchestrators.

[1]Interested readers can refer to [2] for a complete presentation.



Activity 1
O1 <<Receive>> V1
Activity2
O1 <<Assign>> (Copy V1 part1 on V2)
Activity3
O1 <<Assign>> (Copy V1 part1 on V3)
O1
Activity4
O2 <<Invoke>> Ws2-CheckDeliveryTime use V2in V4out
Activity5
O3 <<Invoke>> Ws3-CheckDeliveryTime use V3in V5out
O1
Ws-2 is faster
O1
Ws-3 is faster
Activity6
O2 <<Invoke>> Ws4-DeliveryGoods use V2in V6out
Activity7
O3 <<Invoke>> Ws3-EnableGoodsProtection use V3n V7out
Activity8
O3 <<Invoke>> Ws4-DeliveryGoods use V3n V6out
O1
Activity9
O1 <<Reply>> useV6

Figure 1. Original process.

The result of the step by step partitioning of Figure 1 is presented in Figure 2. It only shows the final outcome, intermediate steps are omitted, but the reader can easily follow the entire process by means of the labels added to the figures.

Activity 1 of Figure 1 is controlled by `O1` and it is not touched by the partitioning process. Activity 2 is controlled by `O1` and produces `V2` that is used by Activity 4, which in turn is controlled by `O2`. Since this implies that `V2` must be forwarded to `O2`, we insert a pair <`invoke` *sendData*, `receive` *receiveData*> in the flow and mark it with **a** in Figure 2. The pair is used to pass `V2` to `O2`. The local process on `O2` is instantiated by activity `receive` *receiveData* This is normal behavior because it is nothing but a BPEL `receive` and as such it can instantiate new processes. The two previous points must be repeated for Activity 3 (marked with **b** in Figure 2).

We use two parallel threads (BPEL `flow` activity) to let `O2` call operation *checkDeliveryTime* on service `WS2` (Activity 4 in Figure 1) while `O3` calls the same operation on `WS3` (Activity 5). The control flow must be passed by `O1`, which controls the structured node that starts the flow, to `O2` and `O3`. This is achieved by means of two pairs <`invoke` *giveControl*, `receive` *receiveControl*> (**c** and **d** in Figure 2). When `O2` and `O3` have both data and control flow, they can perform Activity 4 and Activity 5, autonomously. Then `O2` and `O3` give back the control to `O1` that is in charge of controlling the end of the structured activity (**g** and **h** in Figure 2).

To start the `switch`, `O1` evaluates the associated condition and propagates the result to all the orchestrators involved in the switch. The evaluation is a local operation implemented by using an `assign`, while the propagation is performed in parallel by means of two pairs <`invoke`

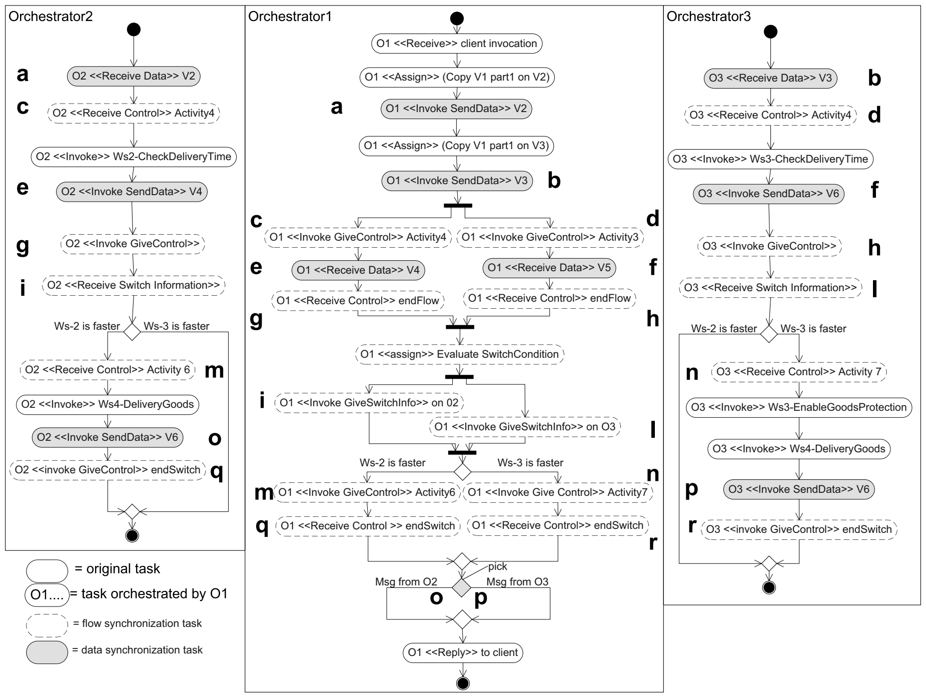

Orchestrator2
Orchestrator1
Orchestrator3
O1 <<Receive>> client invocation
O1 <<Assign>> (Copy V1 part1 on V2)
O1 <<Invoke SendData>> V2
O1 <<Assign>> (Copy V1 part1 on V3)
O1 <<Invoke SendData>> V3
O1 <<Invoke GiveControl>> Activity4
O1 <<Invoke GiveControl>> Activity3
O1 <<Receive Data>> V4
O1 <<Receive Data>> V5
O1 <<Receive Control>> endFlow
O1 <<assign>> Evaluate SwitchCondition
O1 <<Invoke GiveSwitchInfo>> on 02
O1 <<Invoke GiveSwitchInfo>> on O3
Ws-2 is faster
Ws-3 is faster
O1 <<Invoke GiveControl>> Activity6
O1 <<Invoke Give Control>> Activity7
O1 <<Receive Control >> endSwitch
O1 <<Receive Control>> endSwitch
pick
Msg from O2
Msg from O3
O1 <<Reply>> to client
O2 <<Receive Data>> V2
O2 <<Receive Control>> Activity4
O2 <<Invoke>> Ws2-CheckDeliveryTime
O2 <<Invoke SendData>> V4
O2 <<Invoke GiveControl>>
O2 <<Receive Switch Information>>
O2 <<Receive Control>> Activity 6
O2 <<Invoke>> Ws4-DeliveryGoods
O2 <<Invoke SendData>> V6
O2 <<invoke GiveControl>> endSwitch
O3 <<Receive Data>> V3
O3 <<Receive Control>> Activity4
O3 <<Invoke>> Ws3-CheckDeliveryTime
O3 <<Invoke SendData>> V6
O3 <<Invoke GiveControl>>
O3 <<Receive Switch Information>>
O3 <<Receive Control>> Activity 7
O3 <<Invoke>> Ws3-EnableGoodsProtection
O3 <<Invoke>> Ws4-DeliveryGoods
O3 <<Invoke SendData>> V6
O3 <<invoke GiveControl>> endSwitch
a b c d e f g h i l m n o p q r
= original task
O1.... = task orchestrated by O1
= flow synchronization task
= data synchronization task

Figure 2. Partitioned processes.

giveControl, `receive` *receiveControl*> (**i** and **l** in Figure 2). This is not data flow propagation since the data flow graph shows that only the structured node used to start the branch can use variables `V4` and `V5`. The duplication of the `switch` and the need for propagating the choice are imposed by the control flow.

Now each orchestrator knows the right path that must be followed. The symmetry of proposed process allows us to only analyze `O2`. If it executes Activity 6, it gets the control from `O1` by the pair <`invoke` *giveControl*, `receive` *receiveControl*> marked with **m** in Figure 2. Then it performs Activity 6. If we skip for a while the problem of sending `V6` (the output of Activity 6) to `O1`, which is the next orchestrator that uses the variable, we can finally give the control back to `O1` (in charge of controlling the end of the `switch`). If the selected branch were the other, `O2` would do nothing.

Notice that when an orchestrator in a branch of a `switch` has no activities to execute, it is extremely important that it does nothing. This way, it allows the flow to go on. If we simply stopped it, we would not keep the original behavior. Moreover, variable `V6` is used as output for both Activity 6, controlled by `O2`, and Activity 8, controlled by `O3`. This is a typical situation where the simple control flow does not work. In fact, Activity 9 uses variable `v6` that is the output either of Activity 6 or Activity 8, therefore the data source of Activity 9 is not identifiable at design time. The proposed solution is only a special-purpose way to bypass the problem.

Activity 9, which is the last one, seems to be simple, but we need to discuss how to propagate data. Again, at design time, we cannot determine the actual source for `v6`. We can modify the process to avoid this problem, but it is interesting to analyze how we can solve this case. BPEL provides a mechanism called `pick`: It can be seen as a set of `receive` operations. Given the incoming message, the system chooses the `receive` that must be activated and thus the execution follows the corresponding branch. To solve our problem, we use two <`invoke` *sendData*> on the two possible sources, and a `pick` on `O1` to model the two possible <`receive` *receiveData*> for moving the decision about the actual source at runtime (**o** and **p** in Figure 2). Given this solution, the `reply` (Activity 9) is only a simple local activity.

3 Supporting tools

The conceptual approach described in the previous section is supported by prototype tools that cover the whole life cycle. Our tools support the *design* of partitioning rules that can be either generic or be tailored to particular middleware infrastructures, the *deployment* of generated BPEL (sub-) processes, and the *execution* of these processes by supporting the synchronization and message exchange among them.

Design tool The **Breaker**, our design tool, was not conceived from scratch, but it is based on AGG (Attributed Graph Grammars) [4], which supplies suitable graphical interfaces, to design new transformation systems, a Java-like syntax, to specify the type graph and attribute values, and analysis engines, to validate designed systems. The **Breaker** can be used both through a graphical user interface and as a dedicated Web service. Figure 3 shows its main components.

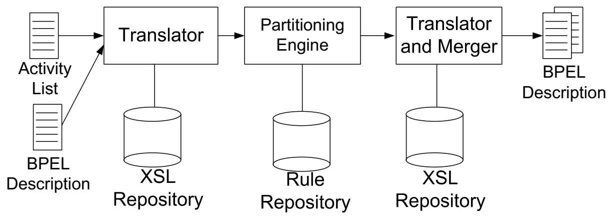


Figure 3. Main components of the Breaker.

The **Translator** is in charge of transforming a BPEL process into an attributed graph. The module receives a BPEL process, the information about the orchestrators, and the list of activities that each orchestrator is supposed to manage. Its output is a GXL (Graphical eXchange Language) file that embeds the attributed graph. The translation from BPEL to GXL uses XSL technology since the BPEL process, the lists of activities, and GXL are all expressed in XML. The **XSL repository** stores a set of XSL stylesheets, which are specific to the different commercial BPEL engines, that address the different BPEL "dialects" adopted by selected engines.

The **Partitioning Engine** is the core of the tool and, as already said, is based on AGG. The engine transforms an attribute graph into another by means of the set of partitioning rules stored in the **Rule Repository**. The engine receives a GXL file, which represents the original BPEL process, and produces a set of GXL files that represent the local processes for the different orchestrators.

The **Translator and Merger** has two goals: It translates each GXL file into a BPEL description, and merges all the BPEL files into a unique XML file that is the final result of the **Breaker**. A second **XSL Repository** stores the stylesheets able to translate a GXL description into a BPEL process.

Runtime support When we move to the runtime support we offer for the distributed execution of BPEL processes, we must recall that the rules presented in Section 2 must be supported by suitable synchronization mechanisms. If we think of the conventional `receive/reply` mechanism (a typical RPC-based communication), when an orchestrator wants to synchronize the execution flow or communicate some data to another orchestrator, the two orchestrators must be connected at the same time. This option is a solution when the whole set of engines is deployed on a wired network. In many other cases, the synchronization may be more complex and we need suitable middleware frameworks to support the cooperation among BPEL engines.

So far we have developed two different solutions (partitioning rules and related functionality) to support the runtime management of partitioned processes. The first solution, called *DOME* (Distributed Orchestration in the Mobile Environments), deals with distributed BPEL processes deployed onto mobile ad hoc networks (MANETs). The synchronization among engines might be complex due to the unpredictable and temporary disconnections of some devices. As for many distributed system, we adopted a specific middleware solution to clearly separate the application layer (BPEL engines) and the communication infrastructure. *DOME* extends the `invoke/receive` mechanism without imposing any modification to existing BPEL engines. We simply provide special-purpose implementations of the two operations.

At *application* level, BPEL engines communicate directly by means of the synchronization activities introduced by the rules; thus when a process has to communicate with another process, it simply executes an `invoke`. At *infrastructure* level, the invocation is translated into the invocation of a special-purpose Web service hosted on the same device as the calling process and the real communication is implemented by exploiting *LIME* (Linda in Mobile Environment, [9]), which is a tuple-based middleware infrastructure designed to support the development of mobile applications over both wired and ad hoc networks. In *LIME*, mobile agents (e.g., Web Services) reside on mobile hosts and the communication among them takes place via transiently shared tuple spaces distributed across the mobile hosts. The tuple space mechanism provided by *LIME* allows *DOME* to manage the communication among orchestrators in a completely transparent way with respect to the application level. At the infrastrcture level, an `invoke` corresponds to an `out` operation on the local tuple space. *LIME* is in charge of moving added data to the target host, where it performs an `in` operation towards the instance of the BPEL process specified by the sender.

The second solution we present in this paper is called *DOPS* (Distributed Orchestration based on Publish and Subscribe). It is based on the publish and subscribe interaction paradigm to support highly flexible and dynamic scenarios. In publish and subscribe systems, application components *subscribe* to message (event) patterns and get *notified* when other components *publish* messages matching their subscriptions. A dedicated component called *dispatcher* is in charge of handling the delivery of the different messages and thus decouples the communication between senders and receivers. Its asynchronous, implicit, and multi-point communication style is particularly amenable to those applications in which components adopt the multicast delivery of information.

Following this approach, the engines do not need to know the receivers of their messages: They publish their messages and interested orchestrators receive them because previously subscribed. *DOPS* is implemented on top of *ReDS* [6], which supports user-defined message formats and filters (used to allow users to subscribe to the different messages). *ReDS* also provides a distributed dispatcher organized as a set of brokers linked onto an overlay dispatching network.

Before starting the execution, we need a *DOPS subscriber* for each orchestrator. This component is responsible for analyzing the local process, detecting all the messages that the orchestrator is supposed to exchange with the other peers, and subscribing for all the message types relevant for the cooperation with other orchestrators. Currently, *DOPS* defines two special-purpose filters to represent messages that correspond to variables and execution flows, and allows each engine to select the appropriate filter according to the particular needs.

The pair `invoke/receive` is obtained by means of two special-purpose Web services that are in charge of publishing produced messages onto the *ReDS* infrastructure and conversely retrieve those for which the processes are subscribed. More precisely, an `invoke` activity is realized by a Web service hosted on the same machine as the engine that publishes the variables or execution flows. A `receive` activity is hosted on the same machine as the target engine and is realized by a Web service that receives the messages and redirects them to the appropriate process instance. *DOPS* and *DOME* adopt instance ids, suitably encoded in exchanged messages, to disambiguate instances.

These two supporting frameworks introduce the problem of selecting the "right" middleware infrastructure for the distributed execution we want to support. A thorough discussion of the possible criteria is out of the scope of this paper. Here, we can only touch the problem by means of an example. Some process descriptions cannot be partitioned due to data liveness; however in some situations a publish and subscribe solution (or a solution based on a tuple space) can overcome the problem of identifying the last user of a variable passed among different engines. For example, in a loan process, the system can update variable *mortgage* by means of dedicated Web services that use different risk profiles, or it can issue an error message for the user if the risk profile is undefined. This situation can easily be modeled with a `switch` activity, where in all branches but one, there is an activity that modifies the same variable. In this case, partitioned processes can only be executed on top of *DOPS* due to the decoupling between publishers and subscribers. The first (and unique) activity that executes must publish the message that contains the new value of *mortgage*, which is then dispatched to the receiver engine. Statically, we do not know the engine that publishes the message, but we know those that are interested in receiving it. We only know that one engine at a time can produce the value and thus we can be sure that published value is always the last value of the shared variable.

4 Related work

The distribution of workflow management has been widely studied. Lack of space does not allow us to present a thorough analysis of the different approaches, and thus we need to limit our attention to the most relevant ones. It is possible to establish a parallelism between research in data integration [3] and research in distributed workflow management. In the first area, there are two possible approaches to integrate schemas of different sources: (i) the *Global As View* (GAV) approach, in which a set of different data schemas are merged into a unique schema, and the *Local As View* (LAV) approach, in which schemas of information sources are derived from a global schema. By analyzing the literature on distributed workflow management, it is possible to identify two similar approaches: the first one supports the integration of autonomous and preexisting workflows and aims mainly at the coordination of the different and independent actors. The second approach supports the decomposition of single workflows to support their autonomous execution by means of different engines. According to this taxonomy, our approach is LAV-based because the global schema is the original process description we want to slice, and the local processes are the "views" of the global description.

ADEPT [11] belongs to GAV-based approaches. It supports the modeling, analysis, and verification of workflow templates to guarantee correctness properties. The main feature of ADEPT, and its evolution ADEPTflex [10], is the possibility of modifying workflow instances at runtime: ADEPT allows us to reduce the network workload by partitioning workflow graphs and migrating the control of workflow instances from one server to another at runtime.

The next two approaches are similar to ours and they both are LAV-based. The *BPEL-based decentralized framework* presented in [5] proposes a decomposition approach to increase the parallelism for improving performance rather than fostering the integration of multi-organizational workflows. The authors use BPEL as *composite Web service* model. They assume a centralized design of the workflow and their system distributes the orchestration of the different portions among different nodes. In this distributed scenario, there is no centralized coordinator that can be a potential bottleneck. Data distribution reduces network traffic and improves transfer time. Control distribution improves concurrency, and the asynchronous messaging among engines brings the benefits of better throughput and graceful degradation. The decentralization process essentially consists of three steps: automatic parallelization and code partitioning, synchronization analysis, and code generation. They are based on data flow analysis techniques, to determine the maximum parallelism, and on a cost function, to determine the most efficient code partition. After partitioning the code, they analyze the interactions among partitions to determine the best synchronization protocols.

The *E-Role based decomposition* system [7] addresses role-based decomposition of a business process model (based on a BPEL subset). They present a mechanism for partitioning a business process where each partition can be enacted by a different participant. An important goal is to disconnect the partitioning itself from the design of the business process, thus simplifying the reassignment of activities to different entities. The result is a set of BPEL-compliant processes, one for each participant, as well as the information needed to wire them together at deployment time and ensure correct instance-level connections at runtime.

5 Conclusions and future work

The paper presents a novel and formal approach to support the distributed execution of BPEL processes. It proposes a formal methodology to automatically partition a monolithic process into a set of coordinated subprocess by means of graph transformation systems. The approach also intertwines with special-purpose middleware infrastructures to better address the communication and synchronization requirements of the different distributed scenarios. So far, the approach integrates with *LIME* and *ReDS* to support tuple-based and publish and subscribe communication patterns. We are quite confident that the approach can be easily generalized —by means of dedicated partitioning rules and communication functionality— to other middleware infrastructures. In many cases, the selection of the right middleware infrastructure, and thus the correct set of partitioning rules, depends on many different factors. The distribution helps bypass the bottleneck of a single centralized executor, but introduces the usual problems that characterize distributed systems.

The first results are encouraging, but we still need to assess the viability of the approach by means of further examples and scenarios and thus refine our supporting tools accordingly.

References

[1] L. Baresi and R. Heckel. Tutorial Introduction to Graph Transformation: A Software Engineering Perspective. In *Proceedings of the First International Conference on Graph Transformation (ICGT 2002)*, volume 2505 of *Lecture Notes in Computer Science*, pages 402–429. Springer-Verlag, 2002.

[2] L. Baresi, A. Maurino, and S. Modafferi. Workflow Partitioning in Mobile Information Systems. *In Proc. of IFIP TC8 Working Conference on Mobile Information Systems*, volume 158 of *IFIP International Federation for Information Processing*, 2004.

[3] C. Batini and M. Scannapieco. *Data Quality*. Springer Verlag, 2006.

[4] M. Beyer. *AGG1.0 - Tutorial*. Technical University of Berlin, Department of Computer Science, 1992.

[5] G. Chafle, S. Chandra, V. Mann, and M. Nanda. Decentralized Orchestration of Composite Web Services. In *Proc. of the Int. World Wide Web conference on Alternate track papers & posters*, pages 134–143, New York, NY, USA, 2004. ACM Press.

[6] G. Cugola and G. Picco. Reds: A Reconfigurable Dispatching System. Technical report, 2005. Politecnico di Milano.

[7] R. Khalaf and F. Leymann. E Role-based Decomposition of Business Processes using BPEL. *icws*, 0:770–780, 2006.

[8] A. Maurino and S. Modafferi. Partitioning Rules for Orchestrating Mobile Information Systems. *Personal and Ubiquitous Computing*, 9(5):291–300, 2005.

[9] G. P. Picco, A. L. Murphy, and G.-C. Roman. Developing Mobile Computing Applications with Lime. In *ICSE*, pages 766–769, 2000.

[10] M. Reichert and P. Dadam. ADEPTflex – supporting Dynamic Changes of Workflows without Losing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

[11] M. Reichert, S. Rinderle, and P. Dadam. ADEPT Workflow Management System:. In W. van der Aalst, A. ter Hofstede, and M. Weske, editors, *Proc. of Int. Conference Business Process Management*, pages 370–379, Eindhoven, The Netherlands, 2003.

[12] S. Thatte. Business Process Execution Language for Web Services, 2003. `http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/`.

SAM: Semantic Advanced Matchmaker

E.S. Ilhan, G.B. Akkus, and A. B. Bener, *Member, IEEE*

***Abstract*— As the number of available web services increase finding the appropriate web services to fulfill a given request becomes an important task. In many cases a single service is not capable of solving large problems. Therefore, the problem of combining multiple web services to satisfy a single task, known as web services composition problem, has recently received a lot of attention by researchers and practitioners. An effective discovery and composition are only possible when there is semantic information. Most of the current solutions and approaches in web service discovery and composition are limited in the sense that they are strictly defined, and they do not use the full power of semantic and ontological representation. In this paper we focus on one of the most challenging tasks in service discovery and composition: matchmaking process. We use an efficient matchmaking algorithm based on bi-partite graphs. Our proposed algorithm uses attribute ranking through weight assignment. We have seen that bi-partite matchmaking has advantages over other approaches in the literature for parameter pairing problem.**

***Index Terms*—Matchmaking, semantic similarity, scoring, ranking, OWL-S, bi-partite graph.**

I. INTRODUCTION

In recent years, web services became the dominant technology in providing the interoperability among different systems throughout the web. If web service is used in limited business domain or with strict rules with known business partners everything will be fine. The problem of finding the right and most suitable web services for user needs emerges when open e-commerce systems are widely used and user requirements dynamically change over time.

Although there are currently proposed technologies for discovery of web services, such as UDDI [5], they do not satisfy the full discovery requirements. This discovery process is based on syntactical matching and keyword search that does not allow the automatic processing of web services. To solve the problem of automatic discovery and processing of web services, the Semantic Web [6] vision is proposed. The Semantic Web is an effort by the W3C consortium [7], and one of its main purposes is to facilitate the discovery of web resources.

There are different efforts and frameworks for semantic annotation and discovery of web services [10, 11, 12]. For web service discovery they also propose some techniques and algorithms. However, they mostly classify the discovered web services in set-based. They do not focus on rating the web services using semantic distance information [13].

The evolution of web services, from conventional services to semantic services, caused service descriptions contain extra information about functional or non-functional properties of web services. The semantic information included in the service descriptions enables the development of advanced matchmaking schemes, capable of assigning degrees of match to the discovered services. Semantic discovery of Web Services means semantic reasoning over a knowledge base where a goal describes the required web service capability as input. Semantic discovery adds accuracy to the search results in comparison to traditional Web service discovery techniques, which are based on syntactical searches over keywords contained in the web service descriptions [3].

Improvement in matching process could be gained by the use of ontological information in a useful form. With the use of this information, it can be possible to rate the services found in discovery process. As in real life, users/ agents should be able to define how they see the relation of ontological concepts from their own perspective. Similarity measures have been widely used in information systems [14, 15, 16], cognitive science, software engineering and AI [17, 18, 19]. So integration of knowledge from these techniques can improve the matching process.

By using semantic distance definition information, we aim to get a rated and ordered set of web services as the general result of the discovery process. We believe that this would be better than set-based classification of discovered services. In this paper, we propose a new scheme of matchmaking that aims to improve retrieval effectiveness of semantic matchmaking process. Our main argument is that conventional evaluation schemes do not fully capture the added value of service semantics and they do not consider the assigned degrees of match, which are supported by the majority of discovery engines. The existing approach to service matchmaking contains subsumption values regarding the concept that the service supports. In our proposed approach, we add semantic relatedness values onto existing subsumption based procedures.

II. RELATED WORK

Semantic Web services aim to realize the vision of the Semantic Web, i.e. turning the Internet from an information repository for human consumption into a world-wide system for distributed Web computing [6]. The system is a machine-understandable media where all the data is combined with semantic metadata. The domain level formalizations of concepts form up the main element within this system, which is called ontology [11]. Ontology represents concepts and relations between the concepts; these can be hierarchical relations, whole-part relations, or any other meaningful type of linkage between the concepts [4].

The semantic matchmaking process is based on ontology formalizations over domains. In the upcoming section we present some of the selective research on the matchmaking process considering the concepts that we build our research on. Matchmaking of Web services considers the relationship between two services. The first one is called the advertisement and the other is called the request [2]. Advertisement denotes the services description of the existing services while the request indicates the picture of service requirements [19].

In [1], the problem of capability matchmaking is analyzed with regarding to Web services, especially the Preconditions and Effects (PE) matchmaking. In the paper, the authors present a service similarity function which determines similar parameter classes by using a matching process over synonym sets, semantic neighborhood, and distinguishing features. Parameter pairing is the process that is used for matching service descriptions. In the work, maximum weight bi-partite graph matching method is utilized for parameter finding; the weights of bi-partite graph's edges are evaluated with matching degree between function parameters calculated by the similarity function mentioned above.

Although good results are obtained with the usage of this method, it should still be improved in two terms: One is that, it is needed to be extended on pre-condition and effect because the matching is performed only on parameters of input and output, and the functional signature is not sufficient to identify what it does. The other is that this framework should be combined with particular directory service like UDDI in order to improve the discovery efficiency.

In [21] the authors present an algorithm that deals with the localization of Web services. The research does not address the interoperability problem. The system introduced uses the service profile ontology from the DAML-S specification language but only considers the matching of input and output concepts defined by the same ontology.

Traditional approaches to modeling semantic similarity between Web Services compute subsume relationship for function parameters in service profiles within a single ontology. In [20] a graph theoretic framework based on bi-partite graph matching for finding the best correspondences among function parameters belonging to advertisement and request is introduced. On computing semantic similarity between a pair of function parameters, a similarity function is introduced, determining similar entity which relaxes the requirement of a single ontology and accounts for the different ontology specifications. The function presented for semantic similarity across different ontologies provides an approach to detect similar parameters. The method is based on a matching process over weighted sum of synonym sets, semantic neighborhood, and distinguishing features. The method mainly lacks use of functional similarities and lexical evaluation of semantic mappings.

In [22], a semantic ranking MSC is designed to rank the results of advertisements matchmaking. MSC stands for the initials of three factors' second words: Semantic Matching Degree (to capture the semantic aspects of attributes), Semantic Support (to describe the interestingness or potential usefulness of an attribute) and Relational Confidence (to capture the association relationships among attributes). Three categories of attributes are defined in advertisements matchmaking: *Generalizable Nominal Attribute (GNA)* whose values can form a concept hierarchy; *Numeric Attribute (NUA)*, called quantitative attribute, whose values are numeral; Nominal Attribute (NOA) whose values are neither numeral nor can form a concept hierarchy. Three new factors are designed to capture the semantic characteristics and relationships of the attributes: Semantic Matching Degree, Semantic Support and Relational Confidence.

Another approach on semantic matchmaking uses Case Based Reasoning (CBR). According to CBR theory the first step is to define all the elements contained in a case and the associated vocabulary that represents the knowledge associated with the context of a specific domain.

In [23], frame structures are adopted for the case representation. However, in this research the problem that research in semantic-based matchmaking and composition has not been addressed sufficiently, rather it is described as the interoperation between independently developed reasoning engines. Without this interoperation, the reasoning engines remain imprisoned within their own framework, which is a drawback, especially that most engines usually specialize in servicing a particular domain, hence interoperation can facilitate inter-domain orchestration.

III. PROBLEM STATEMENT

The first step in service composition is identifying the domain of interest by means of taxonomy of subject categories. The discovery and selection of services that are suitable for a given request is obtained in two phases. Firstly, matchmaking approach is based on the ontological framework. It is applied on the set of available services in order to find services that match needs of the requestor from a functional point of view. Secondly, services are ranked and further refined. This is done by taking into account context information of the requestor. Then preconditions or post-conditions can be defined as mandatory or optional.

The problem that we are concerned is the first step of this scenario: given a request *r*, finding right web services for *r*. The main goal of this research is to gain better precision and recall values on matchmaking by considering user requests in web services discovery.

The previous work on semantic matchmaking focused on taking advantage of a single implementation based on some information retrieval theory. The experimental research so far has shown simple subsumption based matchmaking is not sufficient to capture semantic similarity.

In this research, we aim to provide an efficient and accurate matchmaking algorithm using scoring and ranking based on similarity distance information, extended subsumption and property level similarity assessment in a general semantic web service discovery framework

IV. PROPOSED SOLUTION

In this paper we propose a hybrid approach on semantic matchmaking. Our proposed solution uses decision modules that can be plugged in and out. We have implemented some of these modules to add semantic relatedness values onto existing subsumption based procedures. Our proposed matchmaker agent architecture mainly provides ranking and scoring based on concept similarity. The components of the proposed architecture are shown in Figure 1. Request service definition and the corresponding relevant services set, which are discovered through conventional discovery mechanisms, are presented as input to the system. The ontology and services we use are retrieved from "OWL-S Service Retrieval Test Collection version 2.1". The services in the collection are mostly extracted from public UDDI registries, providing 582 web services described in OWL-S from seven different domains. The OWL-S Test collection version 2.1 contains 29 queries, each of which associated with a set of 10 to 15 services [8]. We extended some ontologies in this test collection for our own purposes in order to better demonstrate the features of our proposed matchmaking agent. We believe that a formal test collection of OWL-S services is crucial for the evaluation of matchmaking agents.

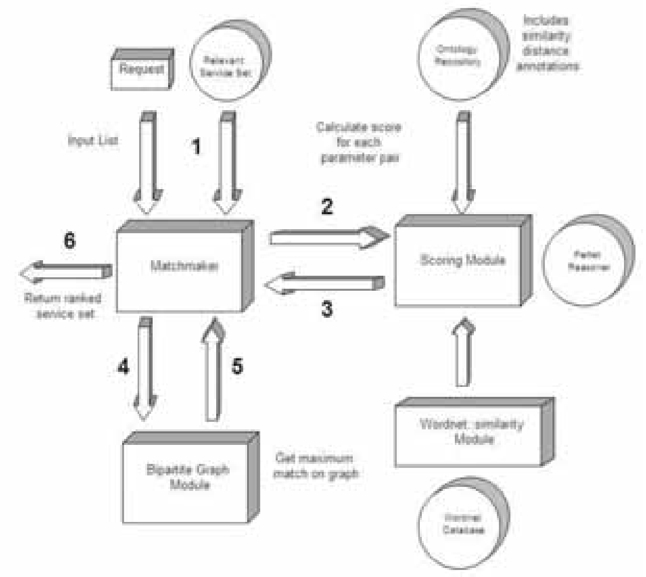

Request
Relevant Service Set
Ontology Repository
Includes similarity distance annotations
Input List
1
Calculate score for each parameter pair
2
6
Matchmaker
Scoring Module
Pellet Reasoner
Return ranked service set
3
4
5
Wordnet similarity Module
Bipartite Graph Module
Get maximum match on graph
Wordnet Database

Figure 1. Matchmaking agent components

The main software components of our proposed matchmaking agent are shown in Figure 2. The top layer represents our matchmaker SAM (Semantic Advanced Matchmaker). OWL-S API models the service, profile, process and grounding ontologies of OWL-S in an easy to use manner. It is a widely used API in semantic applications. OWL-S API also presents interfaces for reasoning operations and utilize Jena constructs at the back-end. At the bottom of the hierarchy we have Pellet reasoner for OWL reasoning operations.

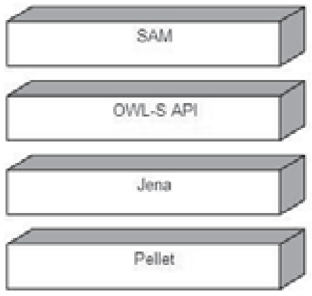


Figure 2. Software components of matchmaking agent

We believe that a discrete scale (exact, plug-in, subsume, and fail) of service classification is not sufficient for a matchmaking process. On the other hand, semantic ranking of services can capture a set of services that are lost in a discrete scale match. Semantic similarity assessment is a crucial step for the ranking process. In our proposed architecture, we present value-added similarity assessment approaches between service and request parameter pairs.

A. *Matching Algorithm*

Previous research has shown that bi-partite graph matching algorithm is a good fit for finding matching parameters in a service and request pair [9]. Bi-partite graph matching provides us a solution for parameter pairing problem. We consider the inputs and outputs as separate cases and partition the service parameters and request parameters to form the bi-partite graph. The similarity assessment process of our matchmaker assigns weights for each parameter pair on this bi-partite graph. A maximum weight match on the final graph leaves us with the optimum matching parameter pairs and with a score that is sum of the weights between matched parameter pairs. We repeat this process for each service and request pair and finally rank the services according to their score from bi-partite graph matching algorithm.

As we stated before the process that differentiates the services is the similarity assessment process. We consider OWL-S profiles of service definitions and assign similarity scores for input and output parameter pairs. We present the following value-added features for similarity assessment: Subsumption based similarity, WordNet based similarity, similarity distance information and WordNet similarity assessment.

Subsumption based Similarity Assessment

We make use of OWL-DL constructs *subClassOf, disjointWith, complementOf, unionOf and intersectionOf* to assess concept similarity based on subsumption. If two concepts are explicitly stated to be complement or disjoint, a zero score is directly assigned. Otherwise, we check for subclass relation and also assess according to property level assessment procedure described below.

We wanted to capture similarity values in bi-partite graph since it is important to decompose concepts that include the characteristic of "a union of". Following this approach, we always pair and assess score for atomic concepts in matchmaking process.

Property-level Similarity Assessment

We have assumed that in matchmaking it is also important to have properties and their associated range in measuring the degree of match. Such as, if two concepts have similar properties (properties having subclass relation) and their range classes are similar, then this improves their level of similarity.

A service, that would normally be eliminated by a conventional matchmaker, is ranked by using property level similarity assessment. For example, a user request may favor a particular author for a novel. A service, which returns articles that are written by that particular author will have a high score even though the concept of "an article" does not compare to the concept of "a novel". Therefore our proposed architecture returns positive results for concepts that have similar properties as well as the similar concepts.

Similarity Distance based Assessment

To represent similarity distance information we applied N-ary relation pattern in OWL, which is used to represent additional attributes on a property. The additional attribute in our case is the similarity distance value. Figure 3 shows how this pattern is organized:

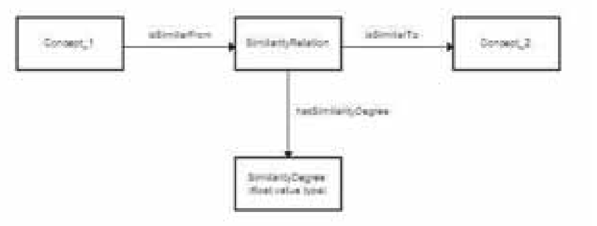


Figure 3. N-ary relation pattern in OWL, representing similarity distance information

SimilarityRelation concept is introduced as a class with this pattern and the similarity distance value is represented as the range of *hasSimilarityDegree* property of this concept. The similar classes are represented as Concept_1 and Concept_2 in Figure 3.

We follow the standards approach by representing similarity distance information in OWL, which can be imported and used in other ontologies [24]. Similarity distance information is useful in reflecting user's profile on the ontology. The importance and relatedness of concepts for the user are represented as weights on the ontology.

WordNet based Similarity Assessment

WordNet organizes words into synonym sets, which are also linked to each other representing a semantic relation. In our architecture we take WordNet as a secondary source of information with the ontology repository. We aimed at reasoning with these highly structured information sources in order to get more reliable result sets.

We make use of *wordnet::similarity* open source project to assess similarity score among words. The path length criterion is used for score assignment. The parameter types of services are presented as input to wordnet::similarity module.

V. Evaluation and Results

In order to evaluate the performance of our proposed matchmaking agent we extended the book ontology in OWL-S Service Retrieval Test Collection (OWL-S TC) and also modified related request and service definitions accordingly [8]. As shown in Figure 4, we added subclasses of *Magazine*, namely *Foreign-Magazine* and *Local-Magazine* classes.

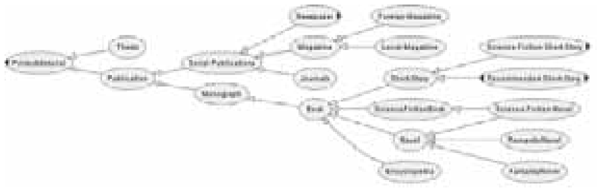


Figure 4. Printed Material ontology section

We created subclasses of *Publisher: Ordinary-Publisher, Alternative-Publisher and Premium-Publisher.* We also created *Local-Author and Foreign-Author* classes, which are subclasses of class *Author.*

The matchmaking agent is developed in Java and it makes use of open source semantic web libraries like OWL-S API and Jena. We also used Pellet as the reasoning engine for OWL operations. To represent subsumption reasoning, similarity distance based assessment and property-level similarity assessment capabilities we define the following request and services as described in Table 1:

	INPUTS	**OUTPUTS**
Request	Book	Author
Service 1	Author	Book, Price
Service 2	Novel	Author, Price
Service 3	Book	Author, Price
Service 4	Local-Magazine	Local-Author
Service 5	Book	Price
Service 6	Foreign-Magazine	Foreign-Author
Service 7	Science-Fiction-Book	Author
Service 8	Science-Fiction-Novel	Author

Table 1. Test request and service set

For the above test collection the property level similarity assessment plays an important role. Even though *Magazine* concept has no subclass relation with Book concept, both concepts have *hasAuthor* and *publishedBy* properties. Thus, our matchmaker applies a subsumption reasoning on ranges for these properties, which are Author with its subclasses, and *Publisher* with its subclasses. Finally, an additional score is provided for these services. These services would have been ignored as a "fail" by a conventional matchmaker.

We use the following weighted formulation for computing final score considering both semantic and WordNet similarity score:

Score = 0.8*Subsumption_Score + 0.2*WordNet_Score (1)

The final rank for the above collection from the most similar service to least is as follows:

Service3-Service2-Service7-Service8-Service4-Service6-Service5-Service1 (2)

To consider how semantic distance information effects our ranking we introduced the following weights into the book ontology as described in Table 2:

CONCEPT 1	**CONCEPT 2**	**SEMANTIC DISTANCE**
Publisher	Ordinary-Publisher	0.2
Publisher	Alternative-Publisher	0.5
Publisher	Premium-Publisher	0.3
Author	Local-Author	0.3
Author	Foreign-Author	0.7
Magazine	Foreign-Magazine	0.7
Magazine	Local-Magazine	0.3
Book	Short-Story	0.2
Book	Science-Fiction-Book	0.4
Book	Novel	0.3
Book	Encyclopedia	0.1
Novel	Science-Fiction-Novel	0.6
Novel	Fantasy-Novel	0.2
Novel	Romantic-Novel	0.2

Table 2. Semantic distance weights

The new ranking with semantic distance information is as follows:

Service3-Service7-Service8-Service2-Service6-Service4-Service5-Service1 (3)

As the semantic distance information favors *Foreign-Author* over *Local-Author* and *Premium-Publisher* over *Ordinary-Publisher*, we have Service 6 ranked higher than Service 4. Also Service 7 has a higher rank compared to Service 2, since genre *Science-Fiction* is favored over *Novel*.

The similarity score formulation considering semantic distance information is as follows:

1/(path_length+1)*semantic_distance1*semantic_distance2*...*semantic_distanceN , (4)

where path-length represents the edge count between the compared concepts in hierarchy and semantic_distanceN represents the weight on the Nth edge.

VI. CONCLUSION AND FUTURE WORK

We proposed a novel advanced matchmaker architecture, which introduces new value-added approaches like semantic distance based similarity assessment, property level assessment and WordNet similarity scoring. Instead of classifying candidate web services in a discrete scale, our matchmaking agent applies a scoring scheme to rank candidate web services according to their relevancy to the request.

The ranking property enables to include some of the relevant web services in the final result set whereas they would have been discarded in a discrete scale classification. Additionally, our proposed matchmaking agent improves subsumption based matchmaking by utilizing OWL constructs efficiently and by considering down to a level of concept properties in the process.

We also introduced semantic distance annotation in ontology to represent relevancy of concepts to the user in a numerical way. Semantic distance annotations improve the relevancy of returned web service set as they actually represent user's view of ontology. WordNet similarity measurement is also presented as a value-added feature, which acts as a secondary source of information, strengthening the power of reasoning.

The development phase of our matchmaking agent is still in progress and the results presented in this paper are preliminary. We also think that preconditions and results of a service should also be considered for a complete matchmaking process. At that point, use of SWRL (Semantic Web Rule Language) in both service advertisements and request description will enhance the capabilities of our matchmaking agent.

Another improvement will be to add context aware decision-making capabilities, enabling our matchmaking agent to reason based on user profiles, preferences, past actions etc. The architecture that we have presented can be considered as a basis for the development of context-aware agent.

REFERENCES

[1] Wang, H., Zengzhi L., Fan L., An Unabridged Method Concerning Capability Matchmaking of Web Services, In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, 2006.

[2] Klusch, M., Fries, B., Khalid, M., and Sycara, K.. OWLS-MX: Hybrid Semantic Web Service Retrieval. In *1st Intl. AAAI Fall Symposium on Agents and the Semantic Web*, AAAI Press, Arlington VA, 2005

[3] U. Keller, Lara R., Polleres A., WSMO Web Service Discovery, http://www.wsmo.org/2004/d5/d5.1/v0.1/20041112/

[4] H. El-Ghalayini, M. Odeh, R. McClatchey, and T. Solomonides, Reverse Engineering Ontology to Conceptual Data Models, In *Proceeding (454) Databases and Applications*, 2005.

[5] *Universal Discovery Description and Integration Protocol,* http://www.uddi. org, 2006.

[6] *Semantic Web, W3C*, http://www.w3.org/2001/sw/, 2006.

[7] *W3C, World Wide Web Consortium*, http://www.w3.org/, 2006.

[8] Mahboob Alam Khalid, Benedikt Fries, Patrick Kapahnke, OWL-S Service Retrieval Test Collection Version 2.1, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH Saarbrücken, Germany, 2006.

[9] Herbert Alexander Baier Saip, Claudio Leonardo Lucchesi, Matching Algorithms for Bi-partite Graphs, Relatorio Tecnico DCC-03/93.

[10] Motta, E., J. Domingue, L. Cabral and M. Gaspari, "IRS-II: A Framework and Infrastructure for Semantic Web Services", *Proceedings of the 2nd International Semantic Web Conference (ISWC2003)*, Vol. 2870 of LNAI., Springer, pp.306-318, Florida, USA, 2003.

[11] *OWL-S Submission*, http://www.w3.org/Submission/OWL-S, 2004.

[12] Fensel, D. and C. Bussler, "The Web Service Modeling Framework: WSMF", *Electronic Commerce: Research and Applications*, Vol. 1, No. 2, pp. 113-137, 2002.

[13] Klein, M., B. Konig-Ries and M. Muussig, "What is needed for semantic service descriptions? A proposal for suitable language constructs", *Proceedings of Inernational Journal of Web and Grid Services*, Vol. 1, No. 3/4, pp. 328-364, 2005.

[14] Voorhees, E., "Using WordNet for Text Retrieval", C. Fellbaum(Editor),"*WordNet: An Electronic Lexical Database*", pp. 285-303, The MIT Press, Cambridge,1998.

[15] Ginsberg, A., "A Unified Approach to Automatic Indexing and Information Retrieval", *IEEE Expert* , Vol. 8, No. 5, pp 46-56, 1993.

[16] Lee, J., M. Kim and Y. Lee, "Information Retrieval Based on Conceptual Distance in IS-A Hierarchies", *Journal of Documentation,* Vol. 49, No. 2, pp. 188-207, 1993.

[17] Agirre, E. and G. Rigau, "Word Sense Disambiguation Using Conceptual Density", *Proceedings of the 16th Conference on ComputationalLlinguistics*, Vol.1, pp. 16-22, 1996.

[18] Hovy, E., "Combining and Standardizing Large-scale, Practical Ontologies for Machine Translation and Other Uses", *Proceedings of the 1st International Conference on Language Resources and Evaluation (LREC)*, Granada, Spain, 1998.

[19] Wang, Y. and E. Stroulia, "Semantic Structure Matching for Assessing Web-Service Similarity", *Proceedings of the 1st International Conference on Service Oriented Computing*, Trento, Italy, 2003.

[20] Ruiqiang Guo, Dehua Chen, Jiajin Le, Matching Semantic Web Services accross Heterogeneous Ontologies, Proceedings of the 2005 The Fifth International Conference on Computer and Information Technology (CIT'05), 2005.

[21] Paolucci,M.; Kawamura,T.; Payne,T.; and Sycara,K. Semantic matching of web services capabilities. In Horrocks, I. And Hendler, J.eds.Proc. of the 1st International Semantic Web Conference(ISWC), pages333-347. Springer,2002.

[22] MSC: A Semantic Ranking for Hitting Results of Matchmaking of Services, Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)

[23] O. Taha, T. Dhavalkumar, Al-Dabass D., Semantic-Driven Matchmaking of Web Services Using Case-Based Reasoning, IEEE International Conference on Web Services, 2006.

[24] Şenvar, M. and Bener, A. , 2006, "Matchmaking of Semantic Web Services Using Semantic- Distance Information", Lecture Notes in Computer Science by Springer Verlag, ADVIS 2006, October, 18-20, İzmir, Turkey.

E.S. İlhan is a Computer Engineering graduate student at Bogaziçi University. He has his undergraduate degree in Computer Engineering department at Boğaziçi University in 2004. His research interests are semantic web services and software engineering. He is a software engineer now and has 3 years of working experience. Contact him at Boğaziçi University, Istanbul, Turkey, erdem.ilhan@cmpe.boun.edu.tr,

G.B.Akkuş is a Computer Engineering PhD student at Bogaziçi University. He has his undergraduate degree in Computer Engineering department at Ege University in 2002. His research interests are semantic web services and software engineering. He is a software engineer now and has 5 years of working experience. Contact him at Boğaziçi University, Istanbul, Turkey, gokay.akkus@boun.edu.tr,

A.B. Bener is a faculty member in the Department of Computer Engineering at Bogazici University. Her research interests include Web services, security, e-commerce, and m-commerce applications and software engineering. Bener has a PhD in information systems from the London School of Economics. She is a member of the IEEE, the IEEE Computer Society, and the ACM. Contact her at Boğaziçi University, Istanbul, Turkey, *bener@boun.edu.tr*

A development platform for distributed user interfaces

Anders Larsson, Magnus Ingmarsson, Bo Sun
Department of Computer and Information Science
Linköping University
Linköping, Sweden
andla@ida.liu.se, magin@ida.liu.se, x04bosun@ida.liu.se

Abstract

Developing user interfaces for a heterogeneous computing environment is a difficult challenge. Partial distribution of the user interface is even more difficult. In particular, providing developers with the means of describing and controlling how components are distributed and redistributed as devices are included or removed. We present an approach to overcome these challenges, by combining ontologies with a reasoning engine. Our tool MaDoE uses Protégé in combination with Jess to exemplify this in a simulated home setting. Our approach allows developers to take advantage of the formal knowledge in the ontologies as well harnessing the power of rules inside the expert system when they design distributed user interfaces.

Keywords: Distributed user interfaces, Ontologies, JESS , Mobile computing

1. Introduction

We move into a more and more distributed and heterogeneous computing environment where we see a development towards applications using multi-machine user interfaces and distributed user interfaces (DUI), and where several devices join to accommodate the user with contextually optimized interaction possibilities [16, 19]. It is becoming clear that current development methods are ill suited to handle this new environment. Today's methods lack support both for describing how to divide and distribute user interfaces and for allowing the user interface to reside on multiple devices.

In a series of projects [8, 4, 20] we have developed systems that take advantage of this new environment. Knowledge from these different systems has provided requirements for a DUI programing framework, called Marve. One essential component within this framework is a reasoning mechanism for describing distribution of ui-components at runtime among a variable set of devices according to a specific application strategy.

This paper presents the MaDoE module that allows detailed descriptions of devices' interaction capabilities in order to control component distribution using knowledge engineering tools together with a rule engine. This is done to accommodate the changing set of devices that the user has at his or her disposal.

While DUIs constitute a more modular and convenient interaction platform for the user, they also constitute a more difficult programming environment for the developers. This difficult environment leads to a need for tools for effective development of DUIs, much in the same way as for GUI-development [17]. We believe that these tools should provide developers with high level of abstraction to handle the complexity of distributing an application. For instance preparing an application for three devices might in the extreme case result in six different permutations of the user interface.

2. Background

2.1. Distributed user interfaces

In a mobile and ubiquitous computing world, users are constantly moving around, making their possible interaction a function of the available I/O-devices. To illustrate DUI:s we present an example below. *Lisa arrives at Stockholm Arlanda airport. As she passes through the security checkpoint and enters the departure hall she receives a message on her PDA. The message contains information about how she can access the lounge area using a special access code. To find the lounge Lisa walks over to one of the wall-mounted screens located throughout the airport. As she approaches the wall-mounted screen, a map of the airport appears, at the same time as Lisa is presented with a search interface on her PDA. She can then search for the lounge and navigate the map on the screen through her PDA. As she starts walking, the map is transferred from the wall* *mounted screen to her PDA and the search interface is removed. When she arrives at the lounge a small screen tells her to swipe her ticket and to enter the four digit access code she received in the message. Lisa swipes her ticket, enters her code on the interface presented on her PDA, and steps into the Lounge.*

A new programming model is needed to both handle and take advantage of this new reality. Distributed user interfaces has been a presented as one way of solving the UI-issues for these new demands [16, 5, 9]. DUI suggests that the interaction components for an application are spread out over the set of devices that the user currently has available to them. Constructing DUIs may be seen as the combination of the fields of migrateable applications and device-independent user interfaces. It is also a natural progression in UI-development from absolute positioning, via relative positioning, to distributed positioning of a UI's components. Allowing applications to be transferred between devices is not new. Over the years there have been more and more sophisticated attempts to allow the user interface to be transferred between devices at runtime, while the application is still running. A few of the first steps toward movable applications includes examples such as X11 remote displays, VNC, or Windows Terminal Server. However, all are based on the fact that an external process other than the application itself has control over its own migration. In contrast the DUI approach makes moving entire or partial UIs a directly usable asset in application designs. While DUIs constitute a richer interaction platform for the user, they also constitute a more difficult programming task for the developers. For a traditional GUI application the only outside influences the developer has to be able to handle is the change in size of the widow. This is commonly solved by layout managers that re-render the interface components when the window size is changed. In DUIs the developer needs to handle changes not only size, but also in the set of devices and the capabilities of the interface that is currently being used. This process can be described in the following steps

1. Applications starts
2. Devices added to application
3. Distribute components over the devices
4. Devices removed/added to application $\longrightarrow$ (3)

This can be referred to as the component distribution cycle [7]. New mechanisms are needed to allow developers to control the distribution cycle, just as developers already can control a traditional application's layout with layout managers. In the Marve platform this is handled by distribution managers which programmers can control. MaDoE adds a formal mechanism which allows for more general means of describing distribution well as a general descriptions of devices used in a DUI system. As the field of distributed user interfaces progresses we can expect to see the same level of abstraction that distributed systems have made for network programming. The distributed systems area has nearly eliminated the need for the programmer to do detailed implementations of network connectivity. Modern languages are fairly easy to use in this respect and provide simple yet powerful ways of handing network connectivity. An example of this is CORBA and Java's RMI.

2.2. Protégé

In this project, Protégé was used (version 3.0 Beta) to develop the ontology. Protégé is a

> "suite of tools to construct domain models and knowledge-based applications with ontologies. At its core, Protégé implements a rich set of knowledge-modeling structures and actions that support the creation, visualization, and manipulation of ontologies in various representation formats." [2]

Protégé is designed for system developers and domain experts to develop knowledge-based systems [1]. It is a tool that permits the integration of creating classes and entering instances of classes. Protégé also makes it convenient for domain experts to model concepts in a specific domain and to create applications in it in order to solve problems pertaining to this domain.

2.3. Jess

As stated earlier a rule engine is needed to control the distribution of components between service systems and mobile devices. In this project, Jess was chosen as rule language to create rule policies. Jess, Java Expert System Shell, is a rule engine and scripting environment written entirely in Sun's Java language. Jess rules may be likened to `if...then` statements in procedural language. The rules created in Jess are fired to take actions based on facts which in this project are items related to instances of classes from the ontology. An example of Jess rules is shown and described below.

```
Jess> (defrule allowed-person
"If a person is older than 22,
     print his (her) name."
?c <- (object (age ?x&:(> ?x 22)))
=>
   (printout t (slot-get ?c name)
    " is older than 22." crlf))
```

This rule has two parts, separated by the => symbol (which can be read as *then*). The first part is:

```
?c <- (object
(age ?x\&:(> ?x 22))).
```

The second part consists of the corresponding action:

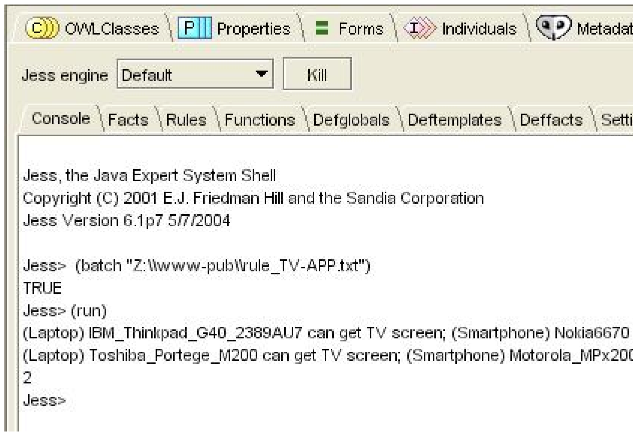
```
(printout t (slot-get ?c name)
  " is older than 22." crlf))
```

This rule means that whenever a person (instance) whose age (property) is more than 22, print his or her name (property).

2.4. JessTab

Since we want to make the Protégé ontology and Jess rules work simultaneously to simulate some interactions under different circumstances, a tool is needed to achieve this goal. JessTab [12] is a plug-in for Protégé that provides a console window allowing Protégé interact with Jess when Protégé is running. See Figure 1 for an example.

Through JessTab, Jess facts can be created from Protégé instances. In addition, Jess rules may be created which directly operate on Protégé's knowledge base (an aggregation of instances).



```
Jess, the Java Expert System Shell
Copyright (C) 2001 E.J. Friedman Hill and the Sandia Corporation
Jess Version 6.1p7 5/7/2004

Jess>  (batch "Z:\\www-pub\\rule_TV-APP.txt")
TRUE
Jess> (run)
(Laptop) IBM_Thinkpad_G40_2389AU7 can get TV screen; (Smartphone) Nokia6670
(Laptop) Toshiba_Portege_M200 can get TV screen; (Smartphone) Motorola_MPx200
2
Jess>
```

Figure 1. Jess running an example for a TV application

3. Related work

One of the earliest migration-aware applications was presented by Bharat and Cardelli in [6]. Their applications could be moved from one platform to another at runtime, provided that the operating system stayed the same and that it only allowed the entire application to be moved. Grolaux et al. [13] have with their system, Migratable, shown how migratable user interfaces can be used to achieve DUIs, by partial migration of the interface from one platform to another. They illustrate their approach by solving the painter's pallet problem (defined by Ayatsuka et al. in [3]) with only 0.5% extra code. Their research demonstrates that new tools are needed to allow construction of DUIs in a sound and professional manner.

Web-Splitter [14] is a project that shows a formal way of describing the transformation of web resources over a set of devices. Web-Splitter provides a framework that allows web pages to be split into personalized partial views depending on the users' different roles. These partial views can be further divided into xml-components that can be transferred to devices located in the users' proximity, thus allowing limited mobile devices to handle multimedia by augmenting them with co-located devices. Another project that aims to allow web systems to be distributed over a set of heterogeneous devices is presented by Vandervelpen et al. in [22]. Their contribution is twofold: it provides means for distributed user interfaces and a platform for collaboration.

Vandervelpen and Conix [10] show a system for high-level descriptions of user interfaces called Dygimes, which rest on a model-based user interface design approach that can be extended to support DUIs. Their approach is a step towards frameworks that aid user interface designers by allowing them to work within a distributed environment where the interface can be partially split over a set of devices.

The Pebbles project [18] (where handhelds and PCs work together) aims to spread computing functions and their related user interfaces over different I/O-devices. In an extension to the Pebbles project, the personal universal controller (PUC) project shows how a high-level description language can be used to describe remote-control facilities for mobile devices to stationary units such as VCRs, stereos and TV-sets. The high-level description can then be used together with automated user interface engines to generate a limited user interface for the stationary unit on the mobile device. A more theoretical work is presented by Demeure et al. in [11] in which they present a schema for classification of distributed user interfaces, with the aim of providing designers with a mechanism that can express the distribution of interfaces over different devices.

One trend in computer-supported cooperative work has been towards cooperative user interfaces, where a common user interface is shared among the users instead of each user having their own separate application running with its own user interface. Pioneering work for cooperative user interfaces was presented by Smith and Rodden [21] in Shared Object Layer (SOL). SOL allows the individual user's interfaces to be projected to devices from a common, shared interface definition. This enables users to be presented with only the tools and functions that they currently need, or are

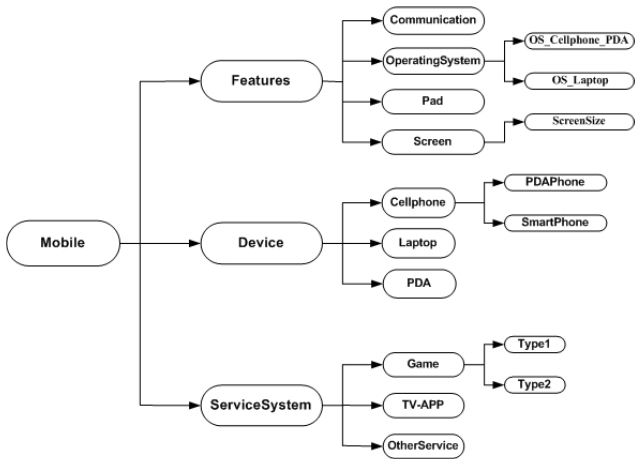


Figure 2. Overview of the ontology

allowed to use.

4. Marve Distribution Manager Ontology Engine (MaDoE)

MaDoE allows for a high-level method of describing the distributions of user-interface components over a changing set of devices. This is accomplished by using an ontology and a rule engine which can operate on the knowledge base described in the ontology. To have an easy way of redesigning and reorganizing the ontology we have used the Protégé ontology editor, and for this implementation we have chosen to use Jess as the rule engine.

To illustrate the functionality of MaDoE a small ontology was created. An overview of the ontology can be seen in Figure 2. This ontology both describes the different types of devices (Devices) that will be available as well as the properties (Features) of those devices. The domain of this model is oriented to mobile units which include Cellphone, PDA (Personal Digital Assistant), and Laptop. The Features included in this ontology are communication schemes (Bluetooth, infrared, etc.), operating systems, and interaction capabilities. The ontology also includes information about the different types of applications (ServiceSystem) that will be used. In this small example the applications are oriented in services that may be found within a home.

4.1. Classes

This ontology was implemented using Protégé. Figure 3 shows the class hierarchy inside Protégé. The major classes defined in the ontology are `ServiceSystem` and `Device` which has subclasses `Laptop`, `PDA`, and `Cellphone`. The class `ServiceSystem` represents the different applications available to the end-users. Some other

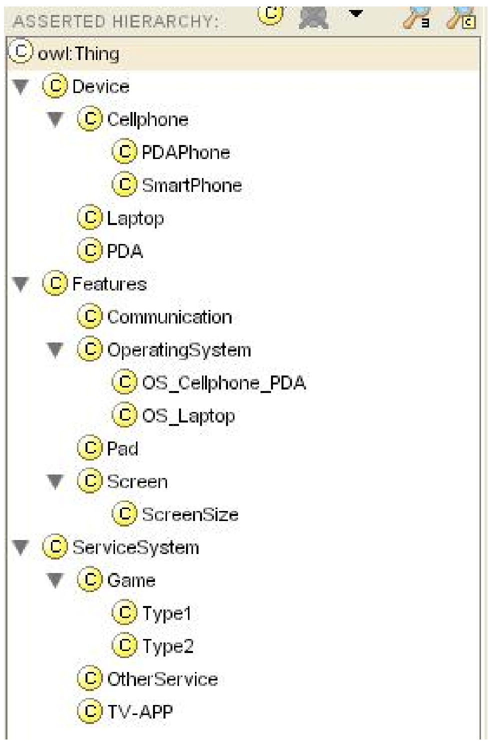


Figure 3. The class hierarchy in Protégé

classes are needed to represent the different features of the Devices. The class `Features` consists of the subclasses `Communication`, `OperatingSystem`, `Pad`, and `Screen`. The class `Communication` represents the wireless communication schemes (WiFi, Bluetooth, and Infrared) used by devices. The class `Pad` represents the interaction types (Keypad, Keyboard, Touchpad) on which users can interact with a device. The class `Screen` represents parameters related to the device's screen, such as screen size and color-depth. For each service in the ontology a set of interaction-components is defined, called sub-service in the ontology. These components are then available for distribution among the different devices.

These features are used by the rule engine to decide which components can be used on what device. More complex systems, such as MagUbi, containing world knowledge can take advantage of this to reason about why certain components should be placed on certain devices [15]. From Figure 4 it can be seen that all properties of `Device` are defined as Instance-type and arranged specifically to each subclass of `Features` and `ServiceSystem`. Then cardinality restrictions are defined and assigned to the properties of the `Device`. The following assumptions are made about the mobile devices in the ontology:

1. Each mobile device has only one operating system while running.

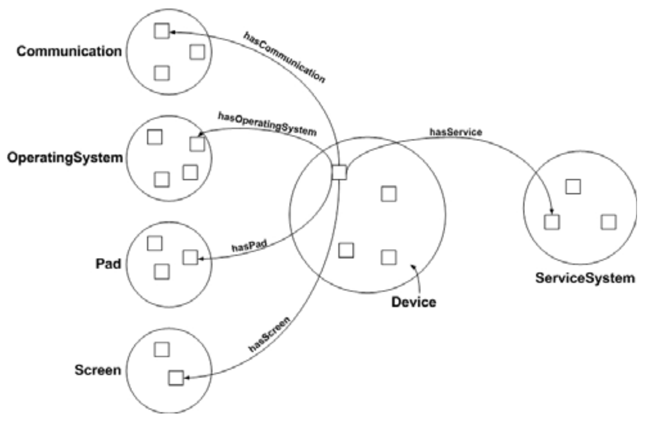


Figure 4. Properties of devices

2. Some mobile devices have been equipped with more than one pad, for example the Sony PEG-UX50 (PDA) has both Touchpad and Keyboard.

3. Some mobile devices have more than one communication port, for example the Acer TravelMate 661xvi (Laptop) has connectivity through Wi-Fi, Bluetooth and Infrared. In this work, it is assumed that a device can connect to a service system through only one communication method at a time.

4. One single device can run only one application provided by `ServiceSystem`.

4.2. Defining Rules

From this ontology we can provide developers with a high-level mechanism for describing how the applications in the ontology should distribute their different user interface components as devices are connected and disconnected from the application. This is accomplished in MaDoE by using Jess and the JessTab plug-in for Protégé. The JessTab provides automatic translation between Protégé classes to Jess facts. MaDoE includes built-in rules to ensure that UI components are not transferred to devices where they cannot be used.

For example, MaDoE enforces that a drop-down menu component is not transferred to a device that does not have interaction capabilities, like a TV, without simultaneous connection to, for instance, a cellphone from which the users make the selection. In this example the TV acts as a output device and the cellphone acts as the input device.

MaDoE also includes a set of rules that can be compared to layout mangers in traditional user interfaces. The MaDoE border distribution rule set, compared to Java Swing BorderLayout, ensures that a component named CENTER always displays on the largest screen available to the user. Formal and sound ways of describe application specific rules for controlling the behavior is also available in MaDoE through Jess rules. MaDoE acts as a plug-in to the Marve system. As devices are added or removed from an application in the Marve runtime, Jess rules are triggered and the underlying platform redistributes the components accordingly to the information provided by the rule engine.

5. Discussion

Allowing developers to control UI-component distribution through the use of ontologies and rule engines provides a sound and formal way of expressing application behavior as devices are added or removed. A rule engine can ensure that components are not transferred to devices where the components cannot be used. To allow for application specific behavior, developers also need to be able to describe their own rules for how an application's user interface should change over time as the number of devices connected to an application change. As the rules supplied by programmer might contradict the built in rules in MaDoE, a set of priorities is required to govern rule precedences. This is needed but not available today in MaDoE.

In MaDoE, knowledge about devices and applications is described in an ontology and then transformed into facts that can be used by the rule engine. This is done to allow a higher abstraction level for describing how an application should change as the number of connected devices are changing. The output from the rule engine is then interpreted by the Marve platform. In the future there might be a way of letting the device ontology be available in the Marve framework and then have a rule engine in the framework itself. Today Marve has a small set of distribution managers which can distribute components according to a predefined schema with a limited degree of freedom for the application developer.

6. Conclusion

To provide developers with the right tools to build and maintain systems that support distributed user interfaces, it is essential to have formal and sound way of describing how the application's user interfaces should change over time as devices are added or removed. The MaDoE module for the Marve framework also allows developers to use ontologies to describe the properties of both devices and information services. If combined with a system with world knowledge such as Magubi [15], proper DUIs can be constructed as well to make sure that mistakes such as displaying the UI on an inaccessible display do not occur.

Users are today equipped with more and more devices (such as traditional personal computers, PDAs, cell-phones, or hybrids like smart-phones) for accessing different information systems. For the developer it is important to accommodate users with adequate interaction capabilities. This project provides a model of mobile devices as well as a service system. The description of this model (ontology) combined with some rules is helpful in designing a service system. This is especially true with regards to the rule engine that offloads the burden of deciding what goes where and when in this highly dynamic environment.

A clear hierarchy of defined classes with related information (properties, instances, and comments) is illustrated. This model is created by Protégé. It aims to give designers of service system a rudimental prototype about different mobile devices, their features, internal structures, etc.

Jess rules created for this model give a way to simulate the interaction between mobile units and service system. They provide some basic ideas about how to design rule policies for the rule engine of a service system. Even though these rules have only been applied in a laboratory setting so far, they could still be incorporated into research prototypes of service systems to test how they are going to work in actual circumstances.

This project shows how developers and system designers can take advantage of such a model in both designing as well as deploying information systems for mobile devices. The rule engine also gives developers a high-level method of defining the requirements and behavior of the system as a whole. The project also demonstrates how services can be split up into sub-services (components) to enable the development of applications supporting DUIs.

References

[1] *The Protege user's guide*. http://protege.stanford.edu/doc/users_guide/index.html.

[2] *What is Protégé?* http://protege.stanford.edu/overview/index.html.

[3] Y. Ayatsuka, N. Matsushita, and J. Rekimoto. Hyperpalette: a hybrid computing environment for small computing devices. In *CHI '00: CHI '00 extended abstracts on Human factors in computing systems*, pages 133–134, New York, NY, USA, 2000. ACM Press.

[4] A. Berglund, E. Berglund, A. Larsson, and M. Bång. The paper remote: An augmented tv guide and remote control. *Universal Access in the Information Society, Springer*, 2004.

[5] E. Berglund and M. Bång. Requirements for distributed user-interface in ubiquitous computing networks. In *MUM2002, Mobile and Ubiquitous MultiMedia connference*, 2002.

[6] K. A. Bharat and L. Cardelli. Migratory applications. In *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 132–142, New York, NY, USA, 1995. ACM Press.

[7] M. Bång, A. Larsson, E. Berglund, and H. Eriksson. Distributed user interfaces for clinical ubiquitous computing applications. *International Journal of Medical Informatics*, pages 545–551, 2005.

[8] M. Bång, A. Larsson, and H. Eriksson. Nostos: A paper-based ubiquitous computing healthcare environment to support data capture and collaboration. In *Proccedings of the 2003 American Medical Informatics Association Annual Symposium*, pages 46–50, 2003.

[9] M. Bång, A. Larsson, and H. Eriksson. Design requirements for ubiquitous computing environments for healthcare professionals. In *Proceedings of Medinfo 2004*, 2004.

[10] K. Coninx, K. Luyten, C. Vandervelpen, J. V. den Bergh, and B. Creemers. Dygimes: Dynamically generating interfaces for mobile computing devices and embedded systems. In *Mobile Human-Computer Interaction - Mobile HCI 2003*.

[11] A. Demeure, G. Calvary, J.-S. Sottet, and J. Vanderdonkt. A reference model for distributed user interfaces. In *TAMODIA '05: Proceedings of the 4th international workshop on Task models and diagrams*, pages 79–86, New York, NY, USA, 2005. ACM Press.

[12] H. Eriksson. The jesstab approach to protégé and jess integration. In *Proceedings of Intelligent Information Processing (IIP 2002)*, pages 237–248, 2002.

[13] D. Grolaux, P. V. Roy, and J. Vanderdonckt. Migratable user interfaces: beyond migratory interfaces. In *Proceedings of MOBIQUITOUS 2004. The First Annual International Confere nce on Mobile and Ubiquitous Systems: Networking and Services*, pages 422–30. IEEE Comput. Soc, 2004.

[14] R. Han, V. Perret, and M. Naghshineh. Websplitter: a unified xml framework for multi-device collaborative web brows ing. In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported co operative work*, pages 221–230, New York, NY, USA, 2000. ACM Press.

[15] M. Ingmarsson. *Modelling User Tasks and Intentions for Service Discovery in Ubiquitous Computing*. Number 1305 in Linköping Studies in Science and Technology. Linköpings universitet, 2007.

[16] A. Larsson and E. Berglund. Programming ubiquitous software applications: requirments for distributed user interface. In *Proceedings of The Sixteenth International Conference on Software Engineering and Knowledge Engineering (SEKE 04)'*, 2004.

[17] B. A. Myers. User interface software tools. *ACM Transactions on Computer-Human Interaction*, 2(1):64–103, 1995.

[18] B. A. Myers. Using handhelds and pcs together. *Communications of the ACM*, 44(11):34–41, 2001.

[19] B. A. Myers, J. Nichols, J. O. Wobbrock, and R. C. Miller. Taking handheld devices to the next level. *Computer*, 37(12):36–43, 2004.

[20] M. Sjölund, A. Larsson, and E. Berglund. The walk-away gui: Interface distribution to mobile devices. In *IASTED-HCI 2005*, 2005.

[21] G. Smith and T. Rodden. Sol: a shared object toolkit for cooperative interfaces. *Int. J. Hum.-Comput. Stud.*, 42(2):207–234, 1995.

[22] C. Vandervelpen, G. Vanderhulst, K. Luyten, and K. Coninx. Light-weight distributed web interfaces: Preparing the web for heterogeneous environmen. In *5th International Conference, ICWE 2005, Sydney, Australia*, volume 3579/2005, pages 197–202. Springer Berlin / Heidelberg, 2005.

A Dynamical System approach to Intrusion Detection Using System Call Analysis[1]

Nitin Kanaskar, Remzi Seker and S. Ramaswamy
Knowledge Enterprises for Scalable Resilient Infrastructures
Computer Science Department, University of Arkansas at Little Rock, Little Rock, AR 72204, USA
{nvkanaskar, rxseker}@ualr.edu, srini@acm.org

ABSTRACT

Code injections can aid successful intrusion attempts, thereby allowing viruses and worms to spread. Current research into intrusion detection is notably focused on application behavior profiling through system call trace analysis. Studying the system call layer has been identified as a potential approach to render revealing details about an application's behavior. System call sequences available from the execution trace of an application can be subjected to different modeling techniques to approximate the application's normal execution. This research views application programs as dynamical systems, and applies dynamical system analysis tools operating on time series data, merely the system calls made by an application, to identify the degree of determinism in a dynamical system. There is some prior work in the literature analyzing programs as dynamical systems, but they lack proper utilization of dynamical system formalisms and associated analysis tools. In our research we utilize a set of dynamical system analysis tools composed of Approximate Entropy, Central Tendency Measure, and Recurrence Plot derived measures. Our initial results are promising in detecting code injections.

Keywords - ***System call sequence, Intrusion Detection, Approximate Entropy, Recurrence Plots, Central Tendency Measure***

1. Introduction

Intrusion detection assumes a vital role in the overall information security framework of any computing network. A majority of intrusive activities takes place as the direct outcome of malicious code injection into an applications' execution memory space. System call analysis has proven to be a very effective tool for detecting the code injections, and therefore, system intrusions. The primary objective of our work can be described as the attempt at the timely detection of system call pattern change to recognize abnormal application behaviors. Through this paper, we introduce the first thorough application of a dynamical system theory approach to system call analysis with the distinct goal of detecting code injection attacks. To accomplish this, we characterize the normal behavior of the application on the basis of certain dynamical system characteristics. Values of these dynamical system characteristics are observed for the abnormal behavior of the application and compared with its normal behavior pattern. Collectively, the analysis using these tools facilitates us to distinctly differentiate between normal and abnormal behaviors of the application.

Section 2 explains some basic theoretical background on dynamical system analysis necessary for system call analysis. Section 3 describes the prototype test environment. Section 4 explains the various analyses approaches. Section 5 discusses and analyzes the results of our prototype implementation. Section 6 compares the dynamical system approach with a few of the relevant research works. Section 7 summarizes the work and presents our conclusions.

2. Dynamical System Approach

Dynamical system theory and chaos theory have been explored by research communities to understand and explain many apparently random phenomena in nature like turbulence in sea, atmosphere, fluctuation in wildlife populations, accumulation of vehicles on highways, oil flow in underground pipes, electronic devices and many other universally diverse events [1]. We believe that a software application's behavior evolution over time shows a similar kind of dynamism. Hence in this paper, we use the system call trace of a software application as the observable characteristic of a complex dynamical system whose values are in the form of a one dimensional time series. Simply put, a computer program's normal behavior can be characterized by applying system behavior analysis techniques used for studying dynamical systems.

System call trace generated by the application program may be considered as observable through a one dimensional time series from which we can reconstruct the state space of the application. Any activity by an application requires invocation of a particular sequence of system calls. So, given a system call, we can predict as to what would be the next system call within a certain degree of probability, and hence a degree of determinism. But, in the long run it is difficult, if not impossible to estimate which system call is going to be executed. This uncertainty articulates the randomness inherent in an application's long term behavior. We believe an application's long term behavior dynamics can be better understood by reconstructing the application's

[1] This research work was funded, in part, by an MRI grant from the National Science Foundation (Grant #: CNS – 0619069).

state space and then applying certain dynamical system analysis tools to analyze it. We choose Approximate Entropy, Central Tendency Measure, and Recurrence Plots analysis techniques to study and characterize applications' long term behavior. These measures define a system's characterization in terms of degree of determinism, similarity, and rate of variability and are the characteristics which we find useful for creating an application's normal behavior profile. Figure 1 presents our overall approach for using an application's system call sequence as a dynamical system observable.

2.1 Approximate Entropy

Using this measure, we study an application's behavior from the perspective of the system's information complexity and utilize the state space reconstructed for the application from its system call time series data. The Approximate Entropy measure was proposed by Pincus [2] to assess a system's information complexity. It is a statistical measure capable of classifying complex systems with relatively few data points. Approximate Entropy has been successfully utilized to quantify complexity in physical systems as well as physiological systems. It works satisfactorily on small lengths of time series data to give their complexity measure [3].

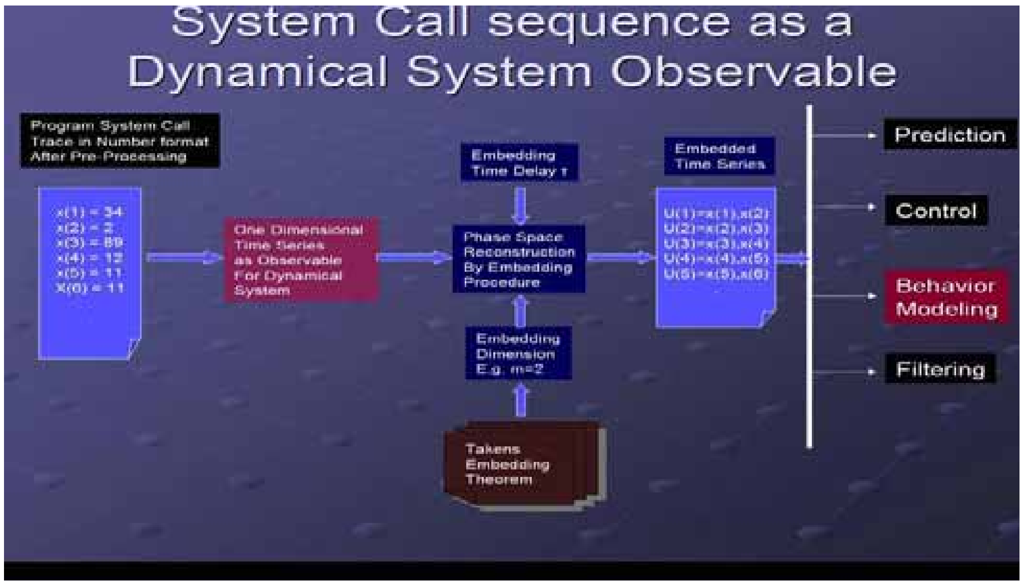

System Call sequence as a Dynamical System Observable
Program System Call Trace in Number format After Pre-Processing
x(1) = 34
x(2) = 2
x(3) = 89
x(4) = 12
x(5) = 11
X(6) = 11
One Dimensional Time Series as Observable For Dynamical System
Embedding Time Delay τ
Phase Space Reconstruction By Embedding Procedure
Embedding Dimension E.g. m=2
Takens Embedding Theorem
Embedded Time Series
U(1)=x(1),x(2)
U(2)=x(2),x(3)
U(3)=x(3),x(4)
U(4)=x(4),x(5)
U(5)=x(5),x(6)
Prediction
Control
Behavior Modeling
Filtering

Figure 1: Dynamical System perspective for System Call analysis

Any kind of intrusive code injection will give rise to a sudden drift in the application's system call sequence pattern which translates into a changed – either increased or decreased – value of the system's complexity. Hence, we study the effect of code injection into the application's memory space by its effect on changing the application's behavior. We give below a brief description of the procedure to calculate Approximate Entropy from a one dimensional time series:

Consider a one dimensional time series

$$u = u(1), u(2), u(3), \ldots\ldots, u(N) \quad \cdots\cdots\cdots 1$$

All the scalar components of this time series are equi-spaced in time. A series of m-dimensional points x(1), x(2), x(3),….x(N-m+1) is formed from the equation 1 such that

$$x(i) = u(i), u(i+1), u(i+2) \ldots u(i+m-1) \quad \cdots\cdots\cdots 2$$

Each of these points specifies a point in the reconstructed m-dimensional state space. A measure C_i^m is defined as

$$C_i^m(r) = N_{ij} / (N - m + 1) \quad \cdots\cdots\cdots 3$$

Where

N_{ij} = Number of j such that Euclidean distance [x(i), x(j)] < r; where r is the radius of the sphere in m-dimensional space centered at x(i)

Euclidean distance is defined to be the length of difference between two vectors (state space points) which is

$$|x(i) - x(j)| = \sqrt{\sum_{k=0}^{m-1} [u(i+k) - u(j+k)]^2} \quad \cdots\cdots\cdots 4$$

We define

$$C^m(r) = (N - m + 1)^{-1} \sum_{i=1}^{N-m+1} C_i^m(r) \quad \cdots\cdots\cdots 5$$

$$\Phi^m(r) = (N - m + 1)^{-1} \sum_{i=1}^{N-m+1} \log C_i^m(r) \quad \cdots\cdots\cdots 6$$

Approximate Entropy for some fixed values of m and r is defined as:

$$ApEn(m,r) = \lim_{N\to\infty}[\Phi^{m}(r) - \Phi^{m+1}(r)] \cdots\cdots 7$$

2.2 Central Tendency Measure

Here, we attempt to establish some measure of the chaotic behavior in the application's system call time series by calculating its central tendency measure (CTM)[4]. CTM is a metric to evaluate the degree of variability in a given data. Second order difference plots centered around the origin give the rate at which data assumes variations in values. The more these values are distributed around the origin, the more the randomness is exhibited by the data. CTM has been employed in the analysis of various physiological processes like heart rate variability and behavior of schizophrenic patients.

In our work, variability scores of the program's behavior before code injection and the one after code injection need to be distinguished from one another as degree of variability is one of the criteria used to distinguish the two scenarios. Second order difference plot for a time series a(1), a(2), ….a(n) – obtained by plotting a(n+2)-a(n+1) Vs. a(n+1)-a(n) - acts as a tool to measure this variability factor. In a time series a(1),a(2),…..a(n),a(n+1),a(n+2) of length N, if r is denoted as the radius of the sphere around the origin, then

$$CTM = \left[\sum_{i=1}^{N-2}\delta(d_i)\right] \Big/ (N-2) \cdots\cdots 8$$

Where,

$$\delta(d_i) = 1, \; if \; \{[a(i+2)-a(i+1)]^2 - [a(i+1)-a(i)]^2\}^{0.5} < r$$
$$= 0, \text{ Otherwise} \cdots\cdots 9$$

The value of radius r is selected depending upon the nature of data [5].

2.3 Recurrence Plots

This is a recently developed dynamical time series analysis technique by J. P. Eckmann [6]. It graphically demonstrates time correlation between different points on the state space of a dynamical system. Point (i, j) in a Recurrence Plot is marked black (or 1) if two points representing the system states at instants i and j are close enough as defined by a criterion of Euclidean distance r. Thus,

$$RP(i,j) = 1, \; if \quad d[x(i),x(j)] \quad <= \quad r \cdots\cdots 10$$
$$=0, \text{ Otherwise}$$

Points x(i) and x(j) are part of an embedded time series data. The ith row of this multidimensional vector represents the system state at ith instant. Recurrence Plots have been exploited to discern hidden patterns and non-stationeries in time series data for physiological systems. Different structural elements in Recurrence Plots denote certain qualitative aspects of the time series data in terms of determinism, recurrent patterns. We define Percent Recurrence, Percent Determinism, and Percent Ratio [7] to characterize an applications' normal behavior.

2.3.1 Percent recurrence

Percent Recurrence gives the fraction of points in multidimensional state space repeating previous system dynamics. It helps us distinguish a process with periodic dynamic behavior from that with an aperiodic behavior. More the points that are observed at same states of a dynamical system, the more periodicity exhibited by its reconstructed state space.

2.3.2 Percent Determinism

Percent Determinism is associated with line structures present in a Recurrence Plot. A Line of Identity (LOI) is formed on a plot for all points where i=j. This is the line having the slope of one which passes through the origin and divides plot area into two congruent triangles. There may appear other line structure(s) which are parallel to LOI. Such a line in the plot is formed by points (1's) that are diagonally adjacent with no white spaces (0's) in between. For example, if pairs of consecutive points [x(i),x(j)], [x(i+1),x(j+1)], [x(i+2),x(j+2)]…[x(i+N),x(j+N)] in multidimensional state space of a dynamical system exhibit the same dynamics, then the corresponding points placed in the Recurrence Plot form a line parallel to the LOI. Percentage of points in these lines articulate how much structure of the state space repeats on consecutive points of the state.

2.3.3 Percent Ratio

Percent Ratio is the ratio of Percent Determinism to Percent Recurrence in the plot. This quantity captures the extent to which the system state space is experiencing sudden variations. So it is an effective indicator of sudden transitions in state dynamics of a process. All of the above defined Recurrence Plot parameters strongly highlight presence of hidden rhythms, and determinism characteristics in data.

3. Test Environment

We simulate the code injection event for the test application by changing its source code and recompiling it into a new executable program. To mimic code injection, the added code is a simple 'for loop' inserted into one of the main source files of the application to invoke certain system calls repeatedly. Since most common viruses and worms execute certain instructions repeatedly, we believe this to be a viable scenario for such intrusions. The injected code causes the runtime executable image of the program to change and this aptly simulates the code injection phenomenon in such intrusions. The application chosen for the prototype implementation of the dynamical system analysis approach is Apache Web Server 2.2.2 on Fedora Core 4 Linux platform with the kernel version 2.6.11. First, we observe

Apache's normal behavior profile by viewing its system call trace after subjecting it to a pre-defined set of HTTP requests. Then, we set up our test environment such that Apache is the only network daemon running on the system to minimize the number of free variables in the system and to better understand Apache's behavior profile. The Linux strace utility is used to capture all of the system calls invoked by Apache into a text file. One of the source files – request.c – in the Apache server's source tree was then changed to add the injected code mentioned earlier; since this is the file accessed by Apache for servicing all HTTP requests. System call trace files for this new Apache server executable in response to the same set of HTTP requests were then collected.

Later, with the unique call mappings present in /usr/include/asm/unistd.h – universally standard to all Linux platforms – the test files are converted into a numerical format. For simplicity, the Apache server's library calls are ignored for analysis purposes. To reduce the complexity of processing, the system call parameters are also ignored. Apache processes invoke – waitpid() and select() system calls – when they are idle; hence these are also ignored as our intent is to profile Apache's behavior when it services HTTP requests. On Linux platforms, when Apache server makes a socket related system call, it is redirected to the respective system call via a common system call – socketcall(). In other words, the unistd.h file contains mappings only for socketcall(). As a result, for our prototype, we map all the invocations of socket related system calls by Apache to socketcall(). There is a category of system calls in Linux kernel which are marked as deprecated. There does not exist a mapping for these system calls in unistd.h file. We decided to ignore these system calls from our dynamical system analysis approach. Needless to say, by taking both the old system calls and the network system calls into account, a better approximation of the application's behavior can be realized.

4. Analysis Approach

4.1 Non-Clustered Analysis

From the one dimensional time series, the state space of Apache's behavior is reconstructed by the process of embedding. Embedding, achieves the aim of better approximation of the time evolution of the system's behavior by reconstructing the state space with specific embedding dimension m and time delay τ values.

4.1.1 Subsystem Approach

For all children processes of Apache web server, one dimensional number time series data are retrieved after the pre-processing stage. Each of these time series is treated as an individual subsystem and subjected to the embedding procedure. Then, these multidimensional time series are subjected to dynamical system analysis methods - Approximate Entropy, Recurrence Plots, and Central Tendency Measure (CTM). For each of the time series, first 1000 data points are processed in a cumulative order starting with 500 data points with an increment of 100. Average values of all the measures are plotted against the changing data lengths using scripts in Matlab™ 7.0.14.

4.1.2 Children Processes as System Dimensions

All of the children processes are considered as individual dimensions of the application's state space. We convert the data from the system call trace of all process specific files into a single numerical matrix having each process's trace in each column of the matrix. Each row represents a point in m-dimensional space. The matrix data is processed in the same cumulative fashion as described for the subsystem approach. A modified definition of Approximate Entropy is employed for this scenario wherein only $\Phi^m(r)$ is calculated. Calculation of $\Phi^{m+1}(r)$ cannot be done with the same data

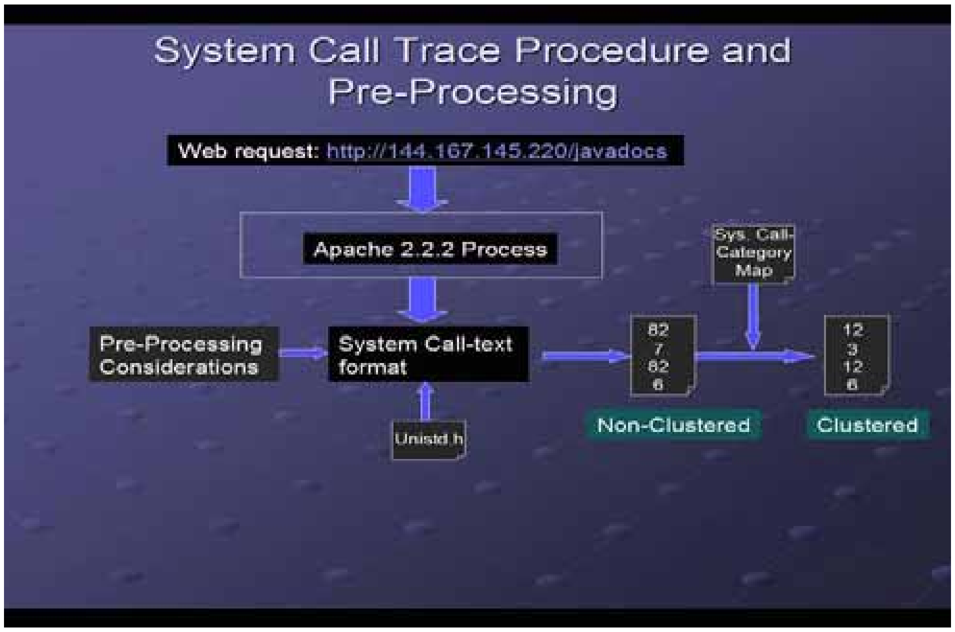

System Call Trace Procedure and Pre-Processing
Web request: http://144.167.145.220/javadocs
Apache 2.2.2 Process
Sys. Call-Category Map
Pre-Processing Considerations
System Call-text format
82 7 82 6
12 3 12 6
Unistd.h
Non-Clustered
Clustered

Figure 2: Apache system call trace pre-processing for Clustered and Non-Clustered approach

vector by changing the number of dimensions. Hence we come up with modified definition of Approximate Entropy ($\Phi^m(r)$).

4.2 Clustered Analysis

In this approach, all of the system calls from the trace are mapped to a particular functional group. Two user defined configuration files are created for this purpose. One file contains all the functional categories mapped to unique numbers. The other file contains all system calls mapped to the respective functional category. Figure 2 gives the block diagram representation of the pre-processing stage. After the pre-processing stage, the trace file contains one dimensional time series having system call category numbers as equi-spaced elements.

The intention behind clustering system calls is to capture Apache's behavior at a more abstract level. A system call category is more indicative of Apache's functionality, and hence in turn, its behavior.

5. Results and Analysis

A collective analysis of all the selected dynamical system measures reinforces our confidence level in the decision making process. The graphs of Approximate Entropy, Recurrence Plot parameters, and CTM are plotted against varying data lengths for the pre-code injection and the post-code injection scenarios.

We investigate the dynamical system characteristics of Apache web server from several different perspectives: a. Non-Clustered subsystem approach, b. Non-Clustered process state variable approach, c. Clustered subsystem approach, d. Clustered process state variable approach. For the subsystem approaches (a and c), the first 1000 data points from the time series are subjected to embedding with dimension values ranging from 2 to 15 and delay varying from 1 to 3. CTM is independent of embedding procedure, thus it is not calculated for the variations of embedding dimension. The values of embedding dimension (m) and time delay (τ) that give the best possible results for shortest data lengths are selected. By best result we mean maximum discrimination between pre and post code-injection states. The distinction between the pre-code injection and post-code injection scenarios need to be achieved for as few data points as possible (the fewer the system calls needed for detection, the more responsive and easier the implement the proposed approach is). For the process state variable approaches (b and d), the system state variables or dimensions are assumed to be equal to the number of children processes; hence they are not subjected to the embedding procedure.

The graphs for Apache's system call trace for the embedding dimension (m) 2 to 5 and time delay (τ) 1 for Non-Clustered subsystem approach are illustrated in Figures 3 and 4. Figures 5 and 6 show the graphs obtained with children processes as state variables, on two different times. We observe here that the subsystem approach gives superior results.

To validate the proposed methodology, along with Apache web server, we tested a few other daemon application programs like vsftpd, DNS server named, cupsd (Unix Printing Service) and subjected them to a similar dynamical system analysis process. Preliminary results substantiate our claim that a thorough dynamical system approach allows us to discriminate between pre and post code injection. For repeatability of the results, we confirmed the validity of these measures by repeating the whole procedure on two

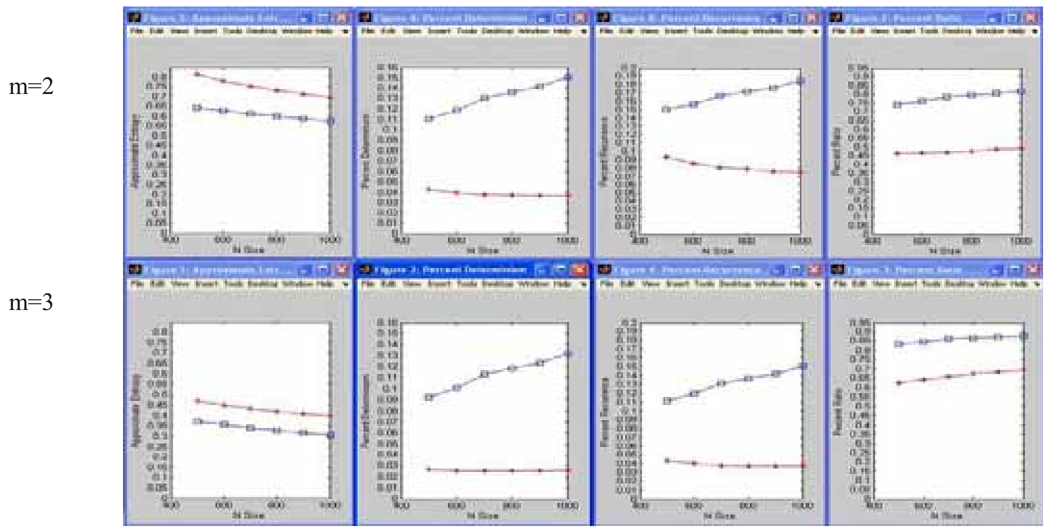


Figure 3: Approximate Entropy, Percent Determinism, Percent Recurrence, and Percent Ratio for τ =1. Curve with squares is for post-code injection and the curve with stars is for pre-code injection scenario. (Non-Clustered Subsystem approach)

different times. The behavioral changes due to code injection are reflected prominently through the graphs we obtain for these applications. We get the best distinction results for embedding dimension 2 and embedding delay 1.

m=4

m=5

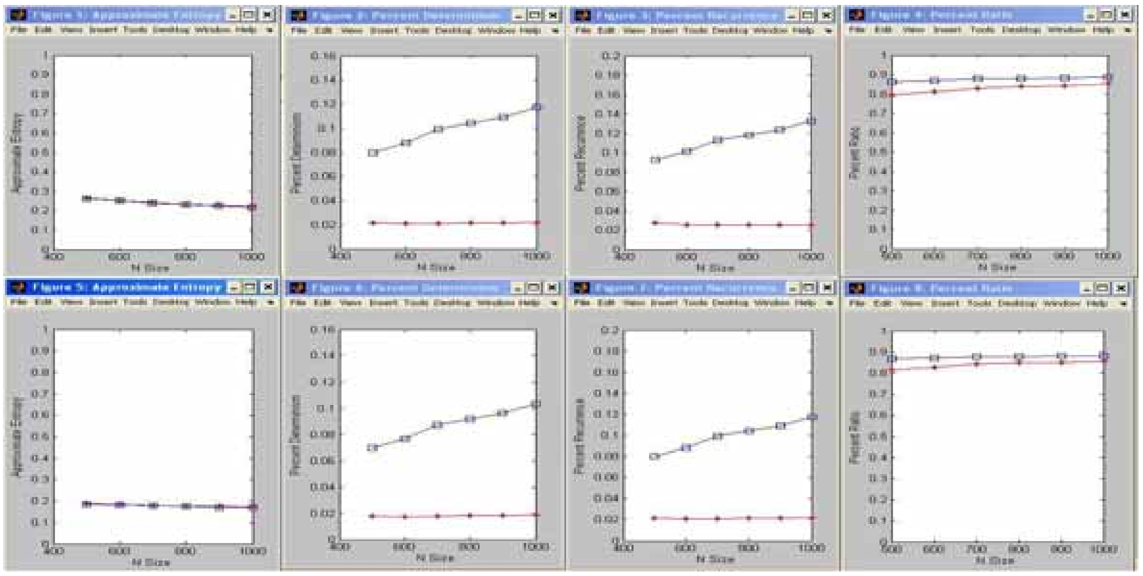


Figure 4: Approximate Entropy, Percent Determinism, Percent Recurrence, and Percent Ratio for m=4 and m=5, t=1. (Non-Clustered Subsystem approach)

Date 1

Date 2

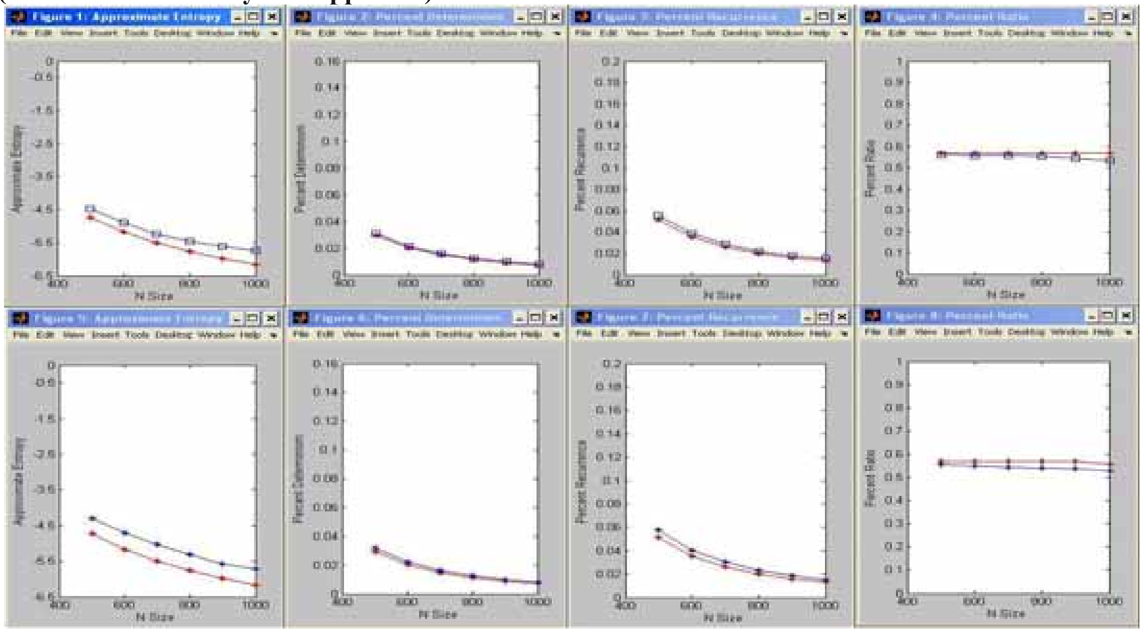


Figure 5: Modified Approximate Entropy, Percent Determinism, Percent Recurrence, and Percent Ratio for the children processes as the state variables on different times

6. Comparative Analysis

To distinguish our proposed approach, in this section we relate our approach with some of the relevant research approaches proposed in literature until now. Hofmeyr, et.al [8] proposed the definition of normal application behavior using small unique sequences of system calls. Using a simplistic measure of Hamming distance, they identified abnormal sequences. They unknowingly utilized the process of embedding by defining a database of normal system call sequences of fixed length. However, they warn that the Hamming distance is not a formally defined and proven metric to determine the abnormality of sequences. Thus, the simplicity advantage is overshadowed by the lack of proper formal analysis of the system call layer. Our approach is based on widely proven techniques employed successfully in diverse fields [1].

Hofmeyr and Kosoreow [9] defined variable length sequences – macros – and deterministic finite automata (DFA) to define normal behavior of application. The DFA tended to be quite large for large applications like sendmail. There was no ideal method defined to create the DFA and was created manually for any application to be tested. Our analysis approach can be fully automated, and is indeed one future improvement planned for our work.

Jones and Li [10] incorporated temporal information between system call pairs in the database. Under the assumption of normal distribution for system call timing, they ignored some of the normal behavior (e.g. I/O) system call timing information because of the large variance. We accommodate all system call sequences for our analysis except the socket related calls, library calls, and some older system calls. Our future improvement plan includes support for the socket calls and the older system calls.

Cabrera, et. al. [11] came up with the concept of anomaly dictionary consisting of anomalous sequences for the classification of anomalies. This feature was built upon the dictionary of normal sequences proposed by Hofmeyr et. al. [8]. This approach required the creation of the anomaly dictionary from known anomaly sequences. Our approach, based on dynamical system analysis techniques is independent of any such database of anomalous sequences.

Nguyen, et. al. [12] developed Buffer Overflow attack Detection system with Linux kernel modification. Their system was based on a database of all children processes forked by a given process. The main drawback of their method was there were many processes whose children processes could not be ascertained apriori. The training period needed for their prototype to be trained was 3 months which is quite a long time.

Qiao, et. al. [13] put forth Hidden Markov Model (HMM) for profiling application behavior profiling. The number of states in HMM was determined experimentally as there was no formal method for doing that. The training process of HMM took a long time.

In comparison to all of the above mentioned approaches, we have shown that a dynamical system analysis approach can fare well in many respects. First, experimentally, we have determined that the minimum number of data points required for the method to be successful is around 500. Second, this method does not require a full fledged database to store normal application sequences. Third, and very importantly, the method is independent of the timing characteristics of system calls.

Mutz et al. [14] adopted an entirely different methodology from all of the above approaches in that they formulated system call argument models in terms of their lengths and character distribution. They did not take system call sequences into consideration. They focused especially on detection of mimicry attacks which cannot be detected by

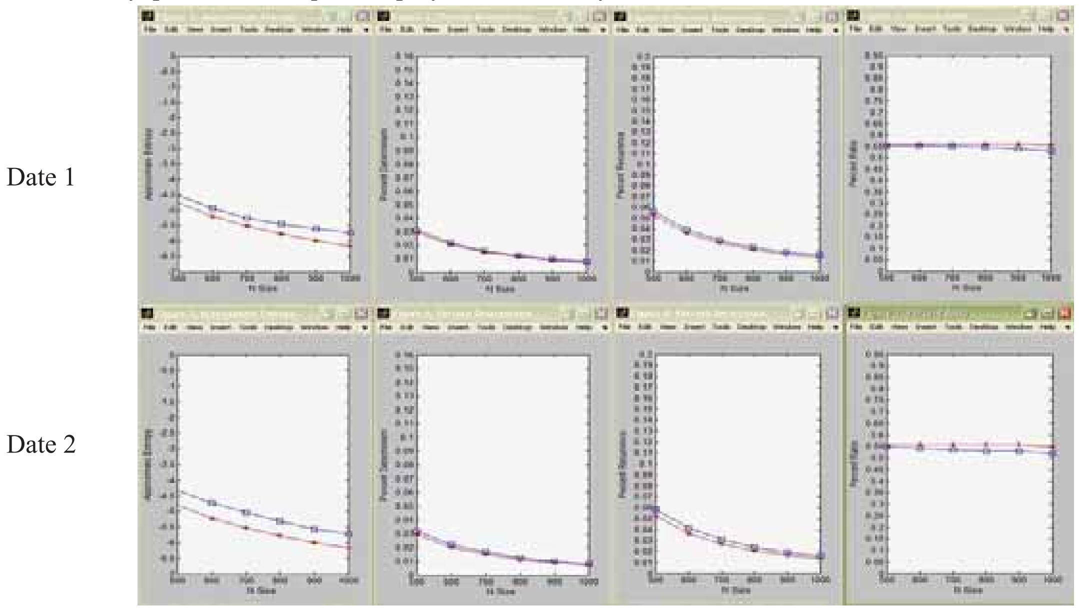


Figure 6: Modified Approximate Entropy, Percent Determinism, Percent Recurrence, and Percent Ratio for the children processes as state variables (Clustered)

most of the above research approaches. We plan to incorporate system call argument modeling into dynamical system perspective in the future.

7. Discussions and Conclusions

Embedding procedure demands specific attention with respect to future work. The parameters required by embedding – the embedding dimension m and embedding delay τ – are determined heuristically in our prototype. A more formal method needs to be devised for the determination of these parameters which will give their optimum values for a particular application. However, all pertinent applications of embedding procedure in the literature have employed a heuristic approach for ascertaining m and τ. We also need to include alternative scheme for mapping socket system calls and older system calls. This enhancement will definitely improve the detection capability of our tool.

By observing the collection of graphs of the dynamical system characteristics – Approximate Entropy, Recurrence Plot derived measures, and CTM – we draw certain inferences. The Subsystem approach gives superior results than the 'children process as state variables' approach. Embedding dimension (m) value 2 and embedding delay (τ) value 1 give the best detection capability for Non-Clustered as well as Clustered subsystem approaches. As the number of embedding dimension increases, majority of dynamical system characteristics tend to be similar for the pre-code injection and the post-code injection scenarios. As a higher number of dimensions are used to recreate the system state space, its attractor is stretched more and this effectively culminates in dispersing the state space points away from each other. Hence, the difference between the characteristics for the two scenarios decreases. Clustered system call analysis gives us similar results as the Non-Clustered approach for all the test applications. We have tested the dynamical system approach on Apache 2.2.2, cups 1.2.7, vsftp 2.0.5, and bind 9.3.3 on Fedora Core 4 Linux platform. A collective analysis of the dynamical measures of information complexity, degree of determinism, and periodicity for these applications gives us promising results in term of detecting abnormal sequences from the normal ones. Through this paper, we have shown that application behavior modeling founded on the principles of dynamical system theory can potentially be leveraged to profiling the behavior of application programs for Intrusion Detection.

REFERENCES

[1] S. Sharma. (2006 March). An Exploratory Study of Chaos in Human-Machine System Dynamics. IEEE Transactions on Systems, Man and Cybernetics. [Online], Part A. 36(2), pp.319-326.

[2] Steven Pincus. (1991 March). Approximate Entropy as a Measure of System Complexity. Proceedings of the National Academy of Sciences. [Online]. 88. pp.2297-2301.

[3] M. Akay. (2005 November). Influence of the Vagus Nerve on Respiratory Patterns during Early Maturation, IEEE Transactions on Biomedical Engineering. [Online]. 52(11), pp.:1863-1868.

[4] M.E.Cohen, D.L.Hudson, P.C.Deedwania. (1996 September). Applying Continuous Chaotic Modeling to Cardiac Signal Analysis. IEEE Engineering in Medicine and Biology. [Online]. 15(5), pp. 97-102.

[5] D.L.Hudson, M.E.Cohen, P.C.Deedwania. (1997). Classification of heart failure patients using continuous chaotic modeling. Proceedings of the 18th World Congress on Medical Physics and Biomedical Engineering.

[6] J.-P. Eckmann, S.O.Kamphorst, D. Ruelle. (1987 November). Recurrence Plots of Dynamical Systems. Europhysics Letters. [Online]. 4, pp. 973-977.

[7] C. L. Webber Jr, J. P. Zbilut. (1994 February). Dynamical Assessment of Physiological Systems and States Using Recurrence Plot Strategies. Journal of Applied Physiology. [Online]. 76(2) pp. 965-973.

[8] Steven Hofmeyr, S.Forrest, A.Somayaji. (1998). Intrusion Detection using Sequences of System Calls. Journal of Computer Security. [Online], 6(3), pp.151-180.

[9] Steven Hofmeyr, Andrew Kosoresow. (1997 September). Intrusion Detection via System Call Tracing. IEEE Software, [Online]. 14(5), pp. 35-42.

[10] Anita Jones, Song Li. (2001 December). Temporal Signatures for Intrusion Detection. Presented at 17th Annual Computer Security Applications Conference. [Online].

[11] Joao Cabrera, Lundy Lewis, Raman Mehra. (2001 December). Detection and Classification of Intrusions and Faults using Sequences of System Calls. ACM SIGMOD Record. [Online]. 30(4), pp.25-34.

[12] N.Nguyen, P.Reiher, G.Kuenning. (2003 June). Detecting Insider Threats by Monitoring System Call Activity. Proceedings of IEEE Workshop on Information Assurance. [Online].

[13] Y.Qiao, X.W.Xin, Y.Bin, S.Ge. (2002 June). Anomaly Intrusion Detection based on HMM. Electronics Letters, [Online]. 38(13), pp. 663-664.

[14]. D. Mutz, F. Valeur, G.Vigna, C.Kruegel, Anomalous System Call Detection. ACM Transactions on Information and System Security, Feb 2006.

Multi-level Anomaly Detection with Application-Level Data

Swapna S. Gokhale and Jijun Lu
Department of Computer Science and Engineering
University of Connecticut, Storrs, CT 06269, USA
Email: {ssg, jijun.lu}@engr.uconn.edu

Abstract

Anomaly detection based on application-level data offers unique advantages over detection based on network-level and system-level data, since more meaningful information with higher quality and density is available at the application level. Detection based on application-level data can be used effectively in the context of general-purpose, distributed, component-based software applications only if the data used for detection is independent of the technology and platform used to implement the application. In this paper we suggest the use of performance metrics of an application as an example of technology-neutral and platform-independent application-level data for anomaly detection. Further, to reduce the false positives associated with anomaly detection, we propose the use of system-level data, namely, CPU usage, in conjunction with application-level data. We demonstrate the feasibility of our multi-level detection methodology on an experimental infrastructure comprising of a VoIP application.

1 Introduction

The penetration of information technology into our society has made our lives dependent on the services provided by component-based software applications. These applications are thus attractive targets of malicious attacks, which are presently the main threats against network and information security. Detection, which consists of rapidly identifying the occurrence of an attack is then crucial to mitigate the damage caused by an attack and to restore the services.

Existing intrusion detection systems can be characterized along two dimensions. Of these, the first dimension addresses the type of attacks (known vs. unknown) that can be detected. Misuse detection considers known attacks and comprises modeling of each known attack with a signature and then comparing the incoming activities with these signature patterns for detection. Anomaly detection considers both known and unknown attacks and consists of identifying substantial deviations from the specified normal behavior of users or applications. Anomaly detection is desirable compared to misuse detection because of its ability to detect previously unseen attacks. However, anomaly detection typically suffers from a very high false positive rate. The second dimension is concerned with the data used for detection, which includes network-level [3, 8], system-level [6] and application-level data [5, 10, 13]. Many systems base detection on multiple data types to reduce the false positive rate associated with anomaly detection [2, 14].

Detection based on network-level and system-level data is more prevalent compared to detection based on application-level data. However, since more meaningful information with higher quality and density is readily available at the application level, application-level data-based detection may be a valuable complement to network-level and system-level data-based detection especially in the context of general-purpose, distributed, component-based software applications. Current detection approaches based on application-level data, however, are not very effective for these applications because: (i) they use data that is specific to the technology and the platform used to implement the application, and (ii) they assume that the entire application resides only on one host.

In this paper we suggest the use of performance metrics of an application as an example of technology-neutral and platform-independent application-level data for anomaly detection. Further, to reduce the false positive rate of anomaly detection, we propose the use of system-level data consisting of CPU usage of the host(s) on which the application resides, in conjunction with application-level data. We illustrate the feasibility of our multi-level detection approach on an experimental infrastructure comprising of a VoIP application.

The paper is organized as follows: Section 2 motivates the use of application-level data for detection. Section 3 presents the multi-level detection methodology. Section 4 describes the experimental infrastructure. Section 5 discusses the emulation of baseline and abnormal scenarios. Section 6 discusses the experimental results. Conclusions and future directions are presented in Section 7.

2 Advantages of application-level data

The advantages of using application-level data over network-level and system-level data are as follows. First, semantic information is readily available from application-level data without any additional processing and this can aid in understanding the impact of an attack on the application services. Although semantic information is also present in network-level and system-level data, it is scattered and less structured, and needs to be extracted by additional processing which incurs overhead. This overhead can be mitigated by using application-level data and thus exploiting the processing of network-level and system-level data performed by the application to provide its services. Second, when an attack is launched by exploiting inherent vulnerabilities in an application, detection based on application-level data may also assist in identifying and isolating the cause of the attack. Third, detection based on application-level data provides a better possibility of guarding against illegitimate activities by insiders, or the insider threat [10].

Prevalent application-level data-based detection approaches suffer from the following limitations. First, the data used for detection is either generated by a standard application such as a web server, or is tied to the features of a specific programming language. Second, these techniques implicitly assume that the application completely functions within the address space of a single host, rather than being distributed on multiple hosts. Third, detection based on source code analysis can be used only if the source code is available. These drawbacks and assumptions make the existing approaches inadequate especially in the context of general purpose, distributed, component-based software applications. For such applications, components residing on multiple hosts routinely interact. Also, the components may be from different sources, and may be developed using different programming languages. Additionally, the source code may not be available for all the components.

The above discussion highlights the need to identify generic, technology-independent, and platform-neutral application-level data which could be used to detect attacks against general-purpose, distributed, component-based software applications. An example of such data is the performance metrics of an application. In the subsequent sections we demonstrate the feasibility of using performance metrics of an application for anomaly detection.

3 Anomaly detection methodology

In this section we describe the multi-level anomaly detection methodology. We focus on anomaly detection because of its potential to detect both known and unknown attacks. A central aspect of the methodology is the use of technology-neutral, platform-independent application-level data consisting of performance metrics for detection. Further, to reduce the false positive rate associated with anomaly detection, we use CPU usage data from the system level.

The two analysis steps involved in the methodology are described in the subsequent subsections.

3.1 Step I: Individual data analysis

In the first step, each type of data (performance metrics and CPU usage) are independently analyzed to generate an anomaly score. This analysis is based on the χ^2 test statistic and proceeds in the following manner.

The value of the χ^2 statistic, denoted ${\chi_i}^2$ is computed using the performance (CPU usage) data X_i for an application run i from Equation (1), where $\overline{X}$ is the mean signaling performance (CPU usage).

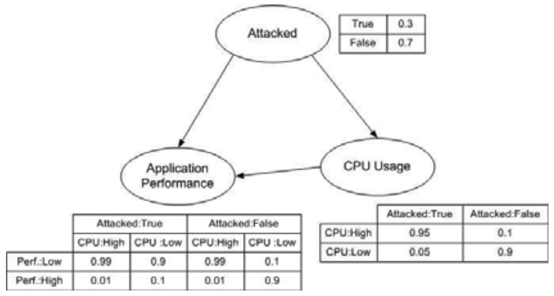
$$ {\chi_i}^2 = \frac{(X_i - \overline{X})^2}{\overline{X}} \tag{1} $$

Using the values of the χ^2 statistic for each measurement, the expected value of the statistic, $E[\chi^2]$ is computed using Equation (2). The expected value is then compared with the preset threshold to generate an anomaly score. The number of measurements, n, used to compute the expected value is determined using the sliding window protocol [4].

$$ E[\chi^2] = \sum_{i=1}^{n} \frac{{\chi_i}^2}{n} \tag{2} $$

3.2 Step II: Correlated data analysis

In the second step, we use the Bayesian network (BN) [9] shown in Figure 1 to correlate the anomaly scores obtained from the independent analysis of the two data types in the first step and infer the posterior probability of an attack.



True	0.3
False	0.7

	Attacked:True		Attacked:False	
	CPU:High	CPU :Low	CPU:High	CPU :Low
Perf.:Low	0.99	0.9	0.99	0.1
Perf.:High	0.01	0.1	0.01	0.9

	Attacked:True	Attacked:False
CPU:High	0.95	0.1
CPU:Low	0.05	0.9

Figure 1. BN for multi-level detection

In Figure 1, the two information nodes represent the anomaly scores obtained from the analysis of application performance and CPU usage data respectively. A causal relationship exists between these two information nodes, since high CPU usage will result in lower performance. This relationship is depicted by an arrow from the CPU usage node to the performance node. Each node is associated with a conditional probability table, obtained by discretizing the anomaly scores, to map a continuous variable to a discrete one. The discrete variable takes two values, namely, high and low, depending on whether the score exceeds or is below a specified threshold. The hypothesis node is a classification that determines whether the application behavior is anomalous during an interval.

4 Experimental testbed

In this section we describe the VoIP infrastructure used for the demonstration of our methodology.

VoIP (IP telephony) is the transport of voice traffic by using the Internet Protocol (IP), rather than the public switched telephone network. Session Initiation Protocol (SIP) [12] is a popular signaling protocol used for VoIP. The peers in a SIP-based session[1] are called user agents (UAs). Figure 2 shows a typical SIP message flow between UAs.

In Figure 2, UA1 initiates the session by sending an INVITE request to UA2. After the INVITE/OK/ACK three-way handshake [12], the session is established and the two UAs begin to exchange data. At the end of the data exchange, UA2 sends a BYE request to UA1, which causes this session to be closed bi-directionally. *Signaling performance* used in this paper is defined as the time taken to set up a session between the two VoIP peers, which is the time between the instant the caller sends out "INVITE" and the instant it receives the "180 Ringing" message.

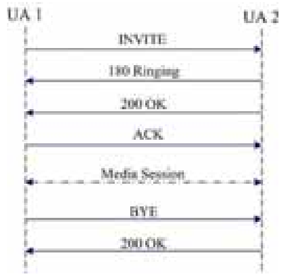


Figure 2. A typical SIP message flow

The experimental infrastructure consists of a public-domain implementation of a VoIP application [11]. The hardware platform consists of: (i) a Dell OptiPlex GX260 (Intel P4 2.4GHz, 1GB of RAM), and (ii) an IBM ThinkPad T40 (Intel P-M 1.5GHz, 512MB of RAM). These are installed with Windows XP Professional SP2 and are connected via a 100M Ethernet connection across a LAN.

[1]The terms call and session are used interchangeably in this paper.

5 Normal and anomalous scenarios

In this section, we describe the emulation of normal and abnormal scenarios for the VoIP infrastructure.

Since the signaling performance is governed by the number of simultaneous sessions between the two UAs, the baseline behavior of the infrastructure is obtained by computing the average signaling performance over a number of incoming session requests when a small number of simultaneous sessions are ongoing. The number of sessions should be low so that the baseline performance would be acceptable under normal conditions. The average CPU usage under these conditions would also provide the baseline.

A number of simultaneous sessions can be sustained in the following manner. UA1 is provided: (i) Distribution of the interarrival time of calls and its parameters, and (ii) Distribution of the call holding time and its parameters. Using the interarrival time distribution, UA1 generates a series of interarrival times $\{t_i\}, i = 1, 2, ...$ Using $\{t_i\}$s, the arrival time of each call $\{S_i\}$ is computed by $S_i = \sum_{j=1}^{i} t_j$. Using the holding time distribution, UA1 also generates the holding time of each call $\{d_i\}$. Then the ending time of each call is computed by $T_i = S_i + d_i$. At each arrival time S_i, UA1 automatically sends an INVITE request to UA2 to initiate a call. This session is held for a duration d_i. At the ending time T_i, UA1 sends a BYE request to UA2 to terminate the call. To sustain a number of simultaneous sessions, the mean holding duration $\bar{d_i}$ is chosen to be higher than the mean interarrival time $\bar{t_i}$. The expected number of simultaneous sessions is approximated as the ratio of $\bar{d_i}$ and $\bar{t_i}$. The baseline signaling performance obtained by setting $\bar{t_i} = 30$ sec., and $\bar{d_i} = 30$ sec. is 15.82 msec.

Abnormal conditions are emulated by increasing the number of simultaneous sessions over what was used to obtain the baseline performance. Thus, abnormal conditions constitute a Denial of Service (DoS) type attack [1]. We consider DoS attacks, as they are the leading cause of financial loss due to cybercrime [7]. The number of simultaneous sessions can be increased over the norm by: (i) reducing the mean interarrival time $\bar{t_i}$, or/and (ii) increasing the mean holding duration $\bar{d_i}$. Abnormal conditions were emulated by using different combinations of $\bar{t_i}$ and $\bar{d_i}$.

The CPU usage on the server machine was also logged each time signaling performance was measured, resulting in a one-to-one correspondence between the two data types. The baseline CPU usage under low-load conditions was 11.41%.

6 Results and discussion

In this section we describe the experiments and summarize and discuss their results.

6.1 Experiment set I

In the first set, a mix of abnormal and normal operating conditions was emulated by dividing each experiment into four phases. The parameters for these phases are selected so that the average number of simultaneous sessions in phases 1 and 3, and in phases 2 and 4 are identical. Eight experiments with parameters in Table 1 are conducted in this set. Referring to the table, for example, in the first experiment, the mean holding duration $\bar{d_i}$ is set to 120 sec. for all the phases, and the mean interarrival time $\bar{t_i}$ is set to 30 sec. for phases 1 and 3 and 10 sec. for phases 2 and 4. In each phase, 40 calls are initiated by UA1. A complete log file for all the four phases (a total of 160 calls) is generated.

Table 1. Parameters of experiment set I

No.	Interarrival time and Duration$(\bar{t_i}, \bar{d_i})$ (s)			
(♯)	Phase 1	Phase 2	Phase 3	Phase 4
1	(30, 120)	(10, 120)	(30, 120)	(10, 120)
2	(30, 180)	(10, 180)	(30, 180)	(10, 180)
3	(30, 240)	(10, 240)	(30, 240)	(10, 240)
4	(30, 300)	(10, 300)	(30, 300)	(10, 300)
5	(30, 120)	(15, 120)	(30, 120)	(15, 120)
6	(30, 180)	(15, 180)	(30, 180)	(15, 180)
7	(30, 240)	(15, 240)	(30, 240)	(15, 240)
8	(30, 300)	(15, 300)	(30, 300)	(15, 300)

This continuous log was processed as per the sliding window method [4], with a window size of 20. This resulted in 141 windows over 160 calls. We note that two competing concerns, namely, the possibility of false positives and false negatives (missing actual attacks) need to be balanced while choosing the window size. The window size is set to 20 considering the total number of calls in each experiment.

Expected values of the χ^2 statistic were computed for each window for each type of data. The anomaly scores are obtained by comparing the expected χ^2 statistic with preset thresholds. For illustrative purposes, we consider the two threshold sets in Table 2. These scores are then processed through the BN to infer the posterior attack probabilities.

Table 2. Thresholds for anomaly detection

Set	Sig. perf.	CPU usage
I	15	300
II	30	500

The expected value of the χ^2 statistic and posterior attack probabilities are shown for experiments 1 and 4 in Figure 3. The remaining results are not included due to space limitations. The figure indicates that within each experiment there are periods of increasing scores, which coincide with transitions among phases 1 and 2 and phases 3 and 4. Similarly, the period of decreasing scores coincides with the transition among phases 2 and 3. Table 1 indicates that the holding duration is higher in phases 2 and 4 than in phases 1 and 3. Thus, the number of ongoing calls in phases 2 and 4 is higher, resulting in a lower signaling performance and higher CPU usage. Across experiments, it can be seen that the peak value of the anomaly score is the highest for experiment 4, because the mean holding duration in phases 2 and 4 is higher for experiment 4.

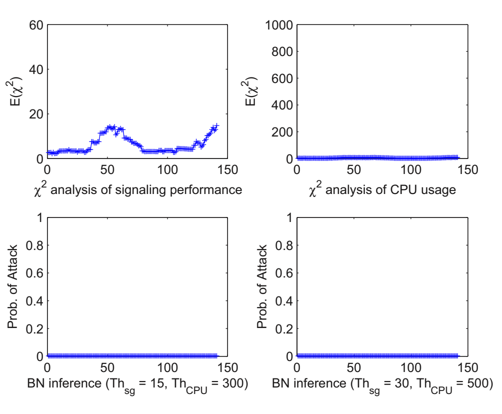
Next we discuss the posterior attack probabilities produced by the BN. For experiment 1, the scores generated from both data types are below their respective thresholds for both sets. The BN thus infers a very low (close to 0) posterior attack probability as can be seen in the bottom plots in Figure 3(a). For experiment 4, both signaling performance and CPU usage exceed their thresholds around the peak under both sets. This results in high posterior attack probabilities around the peaks as can be seen in the bottom figures in Figure 3(b). These figures, however, indicate that high attack probabilities are inferred during narrower periods under the second set of thresholds (which are larger) as compared to the periods under the first set.

These results indicate that only when both signaling delay and CPU usage are high, the BN infers a high posterior probability of attack. Thus, the thresholds set on signaling delay and CPU usage will impact the sensitivity of detection. For lower thresholds, the chance of false positives is higher. However, for larger thresholds, the detection engine may miss some attacks and the chance of false negatives increases. The choice of the thresholds will be governed by the level of attack tolerance, which will be depend on the criticality of the application services. For applications which provide critical services such as emergency response, the level of attack tolerance is smaller, and hence the thresholds should be set lower. On the other hand, for streaming media applications, the level of attack tolerance may be higher, due to which coarser thresholds may suffice.

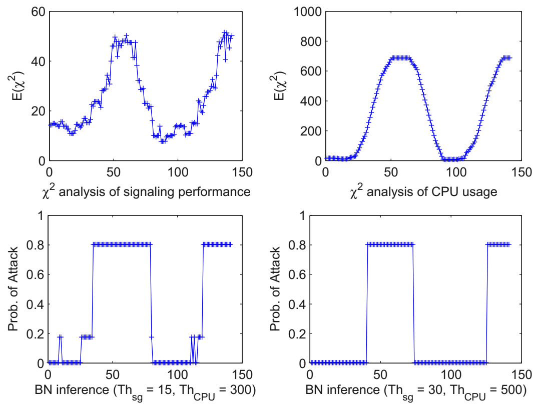
6.2 Experiment set II

In the second set, false alarm scenarios were emulated as follows. After receiving an “INVITE” request, the callee delays the response for a certain period. A delay such as this may be caused by network conditions, or due to the VoIP implementation of the callee’s provider. It is not due to an attack resulting in a CPU intensive process on the callee’s side. However, the ultimate outcome, namely, high signaling delay will occur despite the lack of an attack.

Four experiments, with the parameters in Table 3 were conducted. In the table, for example, in the first experiment,



(a) Experiment 1



(b) Experiment 4

Figure 3. Anomaly scores and posterior attack probabilities (Experiment set I)

the mean interarrival and holding times, $\overline{t}_i$ and $\overline{d}_i$, are set to 15 and 120 sec. After receiving a connection request, the callee delays the response for a wait time t_w of 20 msec.

Table 3. Parameters of experiment set II

No. ($\sharp$)	Interarr. $\overline{t}_i$ (s)	Duration $\overline{d}_i$ (s)	WaitTime t_w (ms)
1	15	120	20
2	15	120	50
3	30	120	20
4	30	120	50

Similar to the first experiment, 160 calls with a sliding window of 20 generated 141 anomaly scores. Figure 4 shows the expected value of the χ^2 statistic for performance and CPU measurements along with the posterior attack probabilities for experiments 1 and 4.

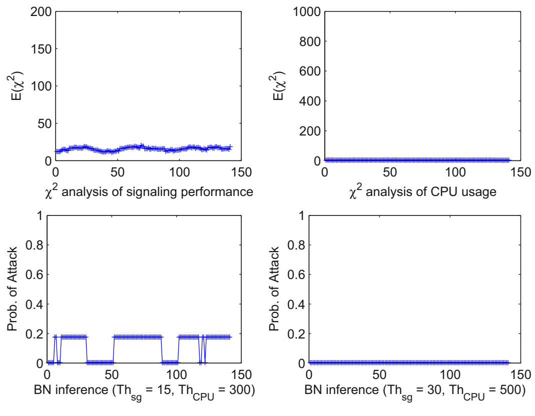
The results indicate that for experiment 1, the anomaly scores generated from CPU usage are below their thresholds for both sets, while the scores for signaling delay fluctuate around the threshold in the first set and are clearly lower than the threshold in the second set. Thus, the BN intermittently infers non-zero posterior attack probability with a peak value of 0.176 under the first set as shown in Figure 4(a). Under the second set, the attack probabilities inferred are close to zero, as shown in Figure 4(a). In experiment 4, the anomaly scores generated from CPU usage remain below the thresholds for both sets. However, the signaling delay exceeds the thresholds under both sets. This is because the waiting time in experiment 4 is higher compared to experiment 1 (Table 3). Despite the fact that the anomaly scores for signaling delay exceed their thresholds, the posterior attack probability inferred by the BN remains low as can be seen in the bottom plots in Figure 4(b). Comparing the BN results from experiments 1 and 4, it can be seen that the posterior attack probabilities in experiment 1 fluctuate between zero and non zero values while the attack probabilities hold steady above zero in experiment 4. Further, the BN accurately reflects the fact that the degree of abnormality is greater in experiment 4 than in experiment 1, by inferring higher and steady non zero posterior attack probabilities for experiment 4.

These results indicate that the BN effectively rules out the possibility of a DoS attack if high signaling delay is not caused by a corresponding high CPU usage. Thus, they demonstrate how detection based on data from multiple levels may help in reducing the incidence of false alarms.

7 Conclusions and future research

In this paper we discussed the need for using application-level data for anomaly detection as a complement to network-level and system-level data, especially in the context of general-purpose, component-based, distributed software applications. We suggested the use of performance metrics of an application as an example of technology-neutral, platform-independent, application-level data for anomaly detection. Further, to reduce the false positive rate associated with anomaly detection, we proposed the use of system-level data, namely, the CPU usage of the host on which the application resides. We demonstrated the feasibility of our multi-level anomaly detection methodology on an experimental infrastructure of a VoIP application.

Our future research consists of: (i) identifying other types of generic application-level data and demonstrating the feasibility of their use for anomaly detection, and (ii) using performance metrics to detect other types of attacks.


(a) Experiment 1

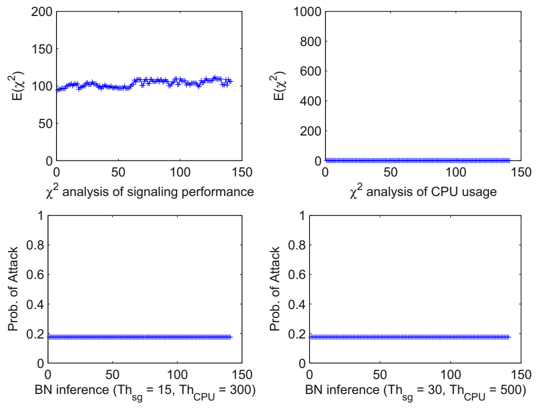

(b) Experiment 4

Figure 4. Anomaly scores and posterior attack probabilities (Experiment set II)

Acknowledgments

This research was supported by the National Science Foundation (NSF) under the following grants: CNS-0406376 and CNS-SMA-0509271.

References

[1] Denial-of-service attack. `http://en.wikipedia.org/wiki/Denial-of-service_attack`, Feburary 2007.

[2] C. Abad, J. Taylor, C. Sengul, W. Yurcik, Y. Zhou, and K. Rowe. Log correlation for intrusion detection: A proof of concept. In *Proc. 19th Annual Computer Security Applications Conference (ACSAC'03)*, pages 255–264, 2003.

[3] D. Bauer, J. Cannady, and R. C. Garcia. Detecting anomalous behavior: Optimization of network traffic parameters via an evolution strategy. In *Proc. SoutheastCon*, 2001.

[4] M. Burgess, H. Haugerud, , S. Straumsnes, and T. Reitan. Measuring system normality. *ACM Trans. on Computer Systems*, 20(2):125–160, May 2002.

[5] J. B. D. Cabrera, B. Ravichandran, and R. K. Mehra. Statistical traffic modeling for network intrusion detection. In *Proc. 8th Int'l Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 466–473, 2000.

[6] D. Endler. Intrusion detection appplying machine learning to Solaris audit data. In *14th Annual conference on Computer Security Applications*, 1998.

[7] L. A. Gordon, M. P. Loeb, W. Lucyshyn, and R. Richardson. 2004 CSI/FBI computer crime and security survey, June 2004. `http://www.gocsi.com/`.

[8] M. Iguchi and S. Goto. Network surveillance for detecting intrusions. In *Proc. Internet Workshop*, 1999.

[9] F. Jensen. *Bayesian networks and decision graphs*. Springer, New York, 2001.

[10] A. K. Jones and Y. Liu. Application intrusion detection using language library calls. In *17th Annual Computer Security Applications Conference (ACSAC'01)*, pages 442–449, 2001.

[11] National Institute of Standards and Technology (NIST). JAIN-SIP project home. `https://jain-sip.dev.java.net/`.

[12] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol, July 2002. RFC 3261.

[13] R. Sion, M. Atallah, and S. Prabhakar. On-the-fly intrusion detection for Web portals. In *Proc. Int'l Conference on Information Technology: Computers and Communications (ITCC'03)*, pages 325–330, 2003.

[14] Y.-S. Wu, B. Foo, Y. Mei, and S. Bagchi. Collaborative intrusion detection system (CIDS): a framework for accurate and efficient IDS. In *Proc. 19th Annual Computer Security Applications Conference (ACSAC'03)*, 2003. 234-244.

A Four-layered Semantic Grid Architecture

Célia Ghedini Ralha, José Nelson C. Allemand and Alba C. M. Melo
Departamento de Ciência da Computação - Instituto de Ciências Exatas
Universidade de Brasília - Campus Universitário Darcy Ribeiro
Caixa Postal 4466 - Brasília - Cep 70.910-900
{ghedini,josenelson,albamm}@cic.unb.br

Abstract

In this paper, we present the design and implementation of a Semantic Grid architecture composed by a knowledge layer to semantically enhance the grid resource discovery service. The knowledge layer has a semantic repository, which enables semantic discovery of different types of computing resources in grid environment. This layer is formed by an ontology and a reasoner component to interact with the semantic repository. Our prototype uses existing technologies, like the Globus Toolkit's MDS with Ganglia, to illustrate our four-layered decoupled Semantic Grid architecture.

1. Introduction

Grid Computing is an emerging technology for enabling resource sharing and coordinated problem solving in dynamic multi-institutional virtual organizations [24, 26, 25]. Grids are used to join various geographically distributed computational and data resources, and deliver these resources to heterogeneous user communities. These resources may be described in distinct manners since they can belong to different institutions, have different usage policies and pose different requirements on acceptable requests. In summary, this complex environment scenery will greatly benefit from the Semantic Grid [1].

In the Semantic Grid, information and services are given a well defined meaning, better enabling people and computers to work in cooperation [35, 33, 36]. In such an environment, the ability to describe the grid resources needed by agents or applications is essential for developing seamless access to resources on the grid [19]. This environment has a fundamental task named resource matching, which involves assigning resources to tasks in order to satisfy task requirements and resource policies. These requirements and policies are often expressed in disjoint application and different resource models, forcing a resource selector to perform semantic matching among them. In addition, as the grid environment is very dynamic, it is desirable and sometimes necessary to automate the resource matching to robustly meet the application requirements.

There are many grid research initiatives to help on this challenge. Unfortunately, the most common Globus Toolkit Monitoring and Discovery System (MDS) [2] supports only traditional resource matching, which is based on symmetric, attribute based matching and does not support semantic description of grid resources or services [16, 41]. This approach is clearly not sufficient in grid environments where resources are generally owned by different communities, with varied administration policies and capabilities, where obtaining and managing these resources is not trivial.

In this paper, we present the design and the implementation of a prototype of a Semantic Grid architecture with a knowledge layer for semantic description and discovery of resources. The knowledge layer is formed by an ontology component and a reasoner one, which interact to each other through the semantic repository. With the proposed architectural framework we allow user or agent to select resources appropriately to the requirements of the application based on different constraints and with specific capabilities.

The rest of the paper is organized as follows. In Section 2, we discuss related work. The proposed Semantic Grid architecture is presented in Section 3, focusing on the knowledge layer. Section 4 describes aspects related to the implementation work. Some experimental results are shown in Section 5. Section 6 concludes the paper discussing about future research work.

2. Related Work

The literature addresses the challenge of Semantic Grid through different approaches. In practice, work on Semantic Grid has primarily meant introducing technologies from the Semantic Web to the Grid. The background knowledge and vocabulary of a domain can be captured in ontologies, that are machine processable models of concepts, their interrelationships and constraints [30]. According to [27], the Semantic Grid lacks a reference architecture or any kind of systematic framework for designing Semantic Grid components or applications. There are a number of recent efforts that we can cite and compare to our work [38, 31, 19, 40, 17, 37].

[38] have designed and prototyped an ontology-based resource selector that exploits ontologies, background knowledge and rules for solving resource matching in the grid. The authors used the Protégé editor to manually create ontology instances [3]. They have developed a prototype matchmaker to support the matchmaking service which did not include the brokering service at the time of writing the paper.

In [31], the authors describe the architecture of the Ontology-based Resource Matchmaker Service (OMMS). OMMS provides dynamic access to matchmaking capability building an online matchmaking service. Their implementation uses the Globus Toolkit for the grid service development, and exploits the monitoring and discovery service in the grid infrastructure to dynamically discover and update resource information.

[19] considered the problem of resource description in the context of a resource broker at the GRIP Project, with GT2, GT3 and Unicore middleware systems. They have shown how the semantic approach, to resource description, facilitates the current problem of not existing a common standard to make the grid transparent at the application level.

[40] presented the Core Grid Ontology (CGO) to provide a common basis for representing grid knowledge and grid systems, including basic concepts and relationships of grid entities and grid resources according to a proposed abstract grid model. The CGO ontology provides a common basis for representing grid knowledge and grid systems; it is used for grid information integration and searching, resource discovery and allocation management. The CGO was described using the OWL language [20], which supports the realization of the Semantic Grid including grid resources, grid middlewares, services, applications and users. As future work the authors intend to support ontology/knowledge queries related to multi-grid environments for what a suitable OWL query language and distributed mechanism to query needs to be developed.

In [17], the authors describe the Semantic-OGSA (S-OGSA), which is one of the early results of the EU-IST project OntoGrid [4]. S-OGSA is proposed as a reference architecture for the project and has been created with the aim to become also a reference framework for the Semantic Grid [28].

In our work, we have designed and prototyped a Semantic Grid architecture based on an ontology template and rules for intelligent resource matching in the grid. Similar to [38] we have not included the brokering service at our architecture, since a decoupled approach allows the use of different middlewares through an XML format resource repository. We haven't focused our work at the ontology definition, since it can be extended later; but we have handled the reasoner component to allow semantic queries related to multi-grid environments. In relation to the Semantic Grid architectural framework our aim is not to propose a reference architecture such as [17] with the S-OGSA, but to design an architecture able to deal with semantic resource discovery matching service. Our prototype uses Globus Toolkit 4 [23] for the grid service development, and exploits the monitoring and discovery service in the grid infrastructure to dynamically discover and update resource information like OMMS [31].

The work we consider closer to our proposal is [37]. The authors have proposed a Semantic Grid architecture by introducing a knowledge layer at the top of the Gridbus broker architecture [5]. The semantic component in the knowledge layer enables semantic description of grid resources with the help of an ontology template [39]. The difference is that their proposed five layered architecture is built on top of the Gridbus broker, while ours follows a decoupled approach not allied to a specific broker. Once our resource repository is an XML file, it allows different brokers to be integrated to the Semantic architecture. However, the prototypes of both works have been implemented using MDS component, being Globus dependent.

3. The Proposed Architecture

For the Semantic Grid infrastructure, we propose a four layered architecture that implements a knowledge layer on top of the middleware layer as shown in Figure 1. In this architecture, the fabric layer provides the computational and network resources, storage systems, logical entities such as distributed storage device to which shared access is mediated by grid protocols.

The middleware layer provides a secure and unified access to remote resources and one can choose from different middlewares such as Globus [6], Unicore [7], Alchemi [8], among others. In our Java prototype this layer was implemented using Globus Toolkit Version 4 [2, 22]. We also used the Ganglia MDS information provider written in Perl [9]. The output of the Ganglia information that gets published into MDS is determined by the GLUE-CE Schema (an abstract modeling for Grid resources and mapping to concrete schema that can be used in Grid Information Services [10]). At the middleware layer, the grid resource data captured by the Ganglia information provider is stored at the resources repository in XML format.

The knowledge layer provides semantic resource discovery service from the huge amount of data aggregated

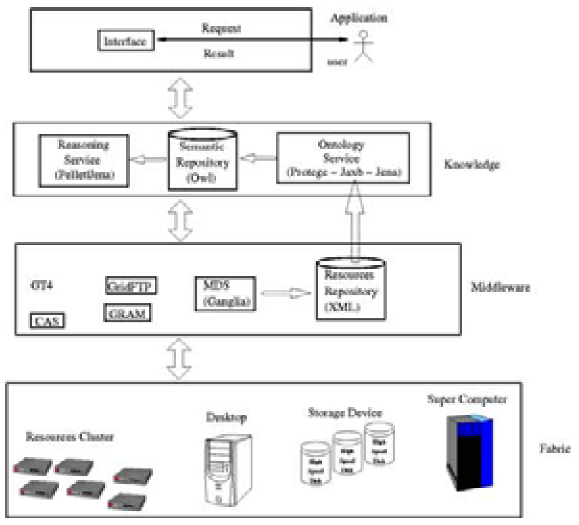


Figure 1. The four-layered Semantic Grid architecture.

from underlying information services layer stored at the resources repository (XML). We used Jena version 2.4 [11], a Java framework for building Semantic Web applications, to capture the grid resource information in XML and to transfer to the semantic repository in the Web ontology language OWL [20, 34]. Jena's architecture provides a mechanism for attaching external reasoners to Jena models. The knowledge layer is the focus of this work since it is responsible to store, represent and infer on semantic information of grid resources. There is also an application layer to enable the use of resources in the grid environment.

3.1. The Knowledge Layer

For the Semantic Grid environment, users and software agents should be able to discover grid nodes offering required services and having particular properties. For that, we propose the knowledge layer, which is formed by the ontology and reasoner components. The following paragraphs describe the implemented components of this layer.

The Ontology Component

An ontology is a specification of a conceptualization [29]. In this context, specification refers to an explicit representation by some syntactic means. In contrast to schema languages (like XML Schema), ontologies try to capture the semantics of a domain by deploying knowledge representation primitives, enabling a machine to partially understand the relationships between concepts in a domain. Additional knowledge can be captured by axioms or rules. In the Web context, OWL [20, 34] and RDF-Schema [12] are recommendations from the W3C for ontology modeling languages.

For the ontology component we used Protégé [3] to semantically deal with grid resources. The Protégé editor offers versatile libraries or Protégé-OWL APIs to perform several operations over the ontology: creating and deleting instances of concepts, assigning values to the properties, among others. For every type of information retrieved from the grid node, we create instances of appropriate concept in the ontology template conforming to respective concept types. In such a way, an instance will be exactly related to only one concept type and the values of various properties retrieved are assigned to respective properties of the appropriate concepts in the ontology template.

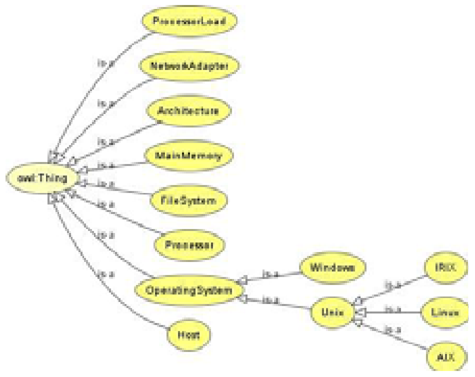


Figure 2. A small ontology template.

In this work, ontology template is defined as a grid resource ontology that provides an hierarchy of concepts along with properties to define their characteristics. Figure 2 shows a small ontology template with concept hierarchy considered in our experiments. The values of the properties considered to define concepts are retrieved from MDS of Globus Middleware automatically with Glue/Ganglia forming the semantic repository of the grid resources (XML). This repository is updated periodically so that addition or removal of resources is accounted in the semantic repository automatically.

The Reasoner Component

Our reasoner component of the semantic discovery service relies on the power of Pellet inference engine, an open-source Java based and OWL DL [21, 13]. Pellet can be used in conjunction with both Jena and OWL API libraries. It is based on the tableaux algorithms developed for expressive Description Logics (DL) [32, 18]. Pellet supports the full expressivity OWL DL including reasoning about nominals (enumerated classes). Therefore, OWL constructs `owl:oneOf` and `owl:hasValue` can be used freely. The Pellet API provides functionalities to see the species validation, check consistency of ontologies, classify the taxonomy, check entailments and answer a subset of RDQL queries. Currently, our prototype supports the following operators: $=$, $>$, $<$, $>=$, $<=$ and $\neq$. Also, the query mechanism is designed to support query with single or multiple constraints.

4. Implementation Aspects

To validate the Semantic Grid architecture for the grid resource discovery service, we implemented a prototype using the Globus Toolkit 4.0.3 [22] at the University of Brasília. Figure 3 represents the modules implemented in the knowledge layer including the ontology and the reasoning services. At the Ontology Service module, we first define the ontology template (with Protégé 3.1.1 build 284) using some of the defined GLUE Schema 1.1 classes related to the Ganglia 3.0.3 monitoring system [14]. At this point, the ontology template with concepts and properties constitutes the Semantic Repository in OWL language (not yet bounded to any grid resources).

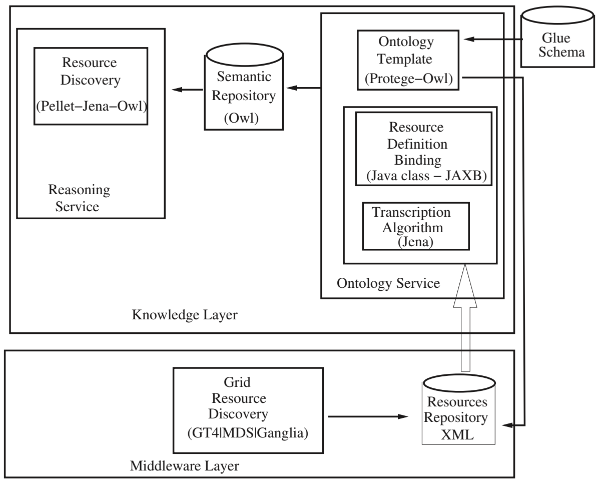

Resource Discovery (Pellet−Jena−Owl)
Semantic Repository (Owl)
Ontology Template (Protege−Owl)
Glue Schema
Resource Definition Binding (Java class − JAXB)
Transcription Algorithm (Jena)
Reasoning Service
Ontology Service
Knowledge Layer
Grid Resource Discovery (GT4|MDS|Ganglia)
Resources Repository XML
Middleware Layer

Figure 3. Implementation of the knowledge layer.

Once defined the ontology template, the resource monitoring discovery service (using Ganglia) automatically saves information about the real resources of the grid environment at a specific time. At this point, the Resource Repository (XML) has the information provided by the MDS/GT4 middleware service. We have tested our prototype on a grid environment with twenty-two machines located in two different laboratories as shown in Table 1.

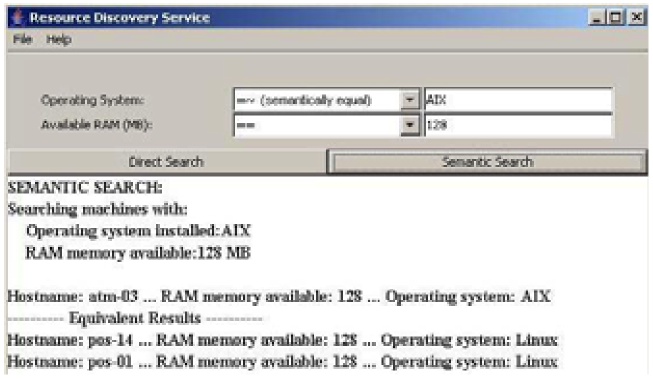
In order to instantiate the ontology template we had first to bind the grid resources defined in XML to Java classes using Java Architecture for XML Binding (JAXB), which provides a convenient way to bind an XML schema to a representation in Java code and is available in the Java Web Services Developer Pack (Java WSDP) 1.5 [15]. Afterwards, we had to develop a transcription algorithm to read the resources from the Resources Repository (XML) to instantiate the Java classes (JAXB) which allow to save at the Semantic Repository the instantiated ontology template (OWL) with Jena 2.4 framework [11].

Table 1. Experimental grid environment.

Host	O. S.	Available RAM
pos-01.cic.unb.br	Linux	128
pos-02.cic.unb.br	Windows	332
pos-03.cic.unb.br	Linux	123
pos-04.cic.unb.br	Windows	333
pos-05.cic.unb.br	Linux	12
pos-06.cic.unb.br	Linux	210
pos-07.cic.unb.br	Linux	87
pos-08.cic.unb.br	Windows	123
pos-09.cic.unb.br	Windows	128
pos-10.cic.unb.br	Linux	7
pos-11.cic.unb.br	Windows	8
pos-12.cic.unb.br	Linux	66
pos-13.cic.unb.br	Windows	44
pos-14.cic.unb.br	Linux	128
pos-15.cic.unb.br	Linux	122
pos-20.cic.unb.br	Windows	12
atm-01.cic.unb.br	AIX	888
atm-02.cic.unb.br	AIX	243
atm-03.cic.unb.br	AIX	128
carbona.laico.cic.unb.br	Solaris	568
fau.laico.cic.unb.br	FreeBSD	354
magicien.laico.cic.unb.br	IRIX	564

At the Reasoning Service part we used the instantiated Semantic Repository with Jena framework 2.4 and Pellet 1.3 reasoner for semantic retrieval of grid resource information. According to the user queries, the reasoner will provide the best results to direct resource matching and semantic matching.

For the application layer, we developed an interface to validate the proposal. The screenshot of the resource discovery service is shown in Figure 4. Note that we have developed two search options - direct and semantic; and illustrated with two machine resources - Operating system installed and available RAM memory (MB). Figure 4 also presents a semantic search query with multiple resources including `?operatingSystemName=AIX` and `?ramAvailable=128`. The presented results demonstrate atm-03 as a best match and semantically equivalent results with pos-14 and pos-01, which have 128 MB and are Unix compatible. Note that our prototype illustration uses only two different machine resources related to the concepts of the ontology template of figure 2. In our implementation, the ontology template depends on the MDS components as we are using Globus with Ganglia resource information, but an extended ontology template can be used to include con-



```
Operating System:    =~ (semantically equal)    AIX
Available RAM (MB):  ==                         128
```

```
SEMANTIC SEARCH:
Searching machines with:
   Operating system installed:AIX
   RAM memory available:128 MB

Hostname: atm-03 ... RAM memory available: 128 ... Operating system: AIX
---------- Equivalent Results ----------
Hostname: pos-14 ... RAM memory available: 128 ... Operating system: Linux
Hostname: pos-01 ... RAM memory available: 128 ... Operating system: Linux
```

Figure 4. Resource Discovery Service interface.

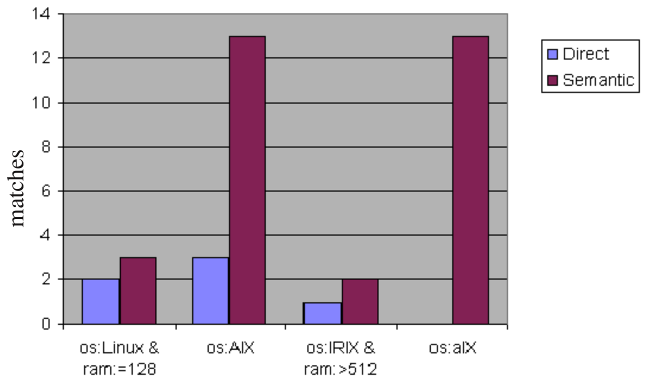


Figure 5. Direct vs Semantic Results.

cepts according to the desired management purposes.

5. Experimental Results

In this paper, we propose a flexible and extensible approach for performing Grid resource selection using an ontology-based matchmaker at a Semantic Grid architecture. Unlike the traditional Grid resource discovery service that describes resource/request properties based on symmetric flat attributes, separate ontologies (i.e., semantic descriptions of domain models) are created to declaratively describe resources requests using an expressive ontology language. Instead of exact syntax matching, our ontology-based matchmaker performs semantic matching using terms defined in the ontology template.

Figure 5 shows the evaluation of the proposed Semantic-based Grid resource discovery service. For the experiments, we executed several direct and semantic search queries, including single and multiple properties of particular resources, e.g., Operating system and available RAM. We estimated the performance considering 'hit' and 'miss' while searching for the resources. With the direct search, whenever the user asks for a resource which is unknown to the Semantic Repository, an error message is given (this is considered a 'miss' and is shown in Figure 5 as zero matches). With the semantic search, the most closely related resource is presented to the user. Note that with the direct search and multiple machine resources (e.g., OS:Linux and RAM:=128), we have two machines pos-01 and pos-14 (see Table 1). Also note that with the semantic search we obtain three machines (pos-01, pos-14 and atm-03), since atm-03 is Unix compatible.

Considering the single resource with the direct search "OS:AIX", three machines were retrieved and with semantic search we obtained thirteen since nine are Linux, three are AIX and one is IRIX, all Unix compatible. Note that in Table 1, there are two other machines which are Unix compatible (carbona.laico.cic.unb.br with Solaris and fau.laico.cic.unb.br with FreeBSD), but they were not retrieved by our semantic resource discovery service (Figure 5) since Solaris and FreeBSD are not included in the ontology template (Figure 2). Finally, consider the direct search like OS:aIX which has no direct search results, but as the prototype treats case the same thirteen machines were found with the semantic search.

6. Conclusion

Our proposal focuses on the integration of the semantic technology for grid resource discovery with MDS through a Semantic Grid architecture. It is common knowledge that the use of a Semantic Grid architecture can make easier the deployment of complex applications, in which several organizations are involved and diverse resources are shared.

The proposed architecture was prototyped with the Grid Information Service of Globus Toolkit 4 to aggregate resource information and automatically create the semantic repository using a resource ontology template. The small ontology template used in our experimental work shows that more accurate results are possible than conventional approaches. For future work, the authors intend to develop a suitable query language to support ontology/knowledge related to multi-grid environments and extend the reasoner component to enhance semantic treatment with multi-grid Information Services.

References

[1] `http://www.semanticgrid.org/`.
[2] `http://www.globus.org/toolkit/mds`.
[3] `http://protege.stanford.edu/`.
[4] `http://www.ontogrid.net/`.
[5] `http://www.gridbus.org/`.

[6] http://www.globus.org/.
[7] http://www.unicore.org/.
[8] http://www.gridbus.org/~alchemi/.
[9] http://www.star.bnl.gov/STAR/comp/Grid/Monitoring/SimpleGangliaIP.html.
[10] http://glueschema.forge.cnaf.infn.it/.
[11] http://jena.sourceforge.net/.
[12] http://www.w3.org/2001/sw/WebOnt/.
[13] http://www.mindswap.org/2003/pellet/.
[14] http://ganglia.sourceforge.net/.
[15] http://java.sun.com/xml/downloads/jaxb.html.
[16] R. Akkiraju, R. Goodwin, P. Doshi, and S. Roeder. A method for semantically enhancing the service discovery capabilities of uddi. In *Proceedings of IJCAI - Workshop of Information Integration on the Web*, Acapulco, Mexico, August 2003.
[17] P. Alper, O. Corcho, I. Kotsiopoulos, P. Missier, S. Bechhofer, D. Kuo, and C. Goble. S-ogsa as a reference architecture for ontogrid and for the semantic grid. In *Proceedings of the 3^{rd} GGF Semantic Grid Workshop (GGF16)*, Macedonia, Feb. 2006.
[18] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge Press, 2003. http://www.cambridge.org/uk/0521781760.
[19] J. Brooke, D. Fellows, K. Garwood, and C. Goble. Semantic matching of grid resource description. In *Grid Computing - LNCS*, volume 3165, pages 240–249. Springer-Verlag, Berlin/Heidelberg, 2004. ISBN 978-3-540-22888-2.
[20] M. Dean and G. Schreiber. Owl web ontology language reference w3c recommendation. http://www.w3.org/tr/owl-ref/.
[21] B. P. E. Sirin. Pellet: An owl dl reasoner. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2004. http://www.mindswap.org/2003/pellet.
[22] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In *IFIP International Conference on Network and Parallel Computing - LNCS*, volume 3779, pages 2–13. Springer-Verlag, 2006.
[23] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.
[24] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufman, San Francisco, USA, 1999.
[25] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Grid services for distributed system integration. *Computer*, 35(6), 2002.
[26] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001. http://www.globus.org/research/papers/anatomy.pdf.
[27] C. Goble. Towards a semantic grid architecture. In C. Goble, C. Kesselman, and Y. Sure, editors, *Semantic Grid: The Convergence of Technologies*, number 05271 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany.
[28] C. Goble, I. Kotsiopoulos, O. Corcho, P. Alper, and S. Bechhofer. An overview of s-ogsa: a reference architecture for the semantic grid. *Journal of Web Semantics*, 4(2):102–115, 2006.
[29] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199220, 1993.
[30] T. R. Gruber. What is an ontology?, 2005. http://www-ksl.stanford.edu/kst/what-is-an-ontology.html.
[31] A. Harth, S. Decker, Y. He, H. Tangmunarunkit, and C. Kesselman. A semantic matchmaker service on the grid. In *Proceedings of the 13^{th} international World Wide Web conference on Alternate track papers & posters (WWW Alt.)*, pages 326–327, New York, USA, 2004. ACM Press.
[32] I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.
[33] M. Li, P. V. Santen, D. Walker, O. Rana, and M. Baker. Sgrid: a service-oriented model for the semantic grid. In *Future Generation Computer Systems*, pages 7–18. 2004.
[34] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. Owl web ontology language semantics and abstract syntax. World Wide Web Consortium, February 2004, http://www.w3.org/TR/owl-semantics/.
[35] D. Roure, N. Jennings, and N. Shadbolt. The semantic grid: A future e-science infrastructure. In *Grid Computing - Making the Global Infrastructure*. John Wiley & Sons, 2003.
[36] D. D. Roure, N. R. Jennings, and N. R. Shadbolt. The semantic grid: Past, present and future. In *The Semantic Web: Research and Applications - LNCS*, volume 3532, page 726. Springer-Verlag, Berlin/Heidelberg, 2005. ISBN 978-3-540-26124-7.
[37] T. S. Somasundaram, R. A. Balachandar, V. Kandasamy, R. Buyya, R. Raman, N. Mohanram, and S. Varun. Semantic-based grid resource discovery and its integration with the grid service broker. Technical Report GRIDS-TR-2006-10, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, May 2006.
[38] H. Tangmunarunkit, S. Decker, and C. Kesselman. Ontology-based resource matching in the grid the grid meets the semantic web. In *The SemanticWeb - ISWC 2003 - LNCS*, volume 2870, pages 706–721. Springer-Verlag, Berlin/Heidelberg, 2003. ISBN 978-3-540-20362-9.
[39] S. Venugopal, R. Buyya, and L. Winton. A grid service broker for scheduling distributed data-oriented applications on global grids. In *Proceedings of the 2^{nd} workshop on Middleware for grid computing (MGC)*, pages 75–80, New York, USA, 2004. ACM Press.
[40] W. Xing, M. D. Dikaiakos, and R. Sakellariou. A core grid ontology for the semantic grid. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, pages 178–184, Washington, DC, USA, 2006. IEEE Computer Society.
[41] Y. Zhang and W. Song. Semantic description and matching of grid services capabilities. In *Proceedings of the 3^{rd} UK e-Science All Hands Meeting*, UK, September 2004.

Performance Analysis of the Active Object Pattern in Middleware

Paul J. Vandal, Swapna S. Gokhale
Dept. of CSE
Univ. of Connecticut
Storrs, CT 06269
{pvandal,ssg}@engr.uconn.edu

Aniruddha S. Gokhale
Dept. of EECS
Vanderbilt Univ.
Nashville, TN
a.gokhale@vanderbilt.edu

Abstract

A number of enterprises are turning towards the Service Oriented Architecture (SOA) approach for their systems due to the number of benefits it offers. A key enabling technology for the SOA-based approach is middleware, which comprises of reusable building blocks based on design patterns. These building blocks can be configured in numerous ways and the configuration options of a pattern can have a profound impact on system performance. A performance analysis methodology which can be used to assess this influence at design time can guide the selection of patterns and their configuration options and thus alleviate the possibility of performance problems arising later in the life cycle.

This paper presents a model-based performance analysis methodology for a system built using the Active Object (AO) pattern. The AO pattern is chosen because it lies at the heart of an important class of producer/consumer and publish/subscribe systems. Central to the methodology is a queuing model which captures the internal architecture of an AO-based system. Using an implementation of the queuing model in CSIM, we illustrate the value of the methodology to guide the selection of configuration and provisioning options for a stock broker system.

1 Introduction and motivation

The introduction of distributed components into the process of Enterprise Application Integration (EAI) has moved traditional integrations towards a more Service Oriented Architecture (SOA) based approach [6]. The SOA-based approach offers advantages such as robust, scalable, and cost-effective systems, achieved by reducing complexity and eliminating redundant code. Since these systems will be used in many critical domains, they will be expected to satisfy multiple Quality of Service (QoS) attributes.

A key enabling technology for SOA-based systems is QoS-enabled middleware [9], which comprises of building blocks based on design patterns to codify solutions to the commonly recurring problems. These patterns are highly flexible since they allow a system to be customized as per its requirements through an appropriate selection of configuration options. The configuration options of a pattern, however, exert a strong influence on system performance. Despite this influence, current trend in the performance analysis of these systems relies on empirical benchmarking and profiling, which involves measuring the system performance after it is implemented. These types of testing techniques, which are applicable very late in the life cycle, can be detrimental to the cost and schedule of a project, since several design and implementation iterations may be needed to achieve the expected performance. A systematic methodology to facilitate design-time performance analysis can guide the process of selecting patterns and their configuration options and may thus alleviate these pitfalls.

In this paper we present a model-based performance analysis methodology for a system built using the Active Object (AO) pattern [10, 5]. The AO pattern is chosen since it is widely used in a class of producer/consumer and publish/subscribe systems. At the heart of the methodology is a queuing model that captures the internal architecture of an AO-based system. Using a CSIM implementation of the queuing model [11], we illustrate the value of the methodology in guiding configuration and provisioning decisions for a case study of a stock broker system.

The paper is organized as follows: Section 2 provides an overview of the AO pattern. Section 3 presents the methodology. Section 4 illustrates the methodology with a case study. An overview of related research is in Section 5. Concluding remarks and future directions are in Section 6.

2 Description of the AO pattern

In a multi-threaded application, several threads may require the utilization of a common resource. These threads then compete for mutually exclusive access to the resource and utilize it for the total time taken to complete the required operation. For low request rates and short session durations, the performance of this architecture may be acceptable. However, for high request rates and long access times, performance degradation may be significant. The AO pattern can be used to alleviate the performance problems in such a system. This pattern provides concurrency and simplifies synchronized access to the shared resource by decoupling method invocation from method execution and creating the shared resource in its own thread of control.

The AO [9] is composed of the following components: Proxy, Activation List, Scheduler, Servant, and Method Requests. The interactions between these are initiated by a client thread invoking a method on the Proxy to the AO. The Proxy lies in the client thread and provides an interface to the public methods on the shared resource. Instead of immediately executing the method when invoked by the client thread, the Proxy constructs a Method Request and enqueues it on the Activation List of the AO. Thus, from the client thread's perspective, the method has been executed.

The Method Request is a structure that carries the parameters along with the other information necessary to execute the request later. It also has guards or synchronization constraints. The Activation List is a buffer which resides in the thread of the AO and holds all the pending requests. A Scheduler monitors the Activation List for requests that meet their synchronization constraints. It then chooses a request to be executed, dequeues it, and dispatches it to the servant, which actually executes the method.

The AO pattern can be used to implement a class of publish/subscribe and producer/consumer systems. In this paper, we focus on an AO-based producer/consumer system.

3 Performance analysis methodology

In this section we discuss the performance analysis methodology for an AO-based producer/consumer system. First, we describe the characteristics of a mutex-based producer/consumer system, which is then enhanced through the use of the AO pattern to mitigate its performance problems. We then present queuing models of mutex- and AO-based systems, followed by a discussion of the metrics that can be used to gauge system performance. Finally, we describe the implementation of the queuing models in CSIM.

3.1 System characteristics

We consider a producer/consumer system in which two applications act as producers to a remote consumer application. In such a system, the producers and the consumers require access to a common resource, for example, a message buffer. The system thus requires a synchronization strategy to create thread safe access to the resource.

Figure 1 shows a system implementation in which mutex constraints are used for multi-threaded synchronization. The solution comprises of a Consumer Handler which exists in its own thread of control and serves as a proxy to the consumer application. This handler contains a Message Queue for outgoing messages that is implemented with the Monitor Object pattern [10] to allow thread-safe synchronous access to the queue. It also contains a Message Broker that is responsible for monitoring the queue for new messages to be sent to the consumer. When the Message Queue contains messages, the Message Broker will contend with the producers to access it. Once it gains access, the Message Broker will get a message from the queue and send it to the consumer application. Additionally, the two producers contend for access to the Message Queue to put messages into it. When the Message Broker is actively working on the get and send functions, the Message Queue is locked from access. Similarly, the Message Queue is locked when a producer is trying to put a message on it. Thus, once an entity (a producer or the Message Broker) acquires the mutex lock from the Monitor Object, it retains control of the Message Queue until its transaction is complete, after which it releases the lock. Thus, the duration of these access times is defined by network latency. For low to moderate network loads, these access times are short and the system performance may be acceptable. In a congested network, however, long access times, partly driven by the TCP flow control, may cause performance problems and starvation of the entities from accessing the Message Queue.

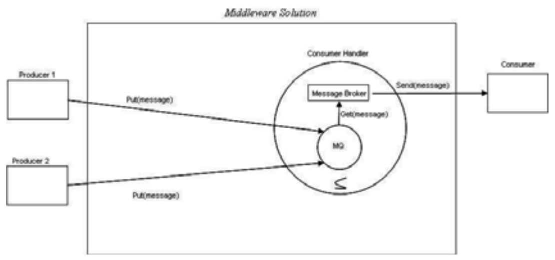


Figure 1. Mutex-based system

The above issues of the mutex-based system can be alleviated by using the AO pattern to decouple producers and consumers as shown in Figure 2. To decouple a producer, a Producer Handler Proxy to the Consumer Handler is introduced and implemented as a distributed AO. Its purpose is to receive messages from the producer and then put them in the Consumer Handler's Message Queue. The AO Proxy resides on the client application and provides an interface for the method to put messages on the Consumer Handler's Message Queue. When the put command is invoked by the client, the Proxy creates the corresponding Method Request and enqueues it on the Producer Handler's Activation List. The synchronization constraint of the put request is the requirement of the Proxy to gain control of the Message Queue. When the synchronization constraint is satisfied, the Scheduler dequeues the request and executes the method to put the message on the Message Queue. Thus, the time required to add a message to the queue is reduced to the internal access time of the middleware, which decouples the impact of the network latency on the producer side.

To decouple the consumer, the sending mechanism of the Message Broker is also implemented using an AO. A proxy interface containing the send method is implemented inside the Message Broker. When the Message Broker invokes the method to send a message, a Method Request is created by the Proxy and enqueued on the Activation List of the consumer-side AO. From the Message Broker's perspective, the sending of the message is nearly instantaneous, allowing it to relinquish control of the Message Queue after the small time required to get the message and to invoke the send command. This allows the affect of network latency to be decoupled from the system. Further, it also allows the processes of getting and sending messages to proceed asynchronously. The send Method Request is guarded while a message is being sent.

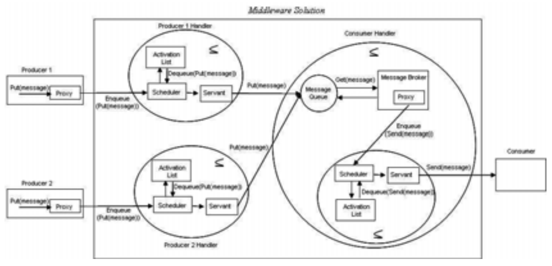


Figure 2. AO-based system

3.2 Queuing models

We assume that the arrival process at the producers is Poisson with rates λ_1 and λ_2. The times taken to put and get messages from the Message Queue remotely, over the network, are assumed to be exponential with parameter μ. The put and get times are assumed to be identically distributed for remote access, since these are governed by the network conditions, which are expected to be similar for both the producers and the consumer. Further, the internal times taken to put and get messages are assumed to be exponential with parameter τ. Since the internal access time is expected to be much lower than the remote access time, τ is at least an order of magnitude higher than μ.

Figure 3 shows the queuing model of the mutex-based system. The producers store the incoming messages in the producer-side buffers PS_1 and PS_2 until they gain access to the Consumer Handler's Message Queue, labeled MQ. The time taken by a producer to put a message on the queue and by the Message Broker to send a message to the consumer application is exponential with rate μ. A producer will not gain access to the queue if its buffer is empty or if the queue is full. Similarly, the Message Broker will not gain access to the Message Queue if the queue is empty.

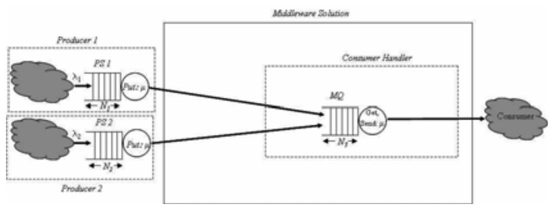


Figure 3. Queuing model: Mutex-based system

Figure 4 shows the queuing model of the AO-based system. The producer-side Activation Lists are modeled as buffers labeled $PHAL_1$ and $PHAL_2$ with capacities N_3 and N_4 respectively. A producer can continue to invoke the put method until its Activation List has spare capacity to enqueue a request. The time taken by a producer to put a message on its Activation List is exponential with rate μ. The time taken to enqueue a message on the queue internally by the producer-side servant is exponential with parameter τ. The servant can put messages on the Message Queue as long as it is not full. Also, it will not gain access to the queue if its corresponding Activation List is empty.

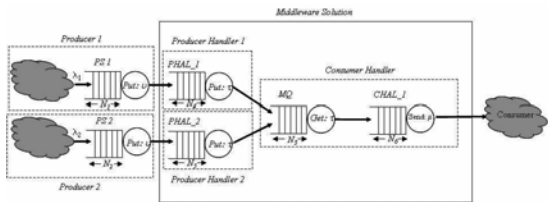


Figure 4. Queuing model: AO-based system

The consumer-side Activation List is also modeled as a buffer labeled $CHAL_1$ with capacity N_5. The time taken by the Message Broker to dequeue a message from the Message Queue is exponential with parameter τ. The rate at which the servant sends messages to the consumer is μ. The Message Broker will not gain access to an empty queue.

3.3 CSIM implementation

The implementation of queuing models using a general purpose simulation language/package such as CSIM [11] is fairly common practice. However, the implementation of the constraint of mutually exclusive access to the Message Queue in the producer/consumer systems required careful consideration and is described here. To allow synchronized access, we keep track of the threads which are "enabled" or whose constraints are satisfied and hence can gain access to the queue, in a single process that runs continuously for the entire duration of the simulation. An entity is considered to be enabled to gain access to the queue if its synchronization or guard constraints are satisfied. For example, in the mutex-based system, the producer is allowed access to the queue if there is at least one message in its buffer and if the queue is not full. This monitoring process then chooses one of the enabled entities according to a uniform distribution. It then provides the chosen entity with a semaphore, a structure called an "event" in CSIM, for the total time the entity needs access to the queue. Once the entity has completed its action on the queue, it releases the event back to the monitoring process, which then repeats the steps.

3.4 Performance metrics

In this section we define the metrics to gauge system performance. We also discuss their relevance from the user's and the provider's perspectives.

1. **Throughput:** This is the average rate at which messages are sent to the consumer application.

2. **Loss probability:** This is the average probability that an incoming message will be discarded on the producer side, due to a lack of buffer space.

3. **Response time:** This is the average time taken for a message to be received at the consumer application from the point it is created by a producer.

4. **Queue length:** This is the average queue length of the various queues in the system, namely, the producer-side queues and the Message Queue.

A service provider typically needs to balance competing concerns that consist of offering superior service performance while keeping the service cost acceptable. In a producer/consumer system, service performance will be deemed superior if the consumer application can receive messages at the same rate at which they are produced by the producers. The loss probability of the messages must thus be negligible. Further, these messages must be delivered with an acceptable response time. Thus, the first three metrics are relevant to a user's perception of performance.

To ensure acceptable service performance while maintaining reasonable costs, it is then the responsibility of the service provider to provision adequate resources. For a producer/consumer system, the storage resources consist of the various buffers used to hold messages. Thus, metric #4 can provide valuable guidance to a service provider in deciding the appropriate levels of resource provisioning.

4 Illustrations

In this section, we illustrate the potential of the methodology to guide configuration and provisioning decisions using the case study of a stock broker system [2]. The system has two producers, one each for creating NYSE and NASDAQ feeds, which we designate as producers #1 and #2 respectively. A remote data mining consumer application receives these feeds and provides stock data to the stock brokers. Since the stock brokers base important trading decisions on this data, it is extremely necessary that these feeds be received in a timely manner. Thus, the response time is a vital performance metric for the stock broker system.

For the sake of illustration, we use the nominal parameter values reported in Table 1. When using the methodology at design time, these values can be obtained for a specific hardware and operating environment either by conducting measurements on similar systems or by consultation with the experts. The performance metrics for the mutex-based and AO-based systems for the nominal values are reported in Table 2. The table indicates that both the systems have identical throughput, and is the same as the total rate at which messages are produced. The response time of the AO-based system, however, is lower than the mutex-based system. This is consistent with the average queue lengths, which are higher in the mutex-based system than the AO-based system. Thus, if the response time of the mutex-based system is unacceptable then the provider may have to disfavor the mutex-based system despite its simplicity and instead use the AO-based implementation.

Table 1. Nominal parameter values

Parameter	Value
Arrival rates (λ_1, λ_2)	15.0/sec.
Service rate (μ)	120.0/sec.
Producer-side buffers (N_1, N_2)	10
Message Queue (N_3)	100
Producer Hdlr. Activation Lists (N_4, N_5)	1
Consumer Hdlr. Activation List (N_6)	1
Internal access rate (τ)	1000.0/sec.

It is important to note that the performance of the AO-based system is better than the mutex-based system, even when the sizes of Activation Lists in the AOs are 1. When the Activation List sizes are 1, the producer blocks and cannot place any message on the Activation List until the producer-side servant gains access and puts the message it already has on the Message Queue. This shows that the AO-based implementation is effective in shielding the system from the impact of the network latency even with minimum possible Activation List sizes. This effectiveness may increase as the sizes of the Activation List increase.

Table 2. Performance of Mutex and AO systems

Metric	Mutex system	AO system
Throughput	30.00/sec.	30.00/sec.
Loss probability	0.00	0.00
Response time	0.0326	0.0227
Producer-side queue	0.261	0.146
Message queue size	0.456	0.052

Next we illustrate the utility of the methodology to enable sensitivity analysis, which is particularly valuable at design time, since at this stage the parameter values are not known with certainty. It is then crucial to determine the parameter ranges over which system performance is acceptable for a given set of configuration options. As an example, in the stock broker system frequent feeds are desirable to improve the accuracy of the data provided to the stock brokers. However, each feed must be delivered in a reasonable time. For given configuration options, the maximum rate of data feeds that can be sustained while providing an acceptable response time must then be determined. For this purpose, we vary the message arrival rate of the producers from 10.0/sec to 30.0/sec in steps of 5.0/sec. The performance metrics for both the systems as a function of the arrival rate are in Figure 5. The top left plot in the figure shows that the throughput of both the systems is identical over most of the range, except when the arrival rate is very close to 30.0/sec., at which the throughput of the mutex-based system dips slightly. As expected, the queue lengths and the response time increase as the arrival rate increases, however, the increase is more pronounced for the mutex-based system than for the AO-based system. When the arrival rate exceeds 25.0/sec. the queue lengths and the response time of the mutex-based system increase sharply, while the increase is still gradual for the AO-based system. Thus, if it is expected that the feed rates will exceed 25.0/sec., the performance of the mutex-based system may be unacceptable, mandating a switch to the AO-based system.

The above examples illustrate how the performance analysis methodology could be used to select an appropriate pattern and its configuration options to achieve acceptable system performance.

5 Related research

Performance and dependability analysis of some middleware services and patterns has been addressed by a few researchers. Aldred *et al.* [1] developed Colored Petri Net (CPN) models for different types of coupling between the application components and with the underlying middleware. They also defined the composition rules for combining the CPN models if multiple types of coupling are used simultaneously in an application. A dominant aspect of these works is related to application-specific performance modeling. In contrast, we are concerned with determining how the underlying middleware that is composed for the systems they host will perform. Kahkipuro [4] proposed a multi-layer performance modeling framework based on UML and queuing networks for CORBA-based systems. The methodology, however, is for generic CORBA-based client/server systems rather than for systems built using design patterns. The research reported in this paper is concerned with performance analysis of a specific design pattern used in the development of producer/consumer systems. The work closest to our work is in [8], where a performance model of the CORBA event service is developed. Our prior research has focused on performance analysis methodologies for other middleware patterns including the Reactor [3] and the Proactor patterns [7].

6 Conclusions and future research

In this paper we presented a model-based performance analysis methodology for an AO-based system. It comprised of a queuing model which captured the internal architecture of an AO-based system. A CSIM implementation of the model was used to demonstrate the utility of the methodology in guiding provisioning and configuration decisions on an example stock broker system.

Our future research consists of developing an analytical/numerical approach for the performance analysis of the AO pattern. Developing a strategy to compose the performance models of patterns mirroring their composition on the middleware stack is also a topic of future research.

Acknowledgments

This research was supported by the following grants from NSF: Univ. of Connecticut (CNS-0406376, CNS-SMA-0509271) and Vanderbilt Univ. (CNS-SMA-0509296).

References

[1] L. Aldred, W. M. P. van der Aalst, M. Dumas, and A. H. M. ter Hofstede. “On the notion of coupling in

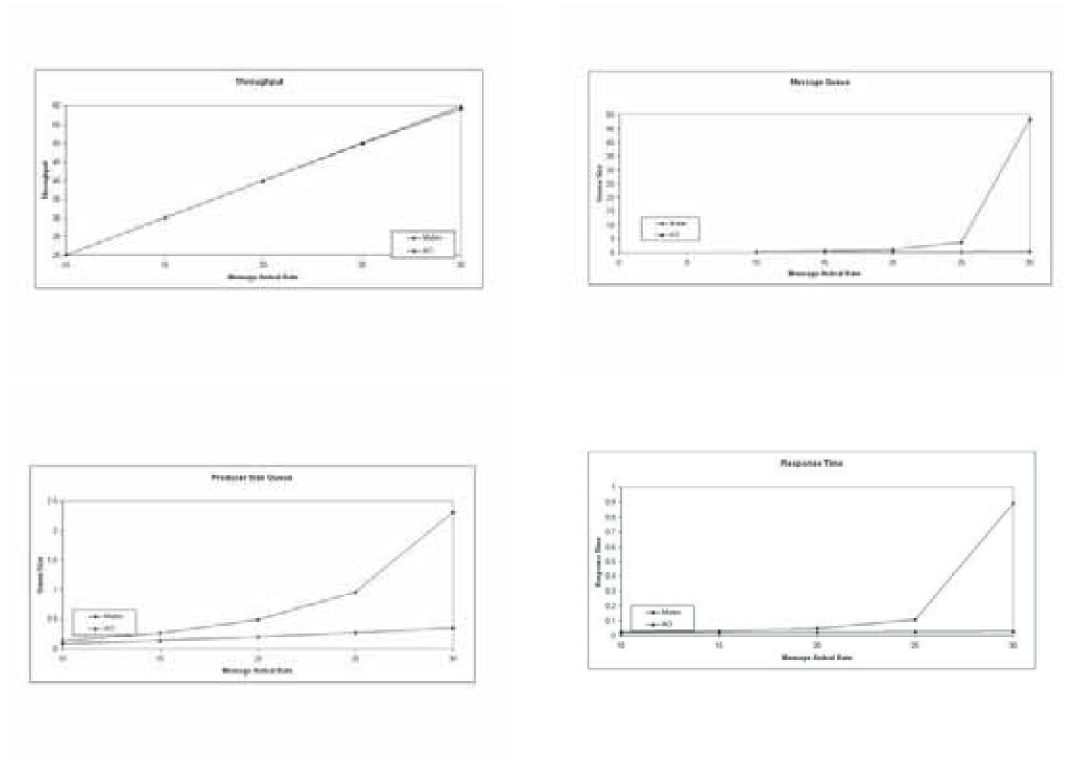


Figure 5. Sensitivity of performance measures to message arrival rates (λ_1, λ_2)

communication middleware". In *Proc. of Intl. Symposium on Distributed Objects and Applications (DOA)*, pages 1015–1033, Agia Napa, Cyprus, 2005.

[2] G. Banavar, T. Chandra, R. Strom, and D. Sturman. "A case for message oriented middleware". In *Proc. of the 13th Intl Symposium on Distributed Computing*, pages 1–18, London, UK, 1999. Springer-Verlag.

[3] S. Gokhale, A. Gokhale, and J. Gray. "Performance analysis of a middleware demultiplexing pattern". In *Proc. of Hawaii Intl. Conference on System Sciences (HICSS)*, January 2007.

[4] P. Kahkipuro. *"Performance modeling framework for CORBA based distrbuted systems"*. PhD thesis, Dept. of Computer Science, Univ. of Helsinki, Helsinki, Finland, May 2000.

[5] R. Greg Lavender and Douglas C. Schmidt. *Pattern languages of program design 2*, chapter Active object: an object behavioral pattern for concurrent programming, pages 483–499. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1996.

[6] ORACLE. "Bringing SOA value patterns to life". White Paper, June 2006.

[7] U. Praphamontripong, S. Gokhale, A. Gokhale, and J. Gray. "Performance analysis of an asynchronous Web server". In *Proc. of Intl. Conference on Computer Science and Applications*, pages 22–25, 2006.

[8] S. Ramani, K. S. Trivedi, and B. Dasarathy. "Performance analysis of the CORBA event service using stochastic reward nets". In *Proc. of the 19th IEEE Symposium on Reliable Distributed Systems*, pages 238–247, October 2000.

[9] R. E. Schantz and D. C. Schmidt. "Middleware for distributed systems: Evolving the common structure for network-centric Applications". In John Marciniak and George Telecki, editors, *Encyclopedia of Software Engineering*, pages 801–813. Wiley & Sons, 2002.

[10] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects, Volume 2*. Wiley & Sons, New York, 2000.

[11] H. Schwetman. "CSIM reference manual (revision 16)". Technical Report ACA-ST-252-87, Microelectronics and Computer Technology Corp., Austin, TX.

Analyzing the Applicability of a Theoretical Model in the Evaluation of Functional Size Measurement Procedures

Nelly Condori-Fernández, Oscar Pastor
Department of Information Systems and Computation
Valencia University of Technology
Valencia-Spain
{nelly,opastor}@dsic.upv.es

Abstract

A number of current proposals for software functional size measurement (FSM) exist in the literature; however there is as yet little validating evidence and no model to facilitate such validation. In the absence of tailor-made evaluation models, the Method Evaluation Model (MEM), used to evaluate Information Systems (IS) design methods, was adapted to evaluate three procedures based on two of the standard FSM methods. This paper analyses the applicability of the MEM in the software measurement context, with a review of three empirical studies.

Key words: FSM procedures, Method Evaluation Model, OO-Method, analyzing empirical studies.

1. Introduction

Software size measurement plays an important role in understanding and controlling the software development process and in obtaining deliverable products, software size being one of the essential parameters for project productivity and quality benchmarking. Over four decades a number of size measurement methods and techniques have been used, from Lines of Code through Function Point Analysis, but with the appearance of the ISO 14143 [1] as an attempt to standardize approaches only four, all Functional Size Measurement (FSM) methods, have been duly approved. These are: ISO/IEC 19761 (COSMI-FFP) [2], ISO/IEC 20926 (IFPUG FPA) [3], ISO/IEC 20968 (MARK II FPA) [4], and ISO/IEC 24570 (NESMA FPA) [5]. However, these methods are generic, and so far full automation of the measurement process has proved hard to achieve.

To overcome this limitation, various measurement procedures have been defined by researchers based on one of these four methods, in order to obtain the functional size of specific artifacts developed in particular contexts. There is increasing interest in this area but little systematic evaluation of accuracy, precision and user perceptions. In addition, as yet there is no evaluation model specifically for FSM.

To remedy this, the MEM model, originally intended for use in evaluating IS design methods, was adapted to evaluate three FSM procedures, OOmFP [6] and OOmFPweb [8], based on ISO/IEC 20926, and RmFFP [7], based on ISO/IEC 19761.

The objective here is therefore to assess how successful MEM model use was in this context. An integrative review of the three empirical studies is presented, together with a meta-analysis of the MEM relationships that underpin the MEM, and a discussion of possible future evaluation work.

This paper is structured as follows. Section 2 explains the main aspects of the MEM. Section 3 contains an overview of the three FSM procedures evaluated using the MEM. Section 4 includes a description of the general context, an investigation of the application of the MEM to the procedures and results, and an analysis of inconsistencies. The paper ends with conclusions and future work.

2. The Method Evaluation Model

The Method Evaluation Model (MEM) has been proposed by Moody [9].

The core of the MEM consists of the same perception-based constructs as the Technology Acceptance Model (TAM) [10], but adapted to evaluate methods. These constructs are called the Method Adoption Model (MAM) [9], [11]:

- *Perceived Ease of Use:* the extent to which a person believes that using a particular method would be free of effort.
- *Perceived Usefulness:* the extent to which a person believes that a particular method will be effective in achieving intended objectives.

- *Intention to Use:* the extent to which a person intends to use a particular method.

MAM is extended with additional constructs to provide inputs and predict final output (i.e. whether the method will be used in practice), i.e.:

- *Actual Efficiency:* the effort required to apply a method. This represents a MAM input variable.
- *Actual Effectiveness:* the extent to which a method achieves its aims; also a MAM input variable.
- *Actual Usage:* the extent to which a method is used in practice; a MAM output variable.

This model has been applied successfully in some design methods evaluated by Poels [14], Moody [11].

3. An overview of the FSM procedures evaluated using the MEM

The specific context of the three FSM procedures concerned is OO-Method [12], an automatic software production method based on model transformation, in which they are used to measure functional size at early stages of the software production process.

OO-Method is represented by three high-level models: 1) the *Requirements Model*, which includes a set of techniques that allow the capture of the functional properties that the system requires; 2) the *Conceptual Model*, which allows capturing of the static and dynamic properties of the system's functional requirements; and 3) the *Execution Model*, which includes a set of transformation rules that permits the transition from problem space to solution space. From this model, systematic and automatic software application can be generated on different platforms.

The three FSM procedures were defined to measure primitives of the Requirements Model and Conceptual Model, as described below.

OOmFP is a measurement procedure defined for sizing OO-Method conceptual schemas [6]. This procedure maps the primitives used onto the concepts used by Function Point Analysis, a standard FSM Method (ISO/IEC 20269).

OOmFPweb [8] is an extension of the above procedure to the sizing of Web applications, measuring presentation and navigation schemas that are specified with the Object-Oriented Web Solutions modeling approach (OOWS) [13].

The third procedure, RmFFP [7], was defined to measure the functional size of object-oriented systems from functional requirements specification obtained using the Requirements Model. RmFFP is based on the COSMIC-FFP standard method (ISO/IEC 19761).

4. Applicability of the Method Evaluation Model

Here we analyze the empirical studies evaluating the FSM procedures defined for the OO-Method approach [15],[16],[17]. These FSM procedures were evaluated in terms of their actual efficacy (accuracy and reproducibility), users' efficiency (measurement time) and how this in turn affects users' perceptions (usefulness and ease of use) and intention to use.

4.1. Empirical studies: general context

Table 1 summarizes the general context of these empirical studies: number of subjects selected, artifact to be measured (experimental object), the FSM procedure used (factor), and the variables that it is assumed will not influence the user's responses (parameters).

Table 1. Empirical Studies using MEM

Conditions	Study A [15]	Study B [17]	Study C [16]
Subjects	22 computer science students enrolled in the "Software Development Environments" course.	15 PhD students enrolled in the "Software Engineering for Web Environments" course.	35 computer science students enrolled in the "Software Development Environments" course.
Experimental Object	OO-Method conceptual schema of a Project Management System.	OOWS conceptual schema of an e-commerce application for a Photography Agency.	OO-Method requirements specification of a Car Rental Management System.
Factor	OOmFP.	OOmFPweb.	RmFFP.
Parameters	Quality of the OO-Method conceptual schema, similarity of knowledge of OOmFP, familiarity of subjects using OO-Method.	Quality of the OOWS conceptual schema, similarity of knowledge of OOmFPweb, familiarity of subjects using OOWS.	Quality of requirements specification, similarity of knowledge on FSM. Familiarity of subjects using OO-Method Requirements Model and the RETO Tool.

The data recorded by each subject was 1) functional size of the experimental object, 2) time used to carry out the measurement with the respective FSM procedure, 3) perceptions of usefulness and ease of use, and intention to use a specific FSM procedure.

In order to capture perceptions and intention to use, a survey, proposed by Moody [11], was adapted and applied to the three studies. Cronbach's alpha statistical test was used to analyze the reliability of the survey. We found that the reliability for the third study (C) was low; this meant that we had to redesign the survey and replicate the study.

Table 2 shows that the mean Cronbach's alpha obtained for PEOU construct was reliable (greater than 0.7) and representative (low standard deviation). However, the mean value for the PU and ITU constructs was not reliable. On the other hand, if we analyze the Cronbach's alpha including the replication of the third study carried out, the reliability of the ITU construct improved, although not that of the PU.

Table 2. Cronbach's alpha for the empirical studies

Construct	**Without replication**		**With replication**	
	Mean	Standard Dev.	Mean	Standard Dev.
PU	0.65	0.1323	0.69	0.1368
PEOU	0.75	0.0624	0.76	0.0572
ITU	0.67	0.1528	0.71	0.1536

4.2. Meta-analyzing the MEM relationships

In this section we aim to analyze whether the perceptions of efficiency (PEOU) and effectiveness (PU) really are the result of actual experience with an FSM procedure, and whether a measurer's performance really has an impact on his/her perceptions. To do this, we explore the results of regression analysis (Table 3 below) in terms of the significance level of the model, and the confirmation of the MEM relationships. Replication of the third study solely focused on analyzing the relationships between constructs of the MEM core, with the aim of improving the reliability of the PEOU, PU and ITU constructs (Table 3). Each MEM relationship is explored taking an integral view of these studies carried out.

With respect to the *influence of efficiency on perceived ease of use*, in studies (A) and (C) this causal relationship was corroborated with a high level of significance, approximately 60% of the variability of perceived ease of use. However, efficiency, quantified as number of size units measurable per unit of time, is only relevant in manual measurement, as in these studies. For automated procedures, there is a need for other variables influencing perceived ease of use to be identified.

With respect to the *influence of effectiveness on perceived usefulness*, a low significance level was shown in the first and third studies and therefore they were not confirmed. This result ran counter to expectations that FSM procedure effectiveness ought to have a strong impact on perceived usefulness. One reason for this could be that if this effectiveness is not perceived directly by the subjects it will be difficult for them to gauge the usefulness of the FSM procedure.

With respect to the *influence of perceived ease of use on perceived usefulness,* this relationship could not be empirically corroborated in the three last studies, meaning that perceived ease of use of a FSM procedure has no significant effect on perceived usefulness.

Table 3. Analysis of regression results for each empirical study

Empirical Studies	**Study A[1]**		**Study B [2]**		**Study C [3]**		**Replication Study C [4]**	
MEM Relationships	Sig. (p)	Confirmed	Sig. (p)	Confirmed	Sig. (p)	Confirmed	Sig. (p)	Confirmed
R1: Efficiency → PEOU	0.000	Yes	0.740	No*	0.000	Yes	--	--
R2: Effectiveness→ PU	0.158	No	0.017	Yes*	0.083	No	--	--
R3: PEOU→ PU	0.005	Yes*	0.292	No	0.432	No	0,329	No
R4: PEOU→ ITU	0.094	No	0.001	No	0.072	No	0.212	No
R5: PU → ITU	0.003	Yes	0.028	Yes	0.035	Yes	0.003	Yes

With respect to the *influence of perceived ease of use on intention to use*, this also could not be empirically corroborated.

Although the intention to use a FSM procedure is influenced significantly by *perceived usefulness*, coefficients of determination obtained in these empirical studies oscillated between 14% and 63.5%. Excluding the coefficient of the third study, where the reliability of the constructs PU and ITU was low, a mean value of 43.3% is obtained. This means that 43.3% of the variability of the intention to use an FSM procedure can be explained by perceived usefulness.

Therefore, the underlying assumption of the MEM applied in the FSM context has not been validated. It would appear that the user's perceptions are not solely affected by the measurer's performance in using a FSM procedure.

5. Conclusions and further work

This paper describes an analysis of the applicability of a design-method evaluation model, MEM, in the evaluation of three FSM procedures based on two ISO-certified FSM methods.

An integrative review of three empirical studies was carried out. Each study was designed to evaluate a specific FSM procedure for measuring certain artifacts modeled using the OO-Method approach. Divergences were revealed in certain MEM relationships when the regression analyses obtained in these empirical studies were analyzed (a meta-analysis).

According to the MEM, perceived usefulness is evaluated by looking at how effective the FSM procedure is in achieving users' intended objectives. However, on the basis of the analysis carried out in this paper, other factors should be included in order to explain perceived usefulness.

In terms of the perceived ease of use, although the results of the analysis verify that the efficiency of a procedure does influence perceived ease of use, this appears only to be applicable when the procedure is not automated.

Finally, the empirical studies analyzed corroborate that intention to use an FSM procedure in practice can be better predicted by perceived usefulness than perceived ease of use.

Future research will be necessary to consider other factors that will need to be included in a specially-developed model for evaluation of software size-measurement. Such work will contribute to the development of a new evaluation model, which will require further academic and practical empirical verification.

Acknowledgment

This work has been supported by the DESTINO project, Ref. TIN2004-03534, Spain.

References

[1] ISO, 1998. ISO/IEC 14143-1, Software measurement-Functional Size Measurement. Part 1: Definition of Concepts. International Organization for Standardization, Geneva, Switzerland, 1998.
[2] ISO, ISO/IEC 19761-COSMIC-FFP-A Functional Size Measurement Method, International Organization for Standardization_ISO, Geneva, 2003.
[3] ISO, ISO/IEC 20926:2003, IFPUG 4.1 Unadjusted functional size measurement method - Counting practices manual, International Organization for Standardization, Geneva, 2003.
[4] ISO, ISO/IEC 20968: 2002, Mk II Function Point Analysis - Counting Practices Manual, International Organization for Standardization, Geneva, 2002.
[5] ISO, ISO/IEC 24570: 2004, NESMA functional size measurement method version 2.1 - Definitions and counting guidelines for the application of Function Point Analysis, Int. Organization for Standardization, Geneva, 2004.
[6] Abrahao S., Poels G., Pastor O. A Functional Size Measurement Method for Object-Oriented Conceptual Schemas: Design and Evaluation Issues. Software & System Modelling, 5(1): 48-71, Springer Verlag, 2005.
[7] Condori-Fernández N., Abrahão S., and Pastor O. On the Estimation of Software Functional Size from Requirements Specifications, Journal of Computer Science and Technology, Springer-Verlag, 22(3): 358-370, 2007.
[8] Abrahão S., Pastor O., Measuring the Functional Size of Web Applications, International Journal of Web Engineering and Technology (IJWET), Vol. 1, No. 1, 5-16 pp. 2003, Inderscience Enterprises, Ltd., England.
[9] Moody D., The Method Evaluation Model: A Theoretical Model for Validating Information Systems Design Methods, Proceedings of the 11th European Conference on Information Systems, Naples-Italy, June 2003.
[10] Davis F. D., "Perceived Usefulness, Perceived Ease of Use and User Acceptance of Information Technology", MIS Quarterly, vol. 3, no. 3, 1989.
[11] Moody D. L., "Dealing with Complexity: A Practical Method for Representing Large Entity Relationship Models", PhD. Thesis, Department of Information Systems, University of Melbourne, Australia, 2001.
[12] Pastor O., Gomez J., Insfran E., Pelechano V., "The OO-Method approach for information systems modelling: from object-oriented conceptual modelling to automated programming", Information Systems 26, pp. 507-534, 2001.
[13] Pastor, O., Abrahão S. M., Fons J., An Object-Oriented Approach to Automate Web Applications Development, J. 2nd International Conference on Electronic Commerce and Web Technologies (EC-Web'2001), Munich, Germany, September 2001, pp. 16–28, Springer-Verlag.
[14] Poels G., Maes A., Gailly F., Paemeleire R., "Measuring User Beliefs and Attitudes towards Conceptual Schemas: Tentative Factor and Structural Equation Model", Fourth Annual Workshop on HCI Research in MIS, December 2005.
[15] Abrahão S., and Poels G., Experimental Evaluation of an Object-Oriented Function Point Measurement Procedure, Information and Software Technology (IST) 49(4): 366-380, 2007, Elsevier.
[16] Condori-Fernández N., Pastor O., An Empirical Study on the Likelihood of Adoption in Practice of a Size Measurement Procedure for Requirements Specification 6th IEEE International Conference on Quality Software, IEEE Computer Society, Beijing, China, October 2006.
[17] Abrahão S., Poels G., Further Analysis on the Evaluation of a Size Measure for Web Applications, 4th Latin American Web Congress, Puebla, Cholula, Mexico, October 25-27 2006, IEEE Press, pp. 230-240.

Software Documents: Comparison and Measurement

Tom Arbuckle*, Adam Balaban†, Dennis K. Peters‡ and Mark Lawford§
*Department of Computer Science and Information Systems, College of Informatics and Electronics,
CSIS Building, University of Limerick, Plassey Park, Limerick, Ireland. Email: tom.arbuckle@ieee.org
†Institute of Informatics, Warsaw University, Banacha 2, 02-097 Warsaw, Poland. Email: ab@mimuw.edu.pl
‡Electrical and Computer Engineering, Faculty of Engineering and Applied Science,
Memorial University of Newfoundland, St. John's NL, Canada A1B 3X5. Email: dpeters@engr.mun.ca
§Department of Computing and Software, Faculty of Engineering, McMaster University,
Hamilton, Ontario, Canada, L8S 4K1. Email: lawford@mcmaster.ca

Abstract— **For some time now, researchers have been seeking to place software measurement on a more firmly grounded footing by establishing a theoretical basis for software comparison. Although there has been some work on trying to employ information theoretic concepts for the quantification of code documents, particularly on employing entropy and entropy-like measurements, we propose that employing the Similarity Metric of Li, Vitányi, and coworkers for the comparison of software documents will lead to the establishment of a theoretically justifiable means of comparing and evaluating software artifacts. In this paper, we review previous work on software measurement with a particular emphasis on information theoretic aspects, we examine the body of work on Kolmogorov complexity (upon which the Similarity Metric is based), and we report on some experiments that lend credence to our proposals. Finally, we discuss the potential advantages derived from the application of this theory to areas in the field of software engineering.**

I. Introduction

In the transition from the idea or concept for an element of software to be implemented to the code that implements it and beyond, it is recognised that there are several (possibly overlapping) stages (perhaps labeled by analysis, design, coding, testing and support). In the production of quality software, we assume that these stages — however they are assigned — are documented. For the purposes of evaluation and assessment, we therefore need to be able to compare documents that describe or specify [1] — or indeed implement — software.

In software engineering, the comparison or measurement of software documents (including source code and executable objects) has traditionally been assigned to the subfield of software measurement – often called 'metrics'. The task of characterising something that is effectively infinitely malleable has been frought with difficulty. It has been difficult to agree what to measure and the suitability of what is measured, and to validate the measures against the implementations. Thus the field has largely become a means of providing indicators, symptoms to be evaluated by experienced practitioners rather than a means of objectively measuring attributes of software and their significance. We believe that the Similarity Metric (of Li, Vitányi and others) can help in this regard. Kolmogorov complexity, on which the measure is based, in a very precisely defined sense describes the inherent complexity of objects. Since the documents produced to describe, specify or implement code are also 'objects' which embody their intended meaning, we regard them as being inherently suitable comparands for this method.

To be more specific, we wish to examine ways in which the Similarity Metric can be employed in software engineering. This will include the comparison of specifications or descriptions of software before the software is implemented.

This paper is structured as follows. In the next sections, we introduce software measurement, information theory and Kolmogorov complexity. There follows an overview of the work, largely by Li and Vitányi, on the definition of a universal comparator, the Similarity Metric [2], whose foundation is the complexity theory of Kolmogorov. In the next section, we will attempt to relate complexity and measurement from the viewpoint of the field of software engineering, in particular software measurement. After a section in which we apply Cilibrasi's implementation [3] of an approximation of the Similarity Metric [4] to a set of software engineering examples, we discuss what we believe to be the significance of these techniques to the software engineering field and outline additional applications beyond those detailed herein. We close with our conclusions and acknowledgements.

II. Software Measurement

A. Introduction

The field of software measurement has a long history [5], [6], [7] and a broad literature. We can measure software to examine its performance; its structure or design; its correctness; its quality; its evolution (in terms of maintenance or extension); the processes executed by its creators. Broadly speaking, the field is important because only by performing measurements (however, defined) on software and comparing those measurements, can we make objective statements about these topics. For a good overview of the field of software measurement, see Fenton and Pfleeger [8].

B. Software Engineering: Metrics and Measures

Largely in agreement with Zuse [9], we define the words 'measure', 'measurement' and 'metric' as follows.

- A *measure* is a mapping from empirical objects (objects to be measured) to formal objects (measurement values). We will define a *measurement* to be the result of applying a 'measure'.

- A *metric* is a means of determining the distance between two entities. A metric function (of two arguments) must also (1) return null for identical entities; (2) be symmetric; and (3) obey the triangle inequality.

In the literature, different definitions for these words are common. In particular, in software engineering, other conventions are often followed. Lorenz and Kidd [10], for example, make the following two definitions.

- *Metric* A standard for measurement. Used to judge the attributes of something being measured such as quality or complexity, in an objective manner.
- *Measurement* The determination of the value of a metric for a particular object.

Our definitions are more in agreement with the mathematics literature but potential for confusion should be borne in mind.

We will not discuss further the extensive literature concerning definitions, axiomatic approaches or properties that software 'metrics' might have [11], [12], [13], [14], [15].

III. Information Theory, Entropy

In his 1948 paper [16], Shannon was concerned with the transmission of information from a source to a sink over a channel. He considered encoding of information, discrete and continuous encodings, and transmission in the presence or absence of noise. As a measure of the 'quantity' of information to be passed through a channel, he introduced entropy, an analogue of the thermodynamic entropy of Boltzmann. In the discrete case, (Shannon) entropy takes its well-known form

$$H(X) = -\sum_{x \epsilon X} p_x log p_x \tag{1}$$

where $H(X)$ is the entropy for a source emitting codes x from a set X, and the p_x are the probabilities the codes x occur.

It can be claimed that Shannon's paper marked the founding of the field of information theory. Certainly, an indication of the quantity of information being produced by a source is a useful concept that has since been applied in many fields [17].

IV. Kolmogorov Complexity

To quote from the book [18] by Li and Vitányi, "Shannon's entropy measures the uncertainty in a statistical ensemble of messages, while Kolmogorov complexity measures the algorithmic information in an individual message." Also sometimes known as 'algorithmic entropy', Kolmogorov complexity was introduced independently by Solomonoff [19], Kolmogorov [20], and Chaitin [21].

More formally, the Kolmogorov complexity, $K(x)$ of a binary string x is the length of the shortest (prefix-free) binary program to compute x on a universal computer such as a universal Turing machine. $K(x)$ represents the number of bits necessary to (computationally) describe the string x.

Although this definition sounds somewhat abstract, there are indeed strong connections with the Shannon entropy [22], [23]. It can be shown, for example, that the expected value of the Kolmogorov complexity for a random string for any probability distribution function will have a value to within an additive constant (dependent only on the executing Turing machine) of the Shannon entropy.

As might be expected, there are also connections between Kolmogorov complexity and thermodynamics. Bennett *et al.* [24] describe how it can be related to the thermodynamic cost (minimal entropy increase in the environment) of data transformations. This follows from early work on thermodynamics in computing by Landauer [25].

V. Measuring Similarity

A. Information Distance

The idea of being able to calculate a value for the complexity of objects leads naturally to the idea of being able to compare how similar objects are to each other. In their 1998 paper, Bennett *et al.* [26] investigated the idea of 'an "absolute information distance metric" between individual objects' that they have subsequently shown [27] to obey the (mathematical) metric properties up to an additive constant. The information distance was to form the basis of the subsequent normalised similarity metric. Here, normalisation means that $0 \leq d(x, y) \leq 1$ with 0 indicating identical objects.

B. The Similarity Metric

A first attempt at a normalised similarity metric was presented by Li *et al.* [28]. However, in their 2004 paper [27], Li *et al.* presented an improved version, the normalised information distance (NID) given by

$$NID(x, y) = \frac{max\{K(x|y), K(y|x)\}}{max\{K(x), K(y)\}} \tag{2}$$

Here $K(x|y)$, for example, is the conditional Kolmogorov complexity of x given y. This is the length of the shortest program for a universal Turing machine to output x when given an input y. They showed that this new distance satisfies the metric properties up to an additive term of $O(1/K)$, where K is the maximum of Kolmogorov Complexities involved. Then $1 - NID(x, y)$ has the natural interpretation of the number of bits of shared information per bit of information of the string with more information.

C. Practical Implementation — CompLearn

Since the idea of the NID concerns compression of data, the normalised information distance can be approximated by using real compressors, including most commonly known compressors, which obey certain properties (idempotency, monotonicity, symmetry and distributivity) [4]. The resulting measure is called the normalised compression distance (NCD).

$$NCD(x, y) = \frac{C(xy) - min\{C(x), C(y)\}}{max\{C(x), C(y)\}} \tag{3}$$

where $C(x)$, for example, denotes the approximation of a Kolmogorov complexity $K(x)$ by the length of the compressed data produced by an instance of a real compressor and xy denotes the concatenation of x and y.

Although the theory for the NID is exact, additional theoretical work was carried out by Cilibrasi and Vitányi [4] to show that the NCD was a good approximation whose difference from the NID was dependent only on the quality of the approximation of a 'perfect' compressor by a real one.

An implementation of the NCD has been made publicly available as the CompLearn toolkit [3]. We have employed version 0.9.7 of the toolkit in the experiments that follow. The toolkit compressors we employ are the Lempel-Ziv zlib algorithm, the bzip2 block-sorting compressor and the blocksort algorithm provided in CompLearn by Cilibrasi.

VI. Software, Complexity, Information Theory

Campbell [29] was already considering whether entropy could be used as a metric (in the mathematical sense) as early as 1965. A 1972 paper by Hellerman [30] employed entropy to measure information in a computer's memory as a measure of computational work. However, possibly the first use of entropy in the discussion of design was van Emden's 1969 paper [31] and his later thesis [32]. Van Emden's concern was the use of a form of entropy (called "surplus entropy", or "entropy loading") to decide how to decompose an object into subcomponents. This work was taken up by Chanon who showed [33], [34] how van Emden's work could be employed for the structuring of software. See also [35], [36], [37].

Since this early work, numerous authors have sought to apply information theory concepts to software engineering. A full review would be a paper in its own right. To list some examples, judging the information-theoretic complexity of software specifications was studied by Coulter *et al.* [38]. Bansiya *et al.* [39] modeled the complexity of object-oriented systems using entropy. (See Tegarden *et al.* for a 'metrics' approach [40].) 'Coupling', the degree of interconnection or dependency between software components, and 'cohesion', the degree of internal interconnection, were modeled in information theoretic terms by Allen *et al.* [41], [42]. There are also studies of apportioning complexity [43], [44] and of the correlation between forms of code complexity and the likelihood of errors [45], [46].

Research on using information theory based methods for quantifying software continues, a recent paper by Sarkar *et al.* [47] being one good example.

VII. Examples

A. Experiments in other fields

There should be no doubt that Cilibrasi's implementation of Li and Vitányi's Similarity Metric has already proven its utility and correctness. Amongst others, it has been applied to program plagiarism detection [48], genomics [4], cross-language textual similarity [49], and the classification of musical styles [50]. Although a subsequent paper by Vitányi [51] states that the program has also been applied to the classification of programs written in the languages Ruby and C, we have not found details of these experiments. In addition, we are unaware of any descriptions of applications of these techniques in the field of software engineering.

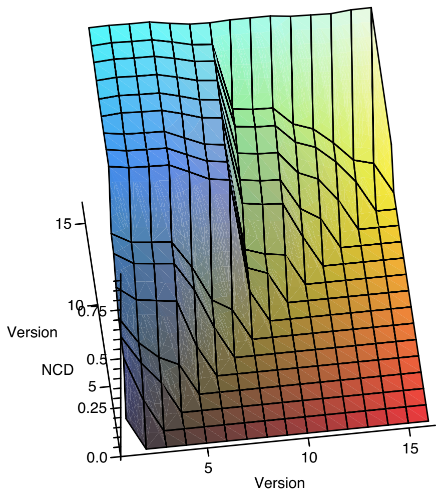


Fig. 1. *ncd* operating on sources

B. Sample Experiment — The 'slocate' package

The open-source software package *slocate* [52] is designed to catalogue and index (as the superuser) all files present in a specified area of a filesystem but to answer queries such that the visibility of files is filtered by the authorisation privileges of the current software user. *slocate* is an ideal candidate for one of our experiments: its code is short but performs fairly complex operations; it has a long history with many publicly available releases; and it has also been redesigned on at least one occasion. Reference indices (for use in diagrams) and source lengths in kilobytes for the releases of *slocate* that we studied are as shown in table I.

TABLE I

slocate RELEASES, INDICES, SIZES (KILOBYTES)

Rel.	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7	3.1
Ind.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Size	26.6	26.7	24.0	23.9	29.7	31.3	32.2	70.3	72.3	72.7	70.7	73.6	82.6	91.3	92.0	67.4

1) Source Code: For each version, we applied the *ncd* program from CompLearn to the concatenation of all ".h" and ".c" files whose lengths were shown in table I. A plot of (half of) the NCD similarity matrix is shown in figure 1.

2) Indexing Trace: Using the *strace* program which provides a listing of calls to system functions from user programs, we created a trace (sequence of program calls) of the operation of the versions of the *slocate* program creating a database on the same data. Figure 2 shows a plot of (half of) the NCD similarity values.

3) Search Trace: The *strace* program was again used to produce a trace of system calls, this time for answering the same search query on the database produced in the previous experiment. Figure 3 shows the plot of (half of) the results of running *ncd* on these traces.

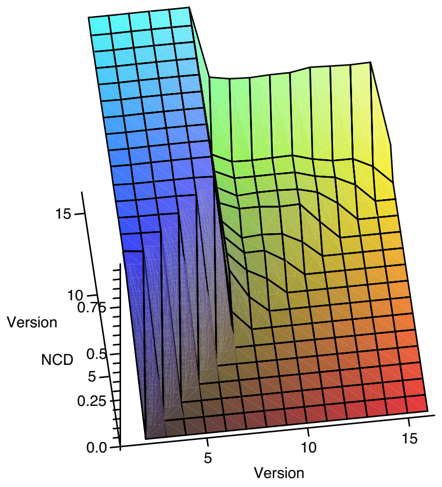


Fig. 2. *ncd* on indexing trace

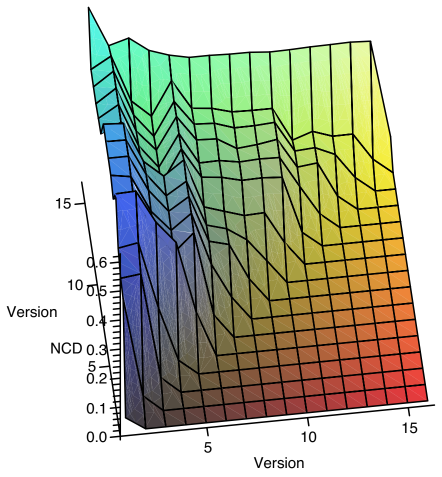


Fig. 3. *ncd* on search trace

C. Analysis

Each of the figures shows a pairwise similarity between different versions of *slocate*. The *ncd* similarity matrix is (almost) symmetric and so values below (not on) the main diagonal of the matrix have been zeroed to permit easier viewing. Remember that NCD values close to zero denote similarity and those close to one denote strong dissimilarity.

Let us examine figure 1. From left to right, the columns represent the NCD values for increasing versions. For indices 1 to 7, we see a steep increase in the degree of dissimilarity with increasing versions so that, above index 8, the degree of dissimilarity forms a plateau in the top-left corner. From index 8 onwards, we see (up the columns) increasing degrees of dissimilarity leading to a 'back wall' for the final index.

This diagram tells us that the author was justified in his versioning system since the discontinuities correspond to major releases (2.0, 3.1). We can also see the gradual differences between minor versions. We can claim that this is evidence of good incremental design. We can further claim that this is evidence of good structure since the changes introduced between minor versions were permitted by the design without engendering major modifications.

Figure 2 shows the results of comparing the (system) operations for building an index with each version of *slocate* being run on the same input. Differences between operations for minor releases of the 1.x code are very apparent. Differences in operations for the 2.x series are more gradual and there appears to be some similarity to the operations of the 3.x series (forming the 'back wall' of the figure). We see that NCD is detecting the right differences for our purposes.

Now examine figure 3 showing the result of a search on the respective databases. Looking along the columns, indices 1 and 2 show that the first two versions behave differently from the others. We can see another 'ridge' at index 5. The two previous versions were 'optimising' the code whereas this version adds functionality (globbing, greater compatibility with GNU *locate*). We can also see a plateau between (horizontal) indices 6 and 10 for indices 11 to 15 indicating that the final releases of the 2.x series behave differently from both the 1.x series and earlier versions of the 2.x series. Again, we note the presence of the 'back wall' in the diagram. The operations performed by version 3.1 are clearly different from those of the 2.x series.

Finally, compare the diagrams with each other. We see clear similarities between figures 1 and 2 and clear differences between these two and figure 3. The building of the database mirrors the versioning with little work being done on it between minor releases. The bulk of the change we see is in the search of the database even within versions.

Suppose a developer has data about their current project similar to that presented here. The developer is aware of the changes being made but may not be so aware of their repercussions. By comparing current behaviour with that of previous versions, the developer can see where a possibly small change in the code results in a much larger change in behaviour (that might not be immediately apparent).

These plots are clearly of great utility to both managers and coders alike. For minimal investment of time and energy, it is easy to see important aspects of change throughout the version history of the project. The successful application of NCD to *strace* logs we consider very promising but there are many other applications for this technique.

The recent trace function method (TFM) [53], a successor of the earlier trace assertion method [54], permits software specification in terms of traces as do other formalisms such as enumerative specification [55], [56], path expressions [57] and the work of Broy [58]. This suggests the following.

Firstly, using simulation techniques applied to those specifications we can generate traces for the specified software. Whereas our earlier experiments required working code and the *strace* program, we can obtain the data for our comparisons at the design stage thereby permitting the measurement of the development progress and of the complexity of the software before the implementation work has begun. We believe that even the design should be done in incremental steps and these tools will be helpful in measuring this. Given a sufficient body of data, design complexity can be validated and properties of future designs quantified before coding begins.

Secondly, Peters *et al.* [59] are seeking to encode program specifications written using TFM in OMDoc [60] XML files. These encodings of software specifications exist before the implementations the specifications describe. Descriptions of existing software can also be created using this encoding. In future experiments, we will be interested to examine OMDoc descriptions or specifications of software.

VIII. Discussion, Future Work

The work of Li, Vitányi and Cilibrasi provides a fundamentally new and theoretically justified approach that we can apply to our problems. Its application in the field of software engineering presents many opportunities to help practitioners do a better job of creating quality software. There are numerous additional applications of the techniques that we have shown.

- We can automate the monitoring of the evolution of code as it moves through different phases of implementation, testing and maintenance. This will allow us to track the introduction of additional complexity or to see simplifications made without loss in functionality.
- Once an interface for an implementation is specified, application of these methods will permit the comparison of different implementations. We can compare internal designs for different modules and identify similar modules as potential refactoring candidates.
- Given some estimate of the complexity of a design, we will be better able to make management decisions about the quantities of resources required to complete a given design or programming task.
- Given a specification, we can implement that specification in many possible programming languages. If we can find a common representation of their execution output, we can compare different languages to see if they aid or hinder the creation of implementations.
- By measuring the modules that comprise the implementation, we will be able to make estimates of the relative complexities of the code for those units. Since error density has been shown to correlate with code complexity, we will have better ideas of where to concentrate our testing efforts and of where errors are likely to occur.
- Gathering requirements and creating usage scenarios allows us to decide what needs to be created and to evaluate whether a given task is viable. Given a suitable encoding, the comparison of different sets of requirements for the same project would permit some evaluation of relative complexities before going on to design.
- The work of different coders or designers working to a common specification can be compared using this method. Planning for future change, a more experienced designer might produce a more complex design. However, even the ability to reveal this is of great utility.
- In addition we intuit connections with considerations of coverage [61] that we would like to investigate.

We are aware that these suggestions do have some drawbacks. The interpretation of these comparisons requires some thought. An attempt to compare a 'large' set of documents might fail for practical reasons. More importantly, however, the question of the coding of the documents needs to be considered. Vitányi has stated [51] that while the technique is robust with respect to a change in underlying compressor types (although see [62]), there are still situations in which a naïve application of the technique may fail. He states that further research is necessary to examine the case where the input strings are overly sensitive to the encoding used. Nevertheless, we foresee that applying the Similarity Metric in software measurement will provide many areas for future exploration and are highly encouraged by our results so far.

IX. Conclusion

In this paper, we have shown that the application of the similarity metric of Li and Vitányi (based on Kolmogorov complexity theory) in the field of software engineering will provide a theoretically sound basis on which to found software measurements. We claim that having a theoretically justified means of comparison of software documents, at many stages in the software lifecycle, will have a beneficial effect on the reproducibility and justifiability of measurement claims. We have shown that using this method can provide useful information for software engineers during design, testing and maintenance. We have described several ways in which this theory can be applied and, in future publications, we will elaborate on our suggestions and their practical utilisation.

Acknowledgments

The authors thank numerous reviewers for their comments on earlier drafts. This research is supported by Science Foundation Ireland (grants 01/P1.2/C009 and 03/CE3/1405).

References

[1] D. L. Parnas, "Precise description and specification of software," in *MDS '95: Proc. 2nd international conference on Mathematics of dependable systems II*. Oxford University Press, Inc., 1997, pp. 1–14.

[2] M. Li, X. Chen, X. Li, B. Ma, and P. Vitányi, "The similarity metric," in *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 2003, pp. 863–872.

[3] R. Cilibrasi, "The CompLearn Toolkit," 2003. [Online]. Available: http://complearn.sourceforge.net/

[4] R. Cilibrasi and P. Vitányi, "Clustering by compression," *IEEE Trans. Information Theory*, vol. 51, no. 4, pp. 1523–1545, April 2005.

[5] R. J. Rubey and R. D. Hartwick, "Quantitative measurement of program quality," in *Proceedings of the 1968 23rd ACM national conference*. New York, NY, USA: ACM Press, 1968, pp. 671–677.

[6] M. H. Halstead, *Elements of Software Science (Operating and programming systems series)*. New York, USA: Elsevier Science Inc., 1977.

[7] B. H. Yin and J. W. Winchester, "The establishment and use of measures to evaluate the quality of software designs," in *Proceedings of the software quality assurance workshop on Functional and performance issues*, 1978, pp. 45–52.

[8] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*. Boston, MA, USA: PWS Publishing Co., 1998.

[9] H. Zuse, *Software Complexity: Measures and Methods*. Hawthorne, NJ, USA: Walter de Gruyter & Co., 1990.

[10] M. Lorenz and J. Kidd, *Object-oriented software metrics: a practical guide*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994.

[11] N. Fenton, "Software measurement: A necessary scientific basis," *IEEE Trans. Softw. Eng.*, vol. 20, no. 3, pp. 199–206, 1994.

[12] B. Henderson-Sellers, "The mathematical validity of software metrics," *SIGSOFT Softw. Eng. Notes*, vol. 21, no. 5, pp. 89–94, 1996.

[13] E. J. Weyuker, "Evaluating software complexity measures," *IEEE Trans. Softw. Eng.*, vol. 14, no. 9, pp. 1357–1365, 1988.

[14] L. C. Briand, S. Morasca, and V. R. Basili, "Property-based software engineering measurement," *IEEE Trans. Softw. Eng.*, vol. 22, no. 1, pp. 68–86, 1996.

[15] N. Fenton, "When a software measure is not a measure," *Softw. Eng. J.*, vol. 7, no. 5, pp. 357–362, 1992.

[16] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423 and 623–656, 1948.

[17] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Wiley-Interscience, 2006.

[18] M. Li and P. Vitányi, *An introduction to Kolmogorov complexity and its applications (2nd ed.)*. Springer-Verlag, 1997.

[19] R. J. Solomonoff, "A formal theory of inductive inference. part I and part II," *Information and Control*, vol. 7, no. 1 and 2, pp. 1–22 and 224–254, 1964.

[20] A. N. Kolmogorov, "Three approaches to the quantitative definition of information," *Probl. Inform. Trans.*, vol. 1, no. 1, pp. 1–7, 1965.

[21] G. J. Chaitin, "On the length of programs for computing finite binary sequences: statistical considerations," *J. ACM*, vol. 16, no. 1, pp. 145–159, 1969.

[22] P. D. Grünwald and P. M. B. Vitányi, "Kolmogorov complexity and information theory with an interpretation in terms of questions and answers," *J. of Logic, Lang. and Inf.*, vol. 12, no. 4, pp. 497–529, 2003.

[23] ——, "Shannon information and Kolmogorov complexity," September 2004, submitted to IEEE Trans. Information Theory.

[24] C. H. Bennett, P. Gács, M. Li, P. M. B. Vitányi, and W. H. Zurek, "Thermodynamics of computation and information distance," in *STOC '93: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM Press, 1993, pp. 21–30.

[25] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM J. Res. Develop.*, vol. 5, no. 3, pp. 183–191, July 1961.

[26] C. H. Bennett, P. Gács, M. Li, P. Vitányi, and W. H. Zurek, "Information distance," *IEEE Transactions on Information Theory*, vol. 44, no. 4, pp. 1407–1423, July 1998.

[27] M. Li, X. Chen, X. Li, B. Ma, and P. Vitányi, "The similarity metric," *IEEE Transactions on Information Theory*, vol. 50, no. 12, pp. 3250–3264, December 2004.

[28] M. Li, J. H. Badger, X. Chen, S. Kwong, P. Kearney, and H. Zhang, "An information-based sequence distance and its application to whole mitochondrial genome phylogeny." *Bioinformatics*, vol. 17, no. 2, pp. 149–154, 2001.

[29] L. L. Campbell, "Entropy as a measure," *IEEE Trans. Information Theory*, vol. 11, no. 1, pp. 112–11, 1965.

[30] L. Hellerman, "A measure of computational work," *IEEE Trans. Computers*, vol. C-21, no. 5, pp. 439–446, May 1972.

[31] M. H. van Emden, "Hierarchical decomposition of complexity," *Machine Intelligence*, vol. 5, pp. 361–380, 1969.

[32] ——, "An analysis of complexity," Ph.D. dissertation, Mathematisches Zentrum, Amsterdam, 1971.

[33] R. N. Chanon, "On a measure of program structure," in *Programming Symposium, Proceedings Colloque sur la Programmation*. London, UK: Springer-Verlag, 1974, pp. 9–16.

[34] ——, "On a measure of program structure." Ph.D. dissertation, Carnegie-Mellon University, 1974.

[35] S. Henry and D. Kafura, "Software structure metrics based on information flow," *IEEE Transactions on Software Engineering*, vol. 7, no. 5, pp. 510–518, September 1981.

[36] N. Chapin, "An entropy metric for software maintainability," in *Proceedings of the Twenty-Second Anuual Hawaii International Conference on System Sciences*, vol. II, Jan 1989, pp. 522–533.

[37] W. R. Torres and M. H. Samadzadeh, "Software reuse and information theory based metrics," in *Proc. 1991 Symposium on Applied Computing*, April 1991, pp. 437–446.

[38] N. S. Coulter, R. B. Cooper, and M. K. Solomon, "Information-theoretic complexity of program specifications," *Comput. J.*, vol. 30, no. 3, pp. 223–227, 1987.

[39] J. Bansiya, C. Davis, and L. Etzkorn, "An entropy-based complexity measure for object-oriented designs," *Theor. Pract. Object Syst.*, vol. 5, no. 2, pp. 111–118, 1999.

[40] D. P. Tegarden, S. D. Sheetz, and D. E. Monarchi, "A software complexity model of object-oriented systems," *Decis. Support Syst.*, vol. 13, no. 3-4, pp. 241–262, 1995.

[41] E. B. Allen and T. M. Khoshgoftaar, "Measuring coupling and cohesion: An information-theory approach," in *METRICS '99: Proc. 6th Int. Symp. on Software Metrics*. IEEE Computer Society, 1999, p. 119.

[42] E. B. Allen, T. M. Khoshgoftaar, and Y. Chen, "Measuring coupling and cohesion of software modules: An information-theory approach," in *METRICS '01: Proceedings of the 7th International Symposium on Software Metrics*. IEEE Computer Society, 2001, p. 124.

[43] J. C. Munson and T. M. Khoshgoftaar, "The dimensionality of program complexity," in *ICSE '89: Proceedings of the 11th international conference on Software engineering*. ACM Press, 1989, pp. 245–253.

[44] T. M. Khoshgoftaar and D. L. Lanning, "Are the principal components of software complexity data stable across software products?" in *Proc. 2nd Int. Software Metrics Symposium*, 1994, pp. 61–72.

[45] D. L. Lanning and T. M. Khoshgoftaar, "Modeling the relationship between source code complexity and maintenance difficulty," *Computer*, vol. 27, no. 9, pp. 35–40, 1994.

[46] N. E. Gold, A. M. Mohan, and P. J. Layzell, "Spatial complexity metrics: An investigation of utility," *IEEE Trans. Softw. Eng.*, vol. 31, no. 3, pp. 203–212, 2005.

[47] S. Sarkar, G. M. Rama, and A. C. Kak, "API-based and information-theoretic metrics for measuring the quality of software modularisation," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 14–32, 2007.

[48] X. Chen, B. Francia, M. Li, B. McKinnon, and A. Seker, "Shared information and program plagiarism detection," *IEEE Trans. Information Theory*, vol. 50, no. 7, pp. 1545–1551, July 2004.

[49] K. Koroutchev and M. Cebrián, "Detecting translations of the same text and data with common source," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 10, October 2006.

[50] R. Cilibrasi, P. Vitányi, and R. de Wolf, "Algorithmic clustering of music based on string compression," *Computer Music Journal*, vol. 28, no. 4, pp. 49–67, 2004.

[51] P. Vitányi, "Universal similarity," in *Proc. IEEE ISOC ITW2005 on Coding and Complexity*, M. Dinneen, Ed., 2005, pp. 238–243.

[52] K. Lindsay, "slocate," 1998, Canonical URL is http://slocate.trakker.ca/. [Online]. Available: http://freshmeat.net/projects/slocate/

[53] D. L. Parnas and M. Dragomiroiu, "Module Interface Documentation – Using the Trace Function Method (TFM)," 2006, submitted to *IEEE Trans. Softw. Eng.*

[54] W. Bartussek and D. L. Parnas, "Using traces to write abstract specifications for software modules," in *Proc. 2nd Conf. European Cooperation in Informatics*, ser. LNCS 65. Springer-Verlag, 1978, pp. 211–236.

[55] S. J. Prowell, "Developing black box specifications through sequence enumeration," in *SESD '99: Proceedings of the Science and Engineering for Software Development: A Recognition of Harlan D. Mills' Legacy*. Washington, DC, USA: IEEE Computer Society, 1999, p. 14.

[56] S. J. Prowell and J. H. Poore, "Foundations of sequence-based software specification," *IEEE Trans. Softw. Eng.*, vol. 29, no. 5, pp. 417–429, 2003.

[57] S. Andler, "Predicate path expressions," in *POPL '79: Proceedings of the 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. New York, NY, USA: ACM Press, 1979, pp. 226–236.

[58] M. Broy and I. Krüger, "Interaction interfaces - towards a scientific foundation of a methodological usage of message sequence charts," in *ICFEM '98: Proc. 2nd IEEE Int. Conf. on Formal Engineering Methods*. IEEE Computer Society, 1998, pp. 2–13.

[59] D. K. Peters, M. Lawford, and B. T. y Widemann, "Software specification using tabular expressions and OMDoc," 2007, Work in progress.

[60] M. Kohlhase, *OMDoc — An Open Markup Format for Mathematical Documents [V. 1.2]*, ser. LNAI 4180. Berlin, Germany: Springer, 2006.

[61] G. H. Walton and J. H. Poore, "Measuring complexity and coverage of software specifications," *Information and Software Technology*, vol. 42, pp. 859–872, 2000.

[62] M. Cebrián, M. Alfonseca, and A. Ortega, "Common pitfalls using the normalized compression distance: What to watch out for in a compressor," *Comms. Info. Sys.*, vol. 5, no. 4, pp. 367–384, 2005.

Proceedings

Industrial Workshop

(SEKE 2007)

Workflow Management and Service Oriented Architecture

Theodorich Kopetzky Dirk Draheim
SCCH - Software Competence Center Hagenberg, Softwarepark 21, A-4232 Hagenberg, Austria
E-mail: theodorich.kopetzky@scch.at, dirk.draheim@scch.at

Abstract

The potential of automatization is still huge in modern enterprises. Flexibility of business processes has always been and still is considered as an important success factor for enterprises. Therefore, major information technology vendors currently focus with their initiatives on flexibility of information systems and business processes. Recently, we have seen two major trends in enterprise application architecture that address this issue from a different perspective: business process execution with its several concrete initiatives like workflow management and business process management and service-oriented architecture. But how to exploit the promises of service-oriented architecture in a workflow-intensive information system scenario? How to implement workflow logic in a service-oriented manner? In this presentation we want to discuss tensions between the two paradigms and possible best practices in bringing them together in a multi-tier system design. We present our lessons learned from a three-year project with a major insurance company. In this project we had to deal with a lot of innovative aspects like online-/offline workflow support and advanced data synchronization techniques.

1. Introduction

In a nearly finished project with the Austrian social insurance company for occupational risks, the AUVA[1] (as customer) and two research and development institutions ventured into the development of a new software system supporting the business processes of the prevention department of the AUVA.

This paper will briefly present in section 2 the context the AUVA operates in and list the main project goals.

In section 3 on the next page we will give a short survey of the challenges the team encountered during the advance of the project.

Finally, we present the most important lessons learned (in section 4 on the following page).

[1]AUVA is short for ***A****llgemeine* ***U****nfall****v****ersicherungs****a****nstalt* [1]

2. Domain and Project Goals

The AUVA has more than 5000 employees organized in different units and covers the occupational risks for approximately 3 million employees and 1.3 million pupils and students (numbers as of [4]). It operates rehabilitation and trauma centres for roughly 300.000 people involved in occupational accidents each year. Additionally each year the AUVA pays 73.000 compensations for victims (or their families) of occupational accidents or occupational diseases. These numbers are a strong motivation to prevent occupational accidents and diseases. In order to support prevention the AUVA employs prevention consultants who are experts in a specific domain, for example noise control. These consultants visit workplaces and suggest possible improvements of the working conditions to reduce the risk of occupational accidents and diseases. Visits of consultants can be requested by the companies (e.g., when a new machine begins operation) or can be initiated by the AUVA. Additionally, the AUVA is bound by law to serve companies in this regard. The AUVA operates several offices throughout the country each of which is responsible for one or more provinces.

2.1. Project Goals

For the purpose of complying with their obligations and improving their service regarding coordination of consultants, faster delivery of results etc. – to name just a few – different project goals have been identified. The following list states some of the most important projects goals.

- Unify the business processes of the different offices. Each office handles, for example, the core prevention process slightly different, so the resulting system should unify the process as far as possible and still tolerate small deviations of the process.
- Integrate old host data. The AUVA uses some databases covering nearly 40 years of data. The new system must integrate this data by migrating it to current database technology.

- Integrate with an existing system from another department. This department already uses software based on Oracle to support their business processes. The new system must integrate with this system and give an integrated view of the different activities happening in both systems.
- Use web service technology. This was a requirement of the information management department.
- Enable offline support for the prevention consultants. As the consultants visit companies they do not have always network connectivity to the AUVA network. Necessary data has to be available on the notebook. Support of part of the business processes must be possible offline as well.
- Good security regarding workflow and sensitive patient information, online and offline as well.

3. Challenges

Looking at the goals some challenges are obvious and some are not so.

One of the early challenges was to find a commercial platform as basis for the system. Evaluations of Microsoft Biztalk Server 2004 [3], currently in the version 2006, and DOMEA ([2], then a product of SER eGovernment Deutschland GmbH, now Open Text Corporation) and a strong preference of the customer for Microsoft solutions led to a prototypical implementation of the business process using the BizTalk-Server. Among other things problems with the performance even with very few users of the prototype caused the decision to discard the BizTalk approach. The new approach was to use IIS and to implement the web services in C#.

Another challenge was the support for working offline and offline workflow. An early assumption in the project was that the aforementioned Oracle Database would be replaced in the near future with a Microsoft SQL Server so the decision was made to utilize Microsoft Desktop Engine (MSDE) as offline database. Later on it became clear that the assumption regarding the MS SQL Server was wrong. A cross-database replication mechanism was not readily available then, thus a replication concept for Oracle and MSDE replications has been elaborated.

Another challenge was the lack of an existing security system. This was covered by the implementation of a company-wide security system.

4. Lessons Learned

Besides lessons learned regarding requirements engineering [5], lessons were learned regarding the migration of old databases (e.g., meaning of fields changed on given dates in the past), the integration of different application on a data level (complex interaction between involved parties), the object-relational mapping of a complex database already in use (already existing relations and constraints proved being difficult for mapping), the transportation of business objects via web services (object graphs versus hierarchical XML data structures), the transportation of state (avoidance of long running transactions, transfer of big objects for current state of workflow), and more.

5. Acknowledgements

The authors gratefully acknowledge support by the Austrian Government, the State of Upper Austria, and the Johannes Kepler University Linz in the framework of the K*plus* Competence Center Program. The authors would especially like to thank all project team members for their perseverance during this project.

References

[1] AUVA: Austrian social insurance company for occupational risks. `http://www.auva.at/`, visited April 2007.
[2] DOMEA. `http://www.domea.com`, visited April 2007.
[3] Microsoft BiztTalk Server 2006. `http://www.microsoft.com/germany/biztalk/default.mspx`, visited April 2006.
[4] AUVA Jahresbericht 2005, 2005. `http://www.auva.at/mediaDB/116926.PDF`, visited April 2007.
[5] M. Pichler, H. Rumetshofer, and W. Wahler. Agile requirements engineering for a social insurance for occupational risks organization: A case study. In *RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*, pages 246–251, Washington, DC, USA, 2006. IEEE Computer Society.

Implementing Agile Development - More than Changing Methodology

Chuck Fredrick, Douglas County Government, CO

Abstract—With the increasing pressure from business units to produce more with less -- and faster -- IT departments are looking for the latest "silver bullet" – the one big change that can reap the most low-hanging fruit. Agile Development in its many flavors can certainly appear on the surface to be just such a solution. But chances are, what made a development process slow to begin with was a series of small missteps across many or all of the domains and disciplines of software development. It only stands to reason that the solution will require a series of small corrections. However, it is all to frequent that IT Managers approach Agile Development by attempting to change only the big, tangible aspects of their software development engine – changes in process, tools, or terminology. But in software development, real change is hard, and requires a holistic view. The paper that follows explains one approach taken by an IT department's management to implement its first Agile project.

I. INTRODUCTION

When the Douglas County CO Government IT Department was asked to create a custom application for the Sheriff's Office to track and manage the County's resident convicted sex offenders, the project estimates using the traditional waterfall-based methodology proved too long and too costly to gain approval from its IT Steering Committee. Rather than cancel the project or purchase a less optimal off-the-shelf product, we implemented it as our first Agile/Scrum project. The outcome was a very successful system implementation delivered in four months - less than half the original schedule estimate. But the journey was equally rewarding, and taught us that "Agile" meant more than just a change to the project management methodology. For us, it meant changing almost everything.

To give ourselves the best opportunity for success on this first agile project, we explored and changed almost every facet of how we developed software: which project we chose to first attempt agile development, how we estimated project size, how we staffed the project, how interactions between team members should occur, which technologies we used, and how we sold the project to our customer, our team, and our IT Steering Committee. Most parts of the first agile plan went as designed; other aspects could have gone better. Key lessons learned on the project were centered on simplifying process and design approaches, as well as improving project communication and team dynamics.

II. PLANNING FOR SUCCESS

A. Choice of Project

We took great care in choosing this particular project as our first attempt at Agile. The project was near ideal to us for a few key reasons. One, the system did not require integration with many other existing systems. This was beneficial in that we were not injecting changes to other systems that would require re-coding or re-testing. Two, the business rules were not complex. Unlike other systems in the County, the business rules for the application were easy to understand, and there was a degree of latitude in how we fulfilled the user's need. (By comparison, the County's system that calculates residents' Certificate of Taxes Due, has very complex business rules where one and only one calculation is correct.) This reduced the risk that developed functionality would have to be reworked upon customer review. Three, the size of the application was smaller compared to other County systems, but large enough to uncover any issues we would face if we chose to apply this methodology to future projects.

B. Trust

Our IT Steering Committee was initially (and rightfully) wary of our pitch that we could reduce the project timeline in half with the implementation of a new methodology. Their experience taught them that IT groups such as ours would promise "big changes" with these new approaches to projects, but in reality would produce little in tangible results. To overcome their reluctance to our approach, the IT management team took an educated gamble – we offered a guarantee. If, after two months of our four month project, the customer was not happy with our progress, *we* would recommend to the Steering Committee that the project be cancelled. While we knew this would increase the pressure on our project team for results, this tactic helped us gain approval from the Steering Committee. And after receiving a round of applause from our customer at the month two demo, it was clear that we had begun to win some trust in our approach.

C. *Staffing*

Determining *who* would work on the project turned out to be as important as any process or technology decision we made. We estimated that the size of the project would require two developers for design and coding, a QA analyst for testing, and a Scrum Master, who managed the project. The team members were hand-picked not just because they were technically astute, but also because we believed they could absorb change quickly, be comfortable working under some ambiguous circumstances, and had some flexibility to work extra hours, if needed, to recover from mistakes along the way. Perhaps the most important decision was staffing the project's Scrum Master. We chose the Software Engineering Lead for this role, rather than a traditional project manager, as he had previous agile experience and could adapt the process on the fly, was well suited to provide technical guidance, and could relate to the developer's concerns, who were implementing new processes and technologies under the gun of the project schedule

D. *Team Dynamics*

One of our best decisions was to co-locate the team for the duration of the project. We temporarily moved the developers and QA analyst into a workroom. While seemingly a small detail, the project team is united in its belief that the resulting increase in communication directly reduced the schedule. Before the project, we expected that the bigger change would be that the Scrum process allowed us to "build a little, test a little," and give us a big schedule lift from our waterfall-based project methodology. Although hard to quantify, we are now certain that co-location was as big a key to success, especially on our first agile project where the constant communication cut down on confusion and wasted time.

E. *Estimation*

Two techniques new to us were particularly useful to estimate the project. The first was the creation an HTML mockup of the core application screens, developed by the team that investigated the project scope. The mockup confirmed high-level functionality and was a useful analogy for the Scrum team to use for the database schema and UI designs. The second technique was using story points and nonlinear estimation techniques, like those elaborated by Mike Cohn in his book *Agile Estimating and Planning* (Prentice Hall, 2005). An estimation scale where the gaps between values doubled each time (1, 2, 4, 8, 16 hours, etc.) proved valuable, and gave a realistic level of "accuracy" for complex or ill-defined story points.

F. *Design Simplicity*

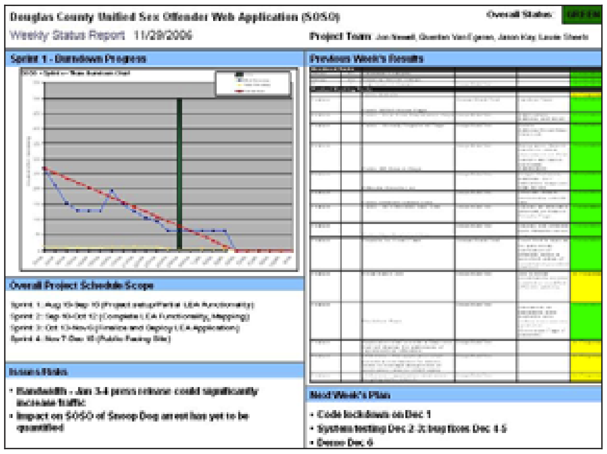
No matter the methodology used, there are upper limits to a developer's productivity. Studies have shown that for Java/J2EE development, a very productive developer can hand-code about 2,000 source lines of code (SLOC) per month[1]. A core strategy for this effort was to drive out unneeded complexity from the software architecture, and reduce the amount of hand-generated code. For example, we excluded the use of Enterprise Java Beans (EJB's) – a typical practice for other J2EE apps developed at the County, as the cost in developer time outweighed the benefits of such a heavy approach. In addition, we used Hibernate for the persistence layer of the architecture. Hibernate is an open source java-based tool that maps database relationships to an object-oriented domain model, and can significantly reduce development time otherwise spent with manual data handling in SQL. Previous project strategies required our developers to hand-code this tedious layer of the software architecture. The Scrum Master also chose to implement Google Map APIs for the application's mapping functionality. This required less code than using the county's traditional mapping solutions, and further reduced the overall application SLOC count. The final application was completed with approximately 25,000 SLOC, with over 8,000 SLOC generated through Hibernate. Given the two-developer, four month project, it is clear that the project would have extended by months without these choices being made.

G. *Project Management*

Tracking progress on an agile project would not have been possible using the county's traditional earned value management and waterfall-based work breakdown structures. The Scrum Master reported progress in the IT PMO's weekly project status meeting using a visual graph of the team's burn-down chart and progress towards that sprint's story points (see Figure 3). The approach simplified the data that

[1] David Consulting Group, *Comparative Sizing and Measurement is Critical to the Improvement of Software Application Development and Maintenance*, 2006 [copyright date], February 2, 2007: < http://www.davidconsultinggroup.com/pdfs/DCG%20Industry%20Data%20White%20Paper%2006%20PDF.pdf >

needed to be maintained to accurately gauge the project's progress, while being as (or perhaps more) informative, than the traditional project management measurements.



Douglas County Unified Sex Offender Web Application (SOSO) Overall Status: GREEN

Weekly Status Report 11/29/2006

Project Team: Jon Newell, Quentin VanEgmen, Jason Kay, Leslie Sheets

Sprint 1 - Burndown Progress

Previous Week's Results

[illegible]

Overall Project Schedule Scope

Sprint 1: Aug 13-Sep 10 (Project setup/Partial LEA functionality)
Sprint 2: Sep 10-Oct 12 (Complete LEA Functionality, Mapping)
Sprint 3: Oct 13-Nov 6 (Finalize and Deploy LEA Application)
Sprint 4: Nov 7-Dec 10 (Public Facing Site)

Issues/Risks

- Bandwidth - Jan 3-4 press release could significantly increase traffic
- Impact on SOSO of Snoop Dog arrest has yet to be quantified

Next Week's Plan

- Code lockdown on Dec 1
- System testing Dec 2-3; bug fixes Dec 4-5
- Demo Dec 6

Fig 3: A simple backlog and Burndown chart was maintained in MS Excel, and summarized in one MS PowerPoint slide to communicate the project's weekly status.

III. Retrospective

A. *Organizational Change*

We underestimated the anxiety that this move to Agile would produce within the IT organization. We communicated that the management team was driving this first attempt at agile development as a pilot only, and created and presented a "Scrum 101" presentation to share the approach. But staff members read *between* the lines of our communications and were apprehensive on the implications of agile development. Business Analysts were concerned that the closer relationship in an Agile project between developers and the customer would eliminate a need for their positions. Project Managers were concerned that they would need to be technical experts, as was our pilot's Scrum Master. IT Operations was concerned that creating constant builds would introduce configuration management issues. We could have spent more time one-on-one with each of these groups to better manage these concerns.

B. *Personal Change*

The move to Agile required more personal change for the project team members than we anticipated. Developing software in a more cyclic, feature-driven, less structured approach was a huge paradigm shift. Moving away from the comfortable patterns with which the team was most familiar, towards new approaches – even if we thought they were better – was initially viewed by them as risk, not opportunity. In hindsight, we could have prepared more to overcome their initial resistance to these changes.

IV. Conclusion

Our first attempt at an Agile project was driven top-down by management, and sold to our IT Steering Committee and project team members. By all measures, it met or exceeded our expectations. Our path to faster delivery was not a *revolutionary* move to a new methodology, but rather an *evolution* of many of our software development processes. Each change we made chipped away at the schedule, and fit together to cut the overall project timeline in half. Our advice to those driving change from the top: spend time evaluating all aspects of your project management and development practices for efficiencies, but do not underestimate the effect of those many changes on the project team members. The extra effort is sure to result in improving your project teams' speed and quality.

Appendix – About the Application

The Douglas County Government IT Department developed an application to (1) automate law enforcement processes to manage the county's convicted sex offender registry, and (2) publish the registry to the public. The application provides powerful management and search tools to all of the police jurisdictions within the county to sex offenders' files, and transfers them between jurisdictions (see Figure 2). County citizens can also view the most accurate sex offender registry available (see Figure 3), and sign up to receive email notifications if new convicted offenders move within a designated radius of their home, children's schools, or other places of interest. The public portion of the application is viewable at: http://www.douglas.co.us/apps/soso/initPublicIndex.do

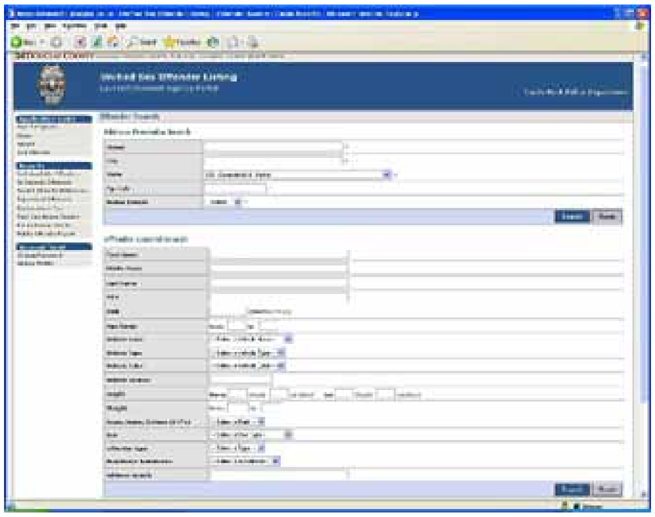


Fig 2 – Law Enforcement Agencies have a tool to unify the management of convicted sex offenders between jurisdictions. The search capabilities provide investigators with multiple ways to search for suspects among previously convicted sex offenders.

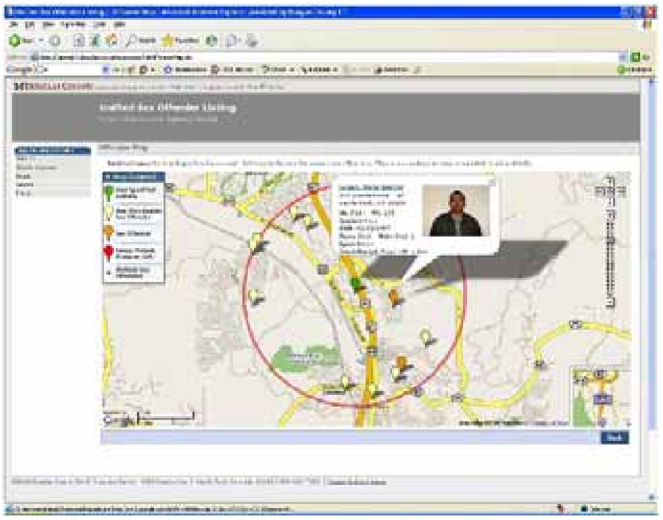


Fig 3 – Citizens can search for previously convicted sex offenders within a designated radius of any address in the county

Knowledge Modeling with UML

Anthony J. Rhem, Ph.D., CKM

A.J. Rhem & Associates, Inc.
500 North Michigan Ave., Suite 300, Chicago, Illinois 60611
Email: tonyr@ajrhem.com

Abstract:

Knowledge Modeling is the visualization of knowledge patterns. These patterns are discovered through knowledge acquisition. During the knowledge acquisition process the knowledge engineer works with the domain expert(s) to uncover and document the results of their meetings. To elicit feedback from the domain experts the knowledge engineer will develop certain knowledge models to visualize what has been learned from the domain expert(s).

Knowledge acquisition is one of the most difficult and error-prone tasks that a knowledge engineer does while performing knowledge modeling for the purpose of developing knowledge management systems. The cost and performance of the application depends directly on the quality of the knowledge acquired. An approach in which to capture knowledge more precisely is to use a standard notation for modeling knowledge, along with a standardized process or framework.

This presentation will demonstrate the use of applying the Unified Modeling Language (UML) as a standard notation and the Rhem-KAF (Knowledge Acquisition Framework) as a standard process to capture and build knowledge models. This presentation on knowledge modeling will focus on three (3) major representations of knowledge. These representations include Ladders, Network Diagrams, and Decision Trees. This presentation will examine the knowledge modeling and UML concepts and applying those concepts to build knowledge models with UML. A demonstration of the Rhem-KAF software will be incorporated within the presentation.

Attendee will learn the following:

- Types of Knowledge
- Methods for Knowledge Acquisition
- Constructing Knowledge Models
- Constructing Knowledge Models with UML
- Applying Knowledge Modeling – Sample Problem
- Validating Your Knowledge Models

Audience Expected Knowledge Level:

- In this seminar it is expected that the attendees be familiar with the Unified Modeling Language (UML).
- Interest in Knowledge Management Systems
- Interest and/or experience in building a Knowledge Based (Expert) Systems
- Interest and/or experience in Knowledge Engineering

Audience's Orientation:

- In this seminar it is expected that the attendees be performing in one or more of the following roles: Knowledge Engineer, System Analyst, or Business Analyst.

Sample Presentation Outline:

Knowledge Modeling ....

- Concepts...
- Instances...
- Processes (Tasks, Activities) ....
- Attributes and Values...
- Rules ....

Knowledge Modeling with UML ....

- UML Applied to Knowledge Models ....
  - Knowledge Use Case Model...
  - Knowledge Use Case Specification ....
  - Knowledge Production Use Cases ....
    - Information Acquisition...
    - Knowledge Claim Formulation...
    - Knowledge Claim Validation ....
  - Knowledge Integration Use Cases ....
    - Searching/Retrieving Stored Data, Information, or Knowledge...............
    - Broadcasting ....
    - Sharing ....
    - Teaching...
  - Knowledge Management Use Cases...
    - Leadership...
    - Building External Relationships ....
    - KM Knowledge Production ....
    - KM Knowledge Integration ....
    - Crisis Handling ....
    - Changing Knowledge-processing Rules...
    - Allocating Resources ....
- UML to Create Knowledge Models...
  - Concept Ladder...
    - Composition Ladder...
    - Decision Ladder ....
    - Attribute Ladder...
    - Process Ladder ....
    - Network Diagrams ....
- Concept Map...
- Process Map ....
- State Transition Network ....
- Decision Trees……………………………………………………………………

Speaker Bio:

Anthony J. Rhem is an Information Systems professional with over twenty-four (24) years of experience focused on implementation of major application systems. Specifically Anthony has been involved as a practitioner in A.I. (Artificial Intelligence)/Knowledge Based Systems and Knowledge Modeling since 1989. He has substantial experience performing Knowledge Acquisition and Knowledge Modeling. Some of his work in this area includes:

- Knowledge Based help desk application for customer inquiries on products.
- Knowledge Based System to aid computer operations in correcting system errors.
- Knowledge Based System to assist marketing in the advertising of product lines.
- Knowledge Based System to assist the underwriters in making a decision on mortgage loans.
- Knowledge Based System to assist in Tariff Pricing for shipments.
- Developing and executing comprehensive Knowledge Transfer Plans
- Developing and executing comprehensive Knowledge Management Strategies

Anthony is also an author and educator, presenting the application and theory of Artificial Intelligence, Knowledge Management, Business Process Re-engineering, Knowledge Acquisition, Requirements Analysis, Unified Modeling Language (UML) and Rational Unified Process (RUP). His seminars have been conducted in the United States, Europe and Korea and he continues his work in this area today.

Anthony's educational background includes a Ph.D. (honoris causa) from Calamus International University (2004) and a Ph.D. in Knowledge Management from Walden University (expected 2008), Masters of Science Degree in Information Systems with a concentration in Artificial Intelligence from DePaul University (1989), Bachelors of Science Degree in Management with a concentration in Marketing and Computer Science from Purdue University (1982), and is a Certified Knowledge Manager (CKM – Knowledge Management Institute).

Anthony is currently the Senior Partner-Chief scientist for A. J. Rhem & Associates Inc., IT System Integration and Training firm focusing on Knowledge Management, Business Rules Implementation and Software Methodologies. Anthony participates on the Industry Advisory Board – International Conference on Software Engineering and Knowledge Engineering (SEKE), Advisory Council - Boardroom Bound – Chicago Chapter, Entrepreneur Advisor Board Chairman, BDPA Thought Leaders. He is a member of The Technology Council of Advisors for the Gerson Lehman Group, member of the Advisory Board for American University Professional Science Master's Degree Program, Member of the National Science Foundation SBIR (Small Business Innovative Research) Review Panel and member of the Board of Directors and CIO of The RHEM Foundation.

Reviewers' Index

A
Silvia Teresita Acuna
Juan Carlos Augusto

B
Doo-Hwan Bae
Maria Teresa Baldassarre
Luciano Baresi
Sami Beydeda
Alessandro Bianchi

C
Danilo Caivano
Gerardo Canfora
Joao W. Cangussu
Christine W. Chan
W.K. Chan
Ned Chapin
Shu-Ching Chen
Panos Constantopoulos
Kendra Cooper

D
Jing Dong
Jin Song Dong
Schahram Dustdar

F
Behrouz Homayoun Far

G
Kehan Gao
Carlo Ghezzi
Holger Giese
Des Greer
Eric Gregoire
Paul Grunbacher

H
Xudong He
Rattikorn Hewett
Mei Hsing

K
Gabor Karsai
Axel Kupper

L
Tao Li
Claudia Linnhoff-Popien
Frank Lin
Xiaodong Liu
Yi Liu
Jian Lu
Zhongyu (Joan) Lu

M.
Antonio Mana
Hong Mei
Ali Mili
Rym Mili
Ana M. Moreno

N
Elisabetta Di Nitto

O
Mehmet Orgun

P
Massimiliano Di Penta

R

Marek Reformat
Robert Reynolds
George Roussos
Guenther Ruhe

S

Masoud Sadjadi
Remzi Seker
Naeem Seliya
Yidong Shen
Michael Shin
Nenad Stankovic
Kurt Stirewalt

T

Genny Tortora

V

Michael VanHilst
Sira Vegas

W

Qianxiang Wang
Yingxu Wang
Victor Winter
Guido Wirtz
Eric Wong

Y

Hongji Yang

Z

Cui Zhang
Zhi-Hua Zhou
Hong Zhu
Xingquan Zhu
Eugenio Zimeo
Andrea Zisman

Authors' Index

SEKE 2008 Call For Papers

The Twentieth International Conference on Software Engineering and Knowledge Engineering

Hotel Sofitel, San Francisco Bay, USA
July 1 - July 3, 2008

Organized by
Knowledge Systems Institute Graduate School

The Twentieth International Conference on Software Engineering and Knowledge Engineering (SEKE'08) will be held at Hotel Sofitel, Redwood City, California, USA, July 1-3, 2008.

The conference aims at bringing together experts in software engineering and knowledge engineering to discuss on relevant results in either software engineering or knowledge engineering or both. Special emphasis will be put on the transference of methods between both domains.

TOPICS

Solicited topics include, but are not limited to:
Agent architectures, ontologies, languages and protocols
Agent-based learning and knowledge discovery
Agent-based software engineering
Autonomic computing
Agent-based auctions and marketplaces
Adaptive Systems
Artificial Intelligence Approaches to Software Engineering
Artificial life and societies
Automated Reasoning
Automated Software Design and Synthesis
Automated Software Specification
Component-Based Software Engineering
Computer-Supported Cooperative Work
Data cleansing and noise reduction
Data streams and incremental mining
Data visualization
E-Commerce Solutions and Applications
Embedded and Ubiquitous Software Engineering
Electronic Commerce
Enterprise Software, Middleware, and Tools
Formal Methods
Human-Computer Interaction
Industry System Experience and Report
Integrity, Security, and Fault Tolerance
Interface agents
Knowledge Acquisition
Knowledge-Based and Expert Systems
Knowledge Representation and Retrieval
Knowledge Engineering Tools and Techniques
Knowledge Visualization
Learning Software Organization
Measurement and Empirical Software Engineering
Middleware for service based systems
Mobile agents
Mobile Commerce Technology and Application Systems
Mobile Systems
Multi-agent systems
Multimedia Applications, Frameworks, and Systems
Multimedia and Hypermedia Software Engineering
Ontologies and Methodologies
Patterns and Frameworks
Pervasive Computing
Process and Workflow Management
Programming Languages and Software Engineering
Program Understanding
Quality of services
Reflection and Metadata Approaches
Reliability
Requirements Engineering
Reverse Engineering
Runtime service management
Secure mobile and multi-agent systems
Semantic web
Service-centric software engineering
Service oriented requirements engineering
Service oriented architectures
Service discovery and composition
Service level agreements (drafting, negotiation, monitoring and management)
Smart Spaces
Soft Computing
Software Architecture
Software Assurance
Software Domain Modeling and Meta-Modeling
Software dependability
Software economics
Software Engineering Case Study and Experience Reports
Software Engineering Decision Support
Software Engineering Tools and Environments
Software Maintenance and Evolution
Software Process Modeling
Software product lines
Software Quality
Software Reuse
Software Safety
Software Security
Swarm intelligence
System Applications and Experience
Time and Knowledge Management Tools
Tutoring, Documentation Systems
Uncertainty Knowledge Management
Validation and Verification
Web and text mining
Web-Based Tools, Applications and Environment
Web-Based Knowledge Management
Web-Based Tools, Systems, and Environments
Web and Data Mining

CONFERENCE SITE (HOTEL INFORMATION)

The SEKE2008 Conference will be held at the Hotel Sofitel, Red Wood City, and San Francisco Bay, USA. The hotel has made available for these limited dates (7/1/2007 - 7/3/2007) to SEKE2008 attendees a discount rate of $89 US dollars for single/double, not including sales tax.

INFORMATION FOR AUTHORS

Papers must be written in English. An electronic version (Postscript, PDF, or MS Word format) of the full paper should be submitted using the following URL: http://conf.ksi.edu/seke08/submit/SubmitPaper.php. Please use Internet Explorer as the browser. Manuscript must include a 200-word abstract and no more than 6 pages of IEEE double column text (include figures and references). Workshop papers should be submitted to the workshops directly.

INFORMATION FOR REVIEWERS

Papers submitted to SEKE'08 will be reviewed electronically. The users (webmaster, program chair, reviewers...) can login using the following URL: http://conf.ksi.edu/seke08/review/pass.php.

If you have any questions or run into problems, please send e-mail to: seke@ksi.edu.

SEKE'2008 Conference Secretariat
Knowledge Systems Institute Graduate School
3420 Main Street
Skokie, IL 60076 USA
Tel: 847-679-3135
Fax: 847-679-3166
E-mail: seke@ksi.edu

IMPORTANT DATES

March 1, 2008	*Paper submission due*
April 1, 2008	*Notification of acceptance*
May 1, 2008	*Camera-Ready Copy*