PROCEEDINGS

SEKE 2006

The 18th International Conference on Software Engineering & Knowledge Engineering

Sponsored by

Knowledge Systems Institute Graduate School, USA

Technical Program

July 5 - 7, 2006

Hotel Sofitel, San Francisco Bay, California, USA

Organized by

Knowledge Systems Institute Graduate School, USA

ISBN 1-891706-18-7 (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'2006 Foreword

On behalf of the Program Committee of the Eighteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'2006), we would like to welcome you to San Francisco, USA. 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 is on the transference of methods between both domains.

It is our pleasure to announce that by the deadline of 15 March 2006, the main conference and three workshops received 219 submissions from 29 countries. All the papers have been rigorously reviewed by at least 3 members of the international Program Committee. Based on the review results, 99 papers have been accepted as regular papers, with an acceptance rate of 45%, and 46 accepted as short papers, with an acceptance rate of 21%. We would like to thank all the authors for their contributions.

This year, we have a rich collection of activities in the technical program, including three keynote speeches, one panel, three workshops, and 21 technical sessions. The keynotes, workshops, and technical sessions cover a wide range of topics in software engineering and knowledge engineering, including:

AI for Energy Production,
Aspect-Oriented Development,
Collaborative Systems,
Component-Based Systems,
Data Mining,
Databases,
Decision Support,
Empirical Software Engineering,
Formal Methods,
Knowledge Acquisition,
Multi-Agent Models,
Ontologies,
Programming Languages,
Rule-based Systems,
Software Architecture,
Software Development,
Software Evolution,
Software Testing,
UML and Modeling,
Web Services, and
Web and Workflow Management.

We are very grateful to the three keynote speakers, Ron Hira, Kuo-Wei Hwang, and Gordon Simpson; and workshop organizers, Christine W. Chan, Jerry Gao, Huaglory Tianfield, Wei-Tek Tsai, Hongji Yang, and Hong Zhu. The members of the Program Committee should be congratulated and specially thanked for their publicity effort and timely reviews of the submitted papers.

Finally, we would like to thank Shi-Kuo Chang for his guidance and leadership throughout organization of this conference. The assistance of the staff at KSI is also greatly appreciated. Special thanks go to Rex Lee, for his effective and efficient assistance in working with the paper submission and review system, which has made the review process smooth and timely.

Kang Zhang, University of Texas at Dallas, USA
George Spanoudakis, City University, UK
Giuseppe Visaggio, University of Bari, Italy

The 18th International Conference on Software Engineering & Knowledge Engineering (SEKE'2006)

July 5-7, 2006
Hotel Sofitel, San Francisco Bay, California, 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

Kang Zhang, *University of Texas at Dallas, USA*

Program Co-Chairs

George Spanoudakis, *City University, UK*
Giuseppe Visaggio, *University of Bari, Italy*

Program Committee

Silvia Teresita Acuña, *Universidad Autónoma de Madrid, Spain*
Anneliese Andrews, *Washington State University, USA*

Juan Carlos Augusto, *University of Ulster, UK*
Doo-Hwan Bae, *Computer Science Dept. KAIST, Korea*
Maria T. Baldassarre, *University of Bari, Italy*
Luciano Barezi, *Politechnico di Milano, Italy*
Alessandro Bianchi, *University of Bari, Italy*
Danilo Caivano, *University of Bari, Italy*
Gerardo Canfora, *University of Sannio, Italy*
Joao W. Cangussu, *University of Texas at Dallas, USA*
Giovanni Cantone, *University of Rome Tor Vergata, Italy*
Christine W. Chan, *University of Regina, Canada*
W.K. Chan Hong Kong, *University of Science and Technology, China*
Ned Chapin, *INforSci Inc, USA*
William Chu, *Tung Hai University, Taiwan*
Panos Constantopoulos, Athens *University of Economics & Business, Greece*
Maria F. Costabile, *University of Bari, Italy*
Kendra Cooper, *University of Texas at Dallas, USA*
Jin Song Dong, *National University of Singapore, Singapore*
Jing Dong, *University of Texas at Dallas, USA*
Yi Deng, *Florida International University, USA*
Carlo Ghezzi, *Politechnico di Milano Technical University, Italy*
Holger Giese, *Universitaet Paderborn, Germany*
Des Greer, *Queens University Belfast, UK*
Paul Grunbacher, *Johannes Kepler University Linz, Austria*
Peter In, *Korea University, Korea*
Dehua Ju, *ASTI Shangai, China*
Natalia Juristo, *Madrid Technological University, Spain*
Gabor Karsai, *Vanderbilt University, USA*
Taghi M. Khoshgoftaar, *Florida Atlantic University, USA*
Jun Kong, *University of Texas, USA at Dallas*
Huimin Lin, *Chinese Academy of Sciences, China*
Jian Lu, *Nanjing University, China*
Zhongyu (Joan) Lu, *The University of Huddersfield, UK*
Michael R. Lyu, *Chinese University of Hong Kong, Hong Kong*
Neil Maiden, *City University, UK*
Antonio Mana, *University of Malaga, Spain*
Hong Mei, *Beijing University, China*
Rym Mili, *University of Texas at Dallas, USA*
Sandro Morasca, *Universit degli Studi dell'Insubria, Italy*
Ana M. Moreno, *Technical University of Madrid, Spain*
Elisabetta Di Nitto, *Politecnico de Milano, Italy*
Mehmet Orgun, *Macquarie University, Australia*
Massimiliano Di Penta, *Universiy of Sannio, Italy*
Mario Piattini, *University Castilla-La Mancha, Spain*

Guenther Ruhe, *University of Calgary, Canada*
Walt Scacchi, *University of California Irvine, USA*
Tony Shan, *Wachoiva Bank, USA*
Yidong Shen, *Chinese Academy of Sciences, China*
Nenad Stankovic, *Univ of Aizu, Japan*
Kurt Stirewalt, *Michigan State University, USA*
Giancarlo Succi, *University of Bolzano, Italy*
Genny Tortora, *University of Salerno, Italy*
Jeffrey Tsai, *University of California Irvine, USA*
T. H. Tse, *University of Hong Kong, Hong Kong*
Sira Vegas, *Universidad Politcnica de Madrid, Spain*
Christiane Gresse von Wangenheim, *Universidade do Vale do Itaja, Brazil*
Guido Wirtz, *Bamberg University, Germany*
Claes Wohlin, *Blekinge Institute of Technology, Sweden*
Eric Wong, *University of Texas at Dallas, USA*
Baowen Xu, *Southeast University, China*
Hongji Yang, *De Montfort University, UK*
Huiqun Yu, *East China University of Science and Technology, China*
Du Zhang, *California State University, USA*
Hong Zhu, *Oxford Brookes University, UK*
Eugenio Zimeo, *Universiy of Sannio, Italy*
Andrea Zisman, *City University, UK*

Proceedings Cover Design

Gabriel Smith, *Knowledge Systems Institute Graduate School, USA*

Conference Secretariat

Judy Pan, *Chair, Knowledge Systems Institute Graduate School, USA*
Tony Gong, *Knowledge Systems Institute Graduate School, USA*
C. C. Huang, *Knowledge Systems Institute Graduate School, USA*
Rex Lee, *Knowledge Systems Institute Graduate School, USA*
Daniel Li, *Knowledge Systems Institute Graduate School, USA*

International Workshop on
Applications of Artificial Intelligence in Energy Production and Environmental Systems Engineering
(AAIEPESE'2006)

July 5-7, 2006
Hotel Sofitel, San Francisco Bay, California, USA

Organizers & Committee

Program Chair

Christine W. Chan, *University of Regina, Canada*

Program Co-Chair

Ni-Bin Chang, *Florida State University, USA*

Program Committee

Aijun An, *York University, Canada*
Nick Cercone, *Dalhousie University, Canada*
Amit Chakma, *University of Waterloo, Canada*
Keith Chan, *Polytechnical University, Hong Kong*
Gordon Huang, *University of Regina, Canada*
Raphael Idem, *University of Regina, Canada*
Raymond Jennings, *Simon Fraser University, Canada*
Yi-Tian Li, *Wuhan University, China*
C. Jim Lim, *University of British Columbia, Canada*
Wison Luangdilok, *Westinghouse Electric, USA*
Paitoon Tontiwachwuthikul, *University of Regina, Canada*
Xinhao Wang, *University of Cincinnati, USA*
Malcolm Wilson, *The Energy Innovation Network, Canada*
Zhifung Yang, *Beijing Normal University, China*
Guangming Zeng, *Hunan University, China*

International Workshop on
Agent-Oriented Software Development Methodology
(AOSDM'2006)

July 5-7, 2006
Hotel Sofitel, San Francisco Bay, California, USA

Organizers & Committee

Program Co-Chairs

Hong Zhu, *Oxford Brookes University, UK*
Huaglory Tianfield, *Glasgow Caledonian University, UK*
Hongji Yang, *De Montfort University, UK*

Program Committee

Michael Berger, *Siemens Corporate Technology, Germany*
Cornelia Boldyreff, *University of Lincoln, UK*
Massimo Cossentino, *Italian National Research Council, Italy*
Sheng-Uei Guan, *National University of Singapore, Singapore*
Xudong He, *Florida International University, USA*
Zhi Jin, *Academy of Mathematics and System Science, China*
Giuseppe A. Di Lucca, *Univ. of Sannio, Italy*
Carlos José Pereira de Lucena, *Pontificial Catholic University of Rio de Janeiro, Brazil*
Jiming Liu, *Hong Kong Baptist University, Hong Kong*
Xinjun Mao, *National University of Defence Technology, China*
Manish Parashar, *Rutgers University, USA*
David Riaño, *Universitat Rovira i Virgili, Spain*
Rainer Unland, *University of Essen, Germany*
Jianjun Zhao, *Fukuoka Institute of Technology, Japan*

International Workshop on
Evaluation and Evolution of Component Composition
(EECC'2006)

July 5-7, 2006
Hotel Sofitel, San Francisco Bay, California, USA

Organizers & Committee

Program Co-Chairs

Wei-Tek Tsai, *Arizona State University, USA*
Jerry Gao, *San Jose State University, USA*
Hong Zhu, *Oxford Brookes University, UK*

Program Committee

Mikhail Auguston, *Naval Postgraduate School, USA*
Fevzi Belli, *University of Paderborn, Germany*
F. B. Bastani, *University of Texas at Dallas, USA*
Mei-Hwa Chen, *University at Albany, State University of New York, USA*
Xiaoying Bai, *Tsinghua University, China*
Sami Beydeda, *Federal Finance Office, Germany*
Jean-Michel Bruel, *University of Pau, France*
Volker Gruhn, *University of Leipzig, Germany*
Mei Hong, *Bejing University, China*
Jingsha He, *Beijing University of Technology, China*
Yoshiaki Kakuda, *Hiroshima City University, Japan*
Michael R. Lyu, *The Chinese University of Hong Kong, Hong Kong*
Steve Roach, *The University of Texas at El Paso, USA*
Sahra Sedigh-Ali, *University of Missouri, USA*
Victor Winter, *University of Nebraska, USA*
Ye Wu, *Science Application International Corporation (SAIC),USA*

Table of Contents

Data Mining

Software Architecture

Verification & Decision Support

Databases

Software Development

Empirical SE

Component

Components and Evolution

Ontologies

Programming Languages

Security

Knowledge Acquisition

Formal Methods

Testing

Web Services

UML and Modeling

Web & Workflow Management

AAIEPESE Workshop

AAIEPESE'06

AOSDM Workshop

AOSDM-I: Multi-Agent Models

AOSDM-II: Agent-Oriented Development

EECC Workshop

EECC-I: Web Service Composition

EECC-II: Component-Based Systems

Note: (**S**) means short paper.

Keynote I: Outsourcing America

Professor Ron Hira

Companies are increasingly sourcing engineering labor globally. They are transferring tasks and jobs traditionally done by American engineers to lower-cost countries, where engineers earn as little as ten percent of a comparable American. Information Technology (IT) is the first industry to move overseas. A "Global Delivery Model," where companies include both an onshore and offshore component when bidding on work, has already become the norm. As a result, IT firms are shifting their workforce towards lower cost countries. While no government organization has reliable figures on exactly how many engineering tasks and jobs have moved to low-cost countries, observable trends indicate the process is accelerating in scale and scope. For example, IBM is projected to have 18% of its workforce in India by the end of 2006 up from just 2% in 2003.

These changes in firm behavior will have significant impacts on American engineers and the US innovation system. This talk will cover why companies are moving work overseas; how much is being outsourced; the types of jobs being outsourced; and, the impacts on developed and developing countries. It will also explore how engineers can adapt to the global technology labor market and the policy debates on outsourcing.

About Professor Ron Hira

Dr. Ron Hira is an Assistant Professor of Public Policy at Rochester Institute of Technology where he specializes in engineering workforce issues and technology policy. Ron is author of the book, Outsourcing America, published by the American Management Association. Outsourcing America has been named a finalist in the best business book category for this year's Benjamin Franklin Awards presented by the Publishers Marketing Association. In 2006 he received the IEEE-USA President's Special Citation Award "for furthering public understanding of an economic trend that has profound implications for the engineering profession through his book Outsourcing America."

Professor Hira is a recognized expert on offshore outsourcing and has testified before the US Congress twice on its implications. He has given over 70 invited talks on offshore outsourcing at universities, to policymakers, as well as to the general public. Ron is frequently quoted and interviewed in many major newspapers and magazines, and on television and radio. Previously Ron worked as a control systems engineer and program manager with Sensytech, NIST, and George Mason University (GMU). Dr. Hira has been a consultant to the U.S. Department of Treasury Executive Institute, Rand Corporation, National Research Council, Enterprise Integration Inc, Deloitte & Touche, General Motors and Newport News Shipbuilding.

Dr. Hira completed his Post-Doctoral Fellowship at Columbia University's Center for Science, Policy, and Outcomes. He holds a Ph.D. in Public Policy from GMU, an M.S. in Electrical Engineering also from GMU, and a B.S. in Electrical Engineering from Carnegie-Mellon University. Ron is a licensed professional engineer and is Vice President for Career Activities of IEEE-USA, the largest engineering professional society in America. In 2004, he was awarded the Citation of Honor from IEEE-USA for his work on behalf of the engineering profession. Ron participated on the Council on Foreign Relations' "Research Roundtable on Technology, Innovation and America's Primacy" and the Council on Competitiveness' "National Innovation Initiative."

Keynote II: Information Services in Service Oriented Architecture -- Challenges and Opportunities

Dr. Kuo-Wei Hwang

Innovative use of information creates new business value and reduces risks for an enterprise. The creation of consistent, reusable services around information is fast becoming a strategic focus amongst leading businesses. This talk looks at forces that are driving enterprise information services requirements, and how the confluence of the service oriented architecture approach along with recent trends in information management technologies is quickly becoming the technology underpinning in implementing information services solutions. Feedback and lessons learned from early SOA projects to surface trusted, integrated enterprise information will be discussed.

About Dr. Kuo-Wei Hwang

Dr. Kuo-Wei Hwang currently leads the Information Services for SOA initiative inside IBM. He directs an organization that is responsible for the development of Information Services architecture patterns, solutions, services offerings, as well as a competency center which works with enterprises and organizations around the world on the deployment of information management technology in service oriented architecture. Dr. Hwang is based in IBM Silicon Valley Lab in San Jose, California.

Dr. Hwang's professional career spans software R&D, international business management positions, and executive positions in the telecommunications industry. He joined IBM at the Santa Teresa laboratory in San Jose, California, in 1990. As a member of IBM Database Technology Institute he participated in the advanced technology transfer project which led to the launch of IBM's object-relational DBMS, DB2 UDB. From 1994 to 1996 he was on international assignment in Beijing, China, as the founding member of IBM's data management operation there. From 1997-2000 he was in Singapore as the regional business unit executive of Software Solutions and Network Computing Solutions divisions for IBM ASEAN/South Asia. He served as CTO and executive VP of Products and Services at MediaRing Inc. in Singapore from 2000-2002 before returning to IBM Silicon Valley Lab in the US.

Dr. Hwang received his Ph.D. in Computer and Information Sciences from the University of Minnesota.

Keynote III: A Pragmatic approach to Enterprise Services Orientation

Gordon Simpson

The green field advantages of Services and SOA are clear. However, for most large enterprises, the major impact of any service oriented approach must include so called "legacy systems". As we deliver the promised business agility, we must leverage the stability and richness of this area of significant IT investment. From the perspective of SAP, the industry leader in service oriented applications, this presentation will discuss the key issues of real world enterprise service architecture.

About Gordon Simpson

Gordon Simpson is a VP at SAP Labs with broad responsibilities within their ESA/BPP architecture strategies. Over the last decade, Gordon has held technology leadership roles with BEA Systems, Vitria Technology and IBM. He has been involved in all aspects of enterprise applications development since the mid-70s; designing and writing end-user applications as well as applications infrastructure while working with most computer "platforms" from punched cards to Enterprise Services/SOA.

Metamodel Access Protocols for Extensible Aspect-Oriented Modeling

Naoyasu Ubayashi
Kyushu Institute of Technology
Fukuoka, Japan
ubayashi@acm.org

Tetsuo Tamai
The University of Tokyo
Tokyo, Japan
tamai@acm.org

Shinji Sano, Yusaku Maeno, Satoshi Murakami
Kyushu Institute of Technology
Fukuoka, Japan
{sano,maeno,msatoshi}@minnie.ai.kyutech.ac.jp

Abstract

Aspect orientation is important not only at the programming-level but also at the modeling-level. We previously proposed an aspect-oriented modeling language called AspectM for managing modeling-level aspects. Although AspectM provides basic modeling facilities for a modeler, the language constructs cannot be extended. In this paper, we propose a mechanism called metamodel access protocol (MMAP) that allows an application modeler to access and modify the AspectM metamodel. MMAP consists of metamodel extension points, extension operations, and primitive predicates for defining pointcut designators. MMAP enables a modeler to represent application-specific crosscutting concerns.

1. Introduction

Aspect-oriented programming (AOP)[9][4] can separate crosscutting concerns from primary concerns. In major AOP languages such as AspectJ[10], crosscutting concerns including logging, error handling, and transaction are modularized as aspects and they are woven to primary concerns. AOP is based on join point mechanisms (JPM)[11] consisting of join points, a means of identifying join points (pointcut), and a means of semantic effect at join points (advice). In AspectJ, program points such as method execution are detected as join points, and a pointcut designator extracts a set of join points related to a specific crosscutting concern from all join points. A compiler called a weaver inserts advice code at the join points selected by pointcut designators.

Recently, aspect orientation has been proposed for coping with concerns at the early stages of the software development phases including requirements analysis, domain analysis, and architecture design phases. Aspects at the modeling-level are mainly applied to not behavior as in AOP but a static model structure such as UML diagrams.

We previously proposed a UML-based aspect-oriented modeling language called AspectM[19] for managing modeling-level aspects. Using AspectM, a modeler can represent crosscutting concerns without considering details of implementation languages and platform because a model can be translated into source code by the AspectM model compiler. Although AspectM provides major JPMs, there might be situations in which a modeler wants to define a JPM specific to an application. For example, a modeler wants to capture a group of methods that are targets of an application-specific logging or transaction. While some kinds of application-specific concerns can be represented by using UML profiling mechanisms, there are applications that need metamodel extension. However, current aspect-oriented modeling languages including AspectM do not allow a modeler to extend JPMs.

In this paper, we introduce a mechanisms called metamodel access protocol (MMAP) that allows an application modeler to access and modify the AspectM metamodel, an extension of the UML metamodel. The mechanism enables a modeler to define a new JPM that includes application-specific join points, pointcut designators, and advice.

The remainder of this paper is structured as follows. In Section 2, we demonstrate the necessity of application-specific extension. In Section 3, we introduce the concept of MMAP. In Section 4, we illustrate an implementation method. In Section 5, we introduce related work. Lastly, in Section 6, we conclude the paper.

JPM type	Join point type	Advice type
PA (pointcut & advice)	operation	before, after, around
CM (composition)	class	merge-by-name
NE (new element)	class diagram	add-class, delete-class
OC (open class)	class	add-operation, delete-operation add-attribute, delete-attribute
RN (rename)	class, operation, attribute	rename
RL (relation)	class	add-inheritance, delete-inheritance add-aggregation, delete-aggregation add-relationship, delete-relationship

Table 1. JPMs in AspectM

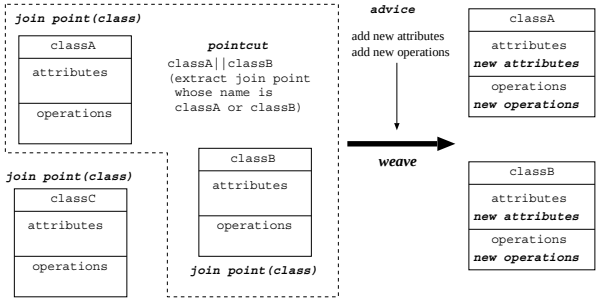
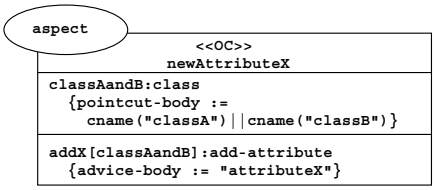


Figure 1. Modeling-level aspect orientation




```
<aspect name="newAttributeX"
        type="ordinary" jpm="OC">
  <pointcut name="classAandB" type="class">
    <pointcut-body>
      cname("classA")||cname("classB")
    </pointcut-body>
  </pointcut>
  <advice name="addX" type="add-attribute"
          ref-pointcut="classAandB">
    <advice-body>"attributeX"</advice-body>
  </advice>
</aspect>
```

Figure 2. AspectM notation

2. Motivation

We briefly excerpt an aspect-oriented modeling language AspectM from our previous work[19]. We also demonstrate the necessity of application-specific aspect orientation.

2.1. AspectM

2.1.1 Aspect orientation at the modeling-level

Although JPMs have been proposed as a mechanism at the programming-level, they can be applied to the modeling-level. Figure 1 shows an example of the modeling-level aspect orientation. In Figure 1, a class is regarded as a join point. The pointcut definition '`classA || classB`' extracts the two classes `classA` and `classB` from the three join points `class A`, `classB`, and `classC`. *Add new attributes* and *add new operations* are regarded as advice. In Figure 1, new attributes and operations are added to the two classes `classA` and `classB`. As shown here, aspects at the modeling-level are applied to not behavior as in AOP but a static model structure.

AspectM supports six kinds of modeling-level JPMs: PA (pointcut & advice as in AspectJ[10]), CM (composition as in Hyper/J[17]), NE (new element), OC (open class as in AspectJ inter-type declaration), RN (rename), and RL (relation). Figure 1 is an example of OC. Table 1 shows the outline of these JPMs.

2.1.2 Language features

AspectM provides the two kinds of aspects: an ordinary aspect and a component aspect. A component aspect is a special aspect used for composing aspects. An aspect can have parameters for supporting generic facilities.

Figure 2 shows the AspectM diagram notations and the corresponding XML formats. The notations of aspect diagrams are similar to those of UML class diagrams. A diagram is separated into three compartments: 1) aspect name and JPM type, 2) pointcut definitions, and 3) advice definitions. An aspect name and a JPM type are described in the first compartment. A JPM type is specified using a stereotype. Pointcut definitions are described in the second compartment. Each of them consists of a pointcut name, a join point type, and a pointcut body. In pointcut definitions, we can use three designators: `cname` (class name matching), `aname` (attribute name matching), and `oname` operation name matching). We can also use three logical operations: && (and), || (or), and ! (not). Advice definitions are described in the third compartment. Each of them consists of

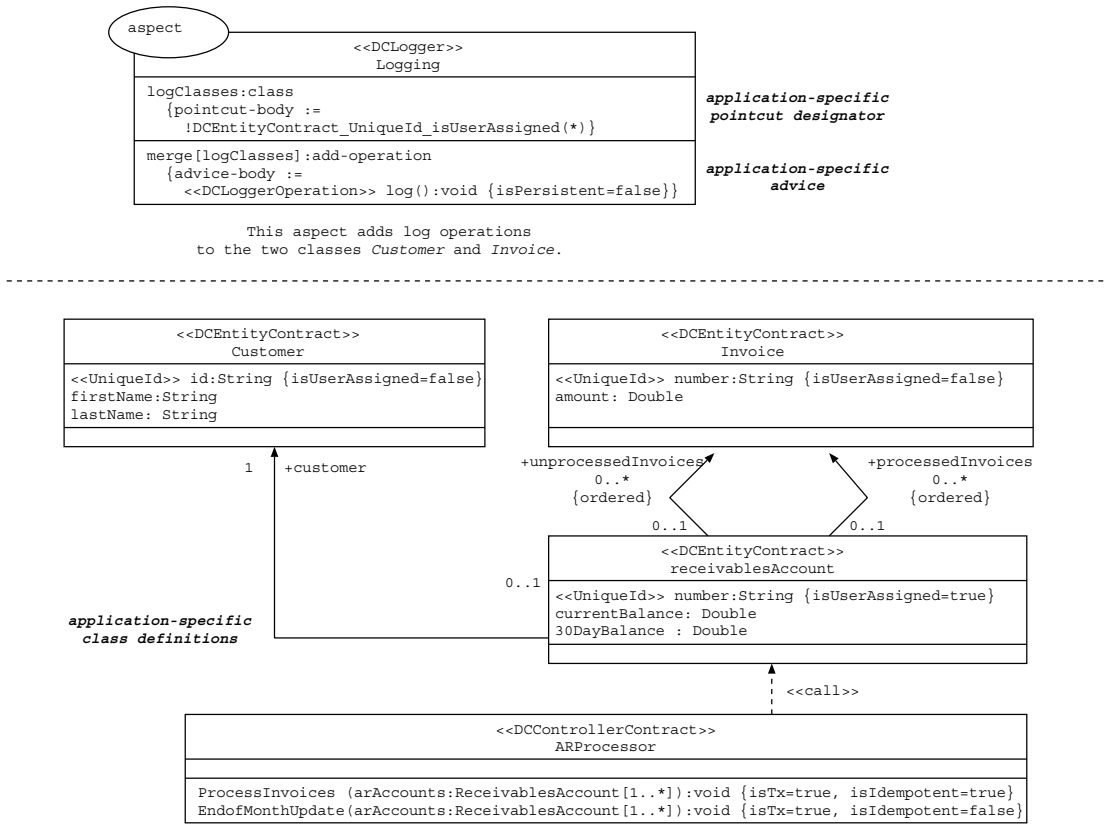


Figure 3. An example of application-specific model

an advice name, a pointcut name, an advice type, and an advice body. A pointcut name is a pointer to a pointcut definition in the second compartment. An advice is applied at join points selected by a pointcut.

2.2. Problems to be tackled

Although AspectM provides basic JPMs for aspect-oriented modeling, a modeler cannot define application-specific JPMs. It would be better for a modeler to describe a model as shown in Figure 3 (the class diagram is cited from [6]). Application-specific model elements are denoted by stereotypes that are not merely annotations but elements introduced by metamodel extension[1].

The model in Figure 3 describes an invoice processing system comprised of two kinds of application-specific distributed components: `DCEntityContract` for defining the contract of a distributed entity component and `DCControllerContract` for defining the contract of a distributed controller component. The model also includes an application-specific JPM `DCLogger` that adds log operations to `DCEntityContracts` whose `UniqueId` is not assigned by users. `DCLogger` consists of application-specific pointcut designators and advice. `DCEntityContract` can be regarded as an application-specific join point.

It is not necessarily easy to describe aspects such as *DCLogger* by using stereotypes as merely annotations because associations among stereotypes cannot be specified. Without extending a metamodel, we cannot specify the following fact: *DCEntityContract* must have a *UniqueId* whose tag is *isUserAssigned*. If we define an aspect based on fragile stereotypes that lack consistency, the aspect might introduce unexpected faults because the aspect affects many model elements. It is necessary to adopt metamodel extension for describing application-specific models precisely.

In this paper, we propose a method for constructing this kind of application-specific models by introducing the concept of MMAP. There are many situations that need application-specific JPMs—for example, application-specific logging as in Figure 3, application-specific resource management, application-specific transaction, and so on.

[1] A stereotype in the first compartment of an aspect diagram indicates an application-specific JPM such as *DCLogger*. On the other hand, stereotypes attached to other model elements—join points—such as classes and attributes indicate application-specific properties such as *DCEntityContract*. We can use these stereotypes to capture a set of related join points.

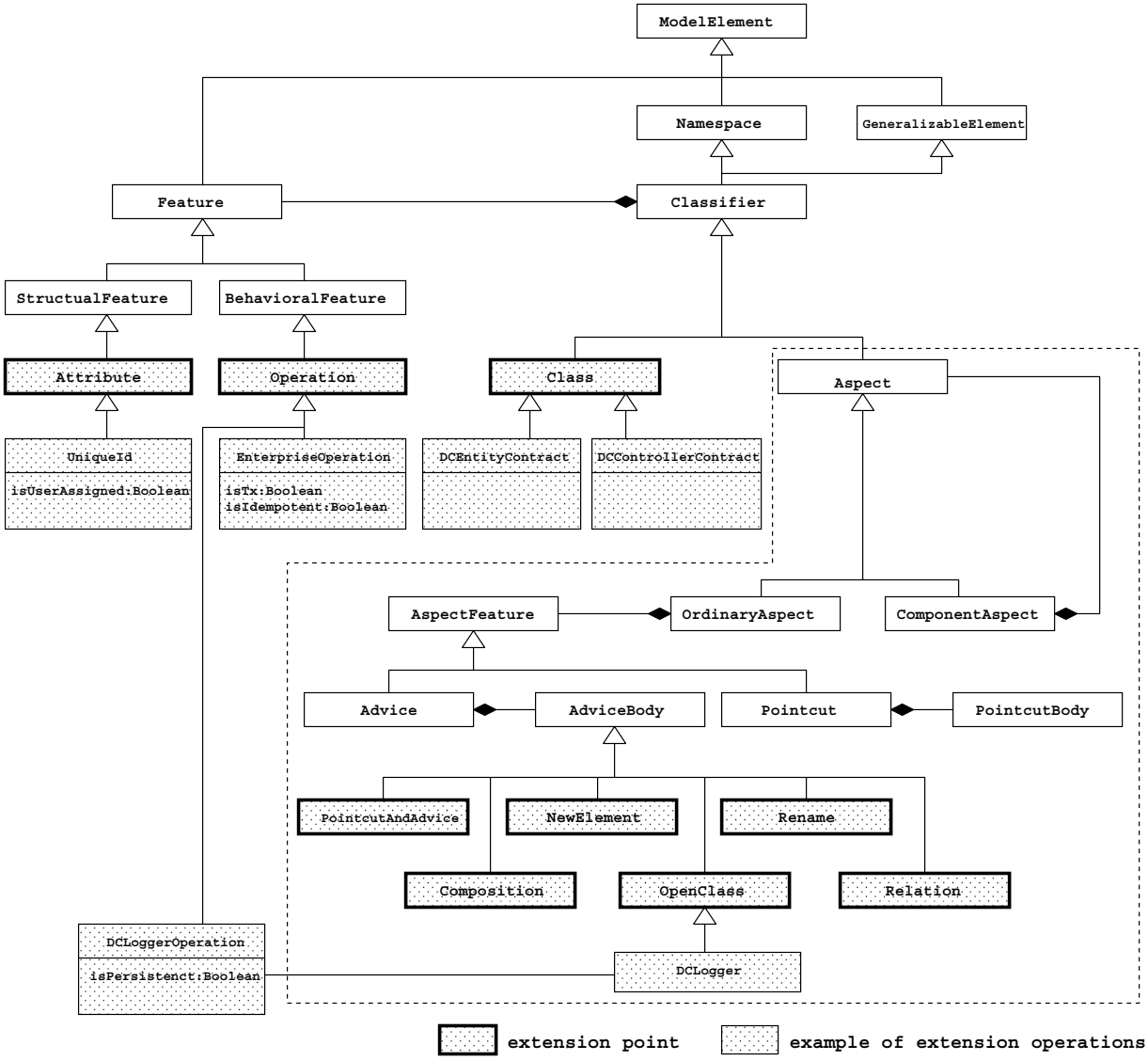


Figure 4. AspectM metamodel

3. Metamodel access protocol

3.1. Background

There are two approaches for extending model elements in UML. The first is a lightweight approach using UML profiles. The second is a heavyweight approach that extends the UML metamodel by using MOF (Meta Object Facility), a language for modeling a metamodel.

UML profiling mechanisms use stereotypes for introducing rich vocabularies such as *UML Profiles for Enterprise Distributed Object Computing (EDOC)*. While an application modeler can easily introduce some kinds of application-specific JPMs—for example, pointcut definitions using stereotypes such as `UniqueId`—, there are situations in which stereotypes as a mere annotation mechanism are insufficient: the typing of tags is weak; and we cannot declare new associations among UML metamodel elements[6]. For example, as explained in 2.2, stereotypes as annotations cannot describe relations among application-specific model elements including `DCEntityContract`, `DCControllerContract`, `UniqueId`, and `DCLogger`: the relations among them are described in the metamodel as shown in Figure 4.

On the other hand, the MOF approach is very strong because all of the metamodel elements can be extended. However, it is not easy for an application modeler to extend the UML metamodel by using the full power of the MOF.

MMAP proposed in this paper is a middle weight approach that restricts available extension by MOF. MMAP aims at introducing an application-specific JPM that cannot be described by UML profiling mechanisms. An application modeler can access and modify the AspectM metamodel by using protocols exposed by MMAP. The AspectM metamodel is an extension of the UML metamodel. The target of MMAP is not a tool developer that needs full access to the AspectM metamodel but an application modeler that wants to introduce rich vocabulary at small cost. Although model elements introduced by MMAP are denoted by stereotypes for convenience, these stereotypes are not merely annotations.

The idea of MMAP originates in the mechanisms of extensible programming languages, such as metaobject protocol(MOP)[8] and computational reflection[15][12]. We think that the idea of MOP is useful at the modeling-level although the targets of MOP are different from those of MMAP: the former focuses on execution behavior, and the latter focuses on model structures.

3.2. Metamodel

The AspectM metamodel is defined by modifying the UML metamodel. OMG defines the four-level metamodel hierarchy consisting of M0, M1, M2, and M3: M0 contains the data such as object instances; M1 contains application models described by a modeling language such as UML; M2 contains metamodels that define modeling languages; M3 contains metametamodels that define metamodels in M2. For example, a class diagram and the UML metamodel reside at the level M1 and M2, respectively. MOF resides at the level M3.

Figure 4 shows the part of the AspectM metamodel and an example of extension with MMAP. This metamodel resides at the level M2. By introducing this metamodel, we can define an aspect in a UML diagram that resides at the M1 level.

3.3. Protocols

MMAP is exposed for a modeler who creates a model at the M1 level to access the AspectM metamodel at the M2 level. MMAP is comprised of extension points, extension operations, and primitive predicates for navigating the AspectM metamodel. An extension point is an AspectM metamodel element that can be extended by inheritance. An extension operation is a modeling activity allowed at the exposed extension points.

Table 2 and 3 show protocols exposed by MMAP. The former is a list of extension points and operations. The latter is a list of primitive predicates that can navigate the AspectM metamodel and define pointcut designators.

Extension point	Extension operation
Class	define subclasses, add attributes to subclasses, create associations among subclasses
Attribute	the same as above
Operations	the same as above
PointcutAndAdvice (PA)	the same as above
Composition (CM)	the same as above
NewElement (NE)	the same as above
OpenClass (OC)	the same as above
Rename (RN)	the same as above
Relation (RL)	the same as above

Table 2. Extension points and operations

Predicate	Explanation
meta-class-of(mc, c)	*mc* is a metaclass of *c*
member-of(m, c)	*m* is a member of a class *c*
value-of(v, a)	*v* is value of an attribute *a*
super-class-of(c1, c2)	*c1* is a superclass of *c2*
related-to(c1, c2)	*c1* is related to *c2*

Table 3. Primitive predicates for defining pointcut designators

The design of MMAP is similar to that of application frameworks in which hot-spots should be exposed. By using MMAP, an application modeler need not to redefine the AspectM metamodel. The modeler has only to extend these hot-spots by using protocols.

3.3.1 Extension points and operations

We can extend the AspectM metamodel as shown in Figure 4 by applying extension operations such as *define subclasses* / *add attributes to the subclasses* at the exposed extension points including `Class`, `Attribute`, `Operation` and `OpenClass`. By adopting this extended metamodel, we can describe an application-specific UML model as shown in Figure 3.

This metamodel introduces the following application-specific model elements: application-specific join points including classes such as `DCEntityContract` and `DCControllerContract`, an attribute `UniqueId`, and operations such as `DCLoggerOperation`; an application-specific advice `DCLogger` that can add an application-specific operation `DCLoggerOperation` to application-specific join points.

In our MMAP, `Advice`/`AdviceBody` are not exposed as extension points because this extension need new weaver modules that can handle new kinds of advice. It is hard for

a modeler to develop a new weaver module. On the other hand, it is relatively easy to extend or synthesize existing advice mechanisms including PA, CM, NE, OC, RN, and RL. For these reasons, our MMAP does not allow a modeler to extend `Advice`/`AdviceBody`.

Although other metamodel elements such as association might be candidates of extension points, they are not included in our MMAP because their usefulness for application-specific extension is not yet clear.

3.3.2 Predicates for pointcut designators

A modeler does not have to extend `Pointcut` or `PointcutBody` because pointcut designators can be defined by composing primitive predicates shown in Table 3. The following is a definition of application-specific pointcut designator appearing in Figure 3:

```
define pointcut
 DCEntityContract_UniqueId_isUserAssigned(c):
  meta-class-of("DCEntityContract", c) &&
  member-of(a, c) &&
  meta-class-of("UniqueId", a) &&
  member-of("isUserAssigned", "UniqueId") &&
  value-of("true", "isUserAssigned")
```

This pointcut designator selects all classes that match the following conditions: 1) the metaclass is `DCEntityContract`; 2) the value of the `isUserAssgned` is true. In case of Figure 3, the negation of this pointcut designator selects the two classes `Customer` and `Invoice`. It is necessary to distinguish quantification of pointcut designators from that of predicates. The latter is existential. A defined pointcut designator represents all elements that satisfy the right-hand side predicates.

By using the primitive predicates, we can define built-in pointcut designators as follows:

```
define pointcut
 cname(c):
  mata-class-of("Class", c) &&
  member-of("Name", "Class") &&
  value-of(c, "Name")
```

4. Implementation

We have developed a prototype tool for supporting AspectM. The tool consists of a model editor and a model compiler. The model editor provides facilities for editing UML and aspect diagrams. The model editor is developed using Eclipse Modeling Framework (EMF)[5], a tool that generates a model editor from a metamodel. The model editor can save diagrams in the XML format. The model compiler is implemented as an XML transformation tool because UML class diagrams can be represented in the XML format.

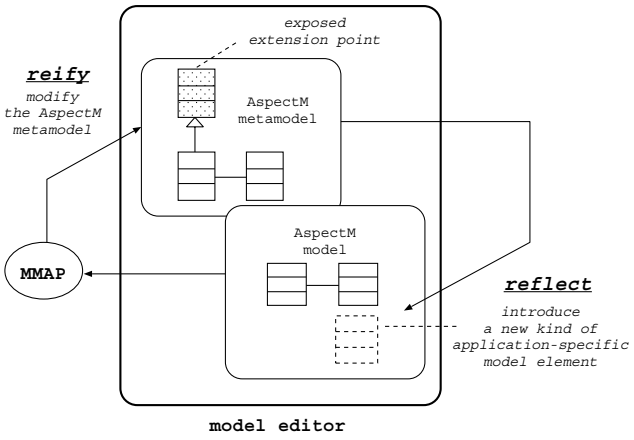


Figure 5. Reflective model editor

Our prototype tool includes the core part of MMAP. The AspectM metamodel and application models are transformed to a set of Prolog predicates. We can access and modify these model elements by using predicates exposed by MMAP.

We also plan to develop a reflective model editor that allows a modeler to not only edit application models but also extend, modify, and customize the AspectM metamodel. Figure 5 illustrates the concept of a reflective model editor. This idea is similar to an edit-time metaobject protocol (ETMOP)[3] proposed by A.D.Eisenberg and G.Kiczales. ETMOP runs as part of a code editor and enables metadata annotations to customize the rendering and editing of code. Using the AspectM reflective model editor, a modeler can edit a model such as Figure 3 by extending the AspectM metamodel as shown in Figure 4.

5. Related work

Extensible AOP languages can consolidate the advantages of domain-specific and general-purpose languages because programmers can customize language features. H.Masuhara and G.Kiczales defined a three-part modeling framework that explains a common structure in different major JPMs such as PA, CM, and OC[13]. Based on the three-part modeling framework, N.Ubayashi et al. proposed a parameterized interpreter that takes several parameters to cover different JPMs[18]. The interpreter is helpful in rapid-prototyping a new AOP mechanism or a reflective AOP system that supports different mechanisms. M.Shonle, K.Lieberherr, and A.Shah proposed an extensible domain-specific AOP language called XAspect that adopts plug-in mechanisms[14]. E.Tanter and J.Noye proposed a versatile kernel for multi-language AOP[16]. S.Chiba et al. proposed `Josh`[1] that allows programmers to define a new pointcut designator as a boolean function. AspectM covers major

JPMs including PA, CM, and OC. Moreover, application-specific AOP features can be introduced by using MMAP proposed in this paper.

Domain-specific aspect orientation is important not only at the programming stage but also at the modeling stage. An approach for supporting domain-specific aspect-oriented modeling is proposed by J.Gray[7]. Logic programming facilities and queries using these facilities will be useful for defining domain-specific pointcuts[20]. M.Eichberg, M.Mezini, and K.Ostermann investigated the use of XQuery for specification of pointcuts[2]. Since a domain can be considered as a set of application families, MMAP is useful for domain-specific aspect orientation by defining JPMs common to related applications.

6. Conclusion

In this paper, we proposed MMAP that enables a modeler to define a new JPM that represents application-specific crosscutting concerns. Although the current MMAP might need refining, we believe that our approach is the first step towards extensible aspect-oriented modeling based on metamodel access protocols.

References

[1] Chiba, S. and Nakagawa, K.: Josh: An Open AspectJ-like Language, In *Proceedings of International Conference on Aspect-Oriented Software Development (AOSD 2004)*, pp.102-111, 2004.

[2] Eichberg, M., Mezini, M., and Ostermann, K.: Pointcuts as Functional Queries, In *Proceedings of International Conference on Asian Symposium (APLAS 2004)*, pp.366-382, 2004.

[3] Eisenberg, A. and Kiczales, G.: A Simple Edit-Time Metaobject Protocol, *Workshop on Open and Dynamic Aspect Languages (OAL) (Workshop at the 5th International Conference on Aspect-Oriented Software Development (AOSD 2006))*, 2006.

[4] Elrad, T., Filman, R.E. and Bader A.: Aspect-oriented programming, *Communications of the ACM*, vol.44, no.10, pp.29-32, 2001.

[5] EMF, http://www.eclipse.org/emf/.

[6] Frankel, D. S.: *Model Driven Architecture — Applying MDA to Enterprise Computing*, John Wiley & Sons, Inc., 2003.

[7] Gray, J., Bapty, T., Neema, S., Schmidt, D., Gokhale, A, and Natarajan, B.: An Approach for Supporting Aspect-Oriented Domain Modeling, In *Proceedings of International Conference on Generative Programming and Component Engineering (GPCE 2003)*, pp.151-168, 2003.

[8] Kiczales, G., Rivieres, J.des , Bobrow, D. G.: The Art of the Metaobject Protocol, MIT Press, Cambridge, MA, 1991.

[9] Kiczales, G., Lamping, J., Mendhekar A., Maeda, C., Lopes, C., Loingtier, J. and Irwin, J.: Aspect-Oriented Programming, In *Proceeding of European Conference on Object-Oriented Programming (ECOOP'97)*, pp.220-242, 1997.

[10] Kiczales, G., Hilsdale, E., Hugunin, J., et al.: An Overview of AspectJ, In *Proceedings of European Conference on Object-Oriented Programming (ECOOP 2001)*, pp.327-353, 2001.

[11] Kiczales, G.: The Fun Has Just Begun , *Keynote talk at AOSD2003*, 2003.

[12] Maes, P.: Concepts and Experiments in Computational Reflection, In *Proceedings of Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'87)*, pp.147-155, 1987.

[13] Masuhara, H. and Kiczales, G.: Modeling Crosscutting in Aspect-Oriented Mechanisms, In *Proceedings of European Conference on Object-Oriented Programming (ECOOP 2003)*, pp.2-28, 2003.

[14] Shonle, M., Lieberherr, K., and Shah, A.: XAspects: An Extensible System for Domain-specific Aspect Languages, *Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2003), Domain-Driven Development papers*, pp.28-37, 2003.

[15] Smith, B. C.: Reflection and Semantics in Lisp, In *Proceedings of Annual Symposium on Principles of Programming Languages (POPL'84)*, pp.23-35, 1984.

[16] Tanter, E. and Noye, J.: A Versatile Kernel for Multi-Language AOP, In *Proceedings of Generative Programming and Component Engineering (GPCE 2005)*, pp.173-188, 2005.

[17] Tarr, P., Ossher, H., Harrison, W. and Sutton, S.M., Jr.: N Degrees of Separation: Multi-dimensional Separation of Concerns, In *Proceedings of International Conference on Software Engineering (ICSE'99)*, pp.107-119, 1999.

[18] Ubayashi, N., Moriyama, G., Masuhara, H., and Tamai, T.: A Parameterized Interpreter for Modeling Different AOP Mechanisms, In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005)*, pp.194-203 (2005).

[19] Ubayashi, N., Tamai, T., Sano, S., Maeno, Y., and Murakami, S.: Model Compiler Construction Based on Aspect-Oriented Mechanisms, In *Proceedings of the 4th ACM SIGPLAN International Conference on Generative Programming and Component Engineering (GPCE 2005)*, pp.109-124 (2005).

[20] Volder, K., Brichau, J., Mens, K., and D'Hondt, T.: Logic Meta Programming, a Framework for Domain-Specific Aspect Languages, http://www.cs.ubc.ca/ kdvolder/, 2001.

Modeling Complex Software Systems Using an Aspect Extension of Object-Z *

Huiqun Yu[1], Dongmei Liu[1], Zhiqing Shao[1], Xudong He[2]
[1] Department of Computer Science, East China University of Sci & Tech, Shanghai 200237, China
{yhq|dmliu|zshao}@ecust.edu.cn
[2] School of Computing & Info Sciences, Florida International University, Miami, FL 33199, USA
hex@cis.fiu.edu

Abstract

Aspect-oriented programming aims to enhancing concern modulization and integration of complex software systems, which arouses great interest for researchers and practitioners. However, less attention has been paid to modeling and quality assurance at the early phase of aspect-oriented software development. This paper proposes a modeling method based on an aspect extension of Object-Z. The aspect extension provides means for observing behaviors of class schemas and depicting their interrelationships. System functionality and other crosscutting concerns can be separately modeled, and a weaving mechanism systematically composes these models into a complete system model. Our aspect-oriented modeling method enjoys good traceability for property analysis, and reusability for system specification.

Keywords*: Aspect orientation, Object-Z, formal method, modeling, analysis*

1. Introduction

Aspect-oriented programming (AOP) [7] has been proposed as a technique for improving separation of concerns in software. Many aspect-oriented techniques have been presented in literature, such as adaptive programming [9], AspectJ [6], Composition Filters [1], multi-dimensional separation of concerns [10], and AspectWerkz [2]. The central idea of AOP is that complex computer systems are better programmed by separately specifying the various concerns of a system and some description of their relationships, and then relying on mechanisms in the underlying AOP environment to weave or compose them together into a coherent program.

*This work was partially supported by the NSF of China under grants No. 60473055 and 60373075, Shanghai Pujiang Program under grant No. 05PJ14030, and the NSF of the USA under grant HRD-0317692.

While most investigations have focused on language constructs for aspect description and aspect weaving at code level, recent work has demonstrated the need to deploy AOP idea early in the software life cycle [13], because quality assurance in the early stage of software development can result in better design, and shorter time to market. Although some works have been done to identify high level aspects [5, 12, 8, 4], they fell short in precise notations for expressing different concerns and for manipulating these concerns in a systematic fashion.

This paper proposes a formal aspect-oriented modeling method based on Object-Z [14]. Object-Z is an extension to Z [15] in order to facilitate modeling in an object-oriented style through the addition of classes. The notion of aspect introduced to Object-Z is inspired by AspectJ. The basic idea is to provide means for observing behaviors of Object-Z schemas and depicting their interrelationships, and to provide ways for weaving interrelated schemas. Correctness of aspect weaving can be analyzed using the formal semantics of Object-Z.

The main contributions of this paper include:

1. proposing the formal aspect-oriented modeling method based on Object-Z, and
2. applying the aspect-oriented method to modeling and analyzing complex software systems.

The rest of the paper is organized as follows: Section 2 proposes an aspect-oriented modeling method based on Object-Z. Section 3 introduces a case study of applying the aspect-oriented approach to modeling a bounded buffer system. Section 4 presents requirement analysis and model reusability. Section 5 is the conclusion.

2. An Aspect Extension to Object-Z

2.1. Object-Z

Object-Z [14] is an extension of Z in which the existing syntax and semantics of Z are retained and new con-

structs are introduced to facilitate specification in an object-oriented style. The major extension in Object-Z is the *class schema* which captures the object-oriented notion of a class by encapsulating a single state schema with all the operation schemas which may affect its variables. A class schema defines a type whose instances are objects. The basic structure of a class schema is as follows.

ClassName[*generic parameters*]
type definitions
constant definitions
state schema
initial state schema
operation schemas

Type and constant definitions are as in Z. The state schema is nameless and comprises declarations of state variables and a state predicate. The state predicate forms the class invariant. It is implicitly included in every operation and the initial state schema. The initial state schema defines the possible initial states. The initial state is not unique. Operation schemas describe the methods defined for the class. Operation schemas have a Δ-list of those individual state variables who's value may change when the operation is applied to an object of the class.

As an example, consider the following class *BoundedBuffer*.

BoundedBuffer

$MAX : \mathbb{N}$

$card : \mathbb{N}$
$buf : \text{seq } Item$

INIT

$card = 0$
$buf = \langle\,\rangle$

Put

$\Delta(buf, card)$
$inItem? : Item$

$card < MAX \land card' = card + 1$
$\exists\, b_1, b_2 \bullet (buf = b_1 \frown b_2 \land$
$\quad buf' = b_1 \frown \langle inItem? \rangle \frown b_2)$

Get

$\Delta(buf, card)$
$outItem! : Item$

$card > 0 \land card' = card - 1$
$\exists\, b_1, b_2 \bullet (buf = b_1 \frown \langle outItem! \rangle \frown b_2 \land$
$\quad buf' = b_1 \frown b_2)$

The class has a constant *MAX*, and two state variables *card* and *buf*. Suppose the type *Item* has already been defined. Initially, *card* has the value 0 while *buf* $\langle\,\rangle$. Two of the class operators *Put* and *Get* are defined by operation schemas. The Δ list in *Put* indicates that the operator *Put* may change the values of both *card* and *buf*. The predicates in *Put* requires that the buffer is not full. On application of *Put*, the value of *card* will increase by one and the *inItem*? is inserted somewhere in the buffer[1]. The operator *Get* gets an *outItem*! from the buffer.

Conjunction, disjunction and the other Z schema operations can be used to form new operations. $[]$ indicates non-deterministic choice of one operation from the constituent operations with satisfied pre-conditions. The parallel operator, $\|$ achieves inter-object communication. The operation conjoins operation schemas but also identifies and hides input and outputs. For details of Object-Z, the reader is referred to [14].

2.2. Adding Aspect to Object-Z

The aspect orientation notion extended to Object-Z is inspired by the idea of AspectJ [6]. We introduce the concepts of join point, pointcut, advice and aspect into Object-Z. Join points are operator identifiers in Object-Z specification. A *pointcut* is a means of referring to a collection of join points and the common property at those join points; *Advice* is a mechanism used to declare that certain behavior should be performed at each join point in a pointcut. The general form of an aspect specification is as follows.

AspectName

Ω*ClassRef*
Pointcut PointcutName : $\mathbb{P}(Ident)$
...

INIT
...

PointcutName
...
...
[*insert* | *replace*]*Predicate*

...

An aspect specification has the same structure as that of class specification. However, the former includes aspect notations in state schema and operation schemas. Ω*ClassRef*

[1]For simplicity, we assume that each item in the bounded buffer be distinct.

indicates that *AspectName* crosscuts *ClassRef*. Advices are categorized into two types: *insert* type, and *replace* type, which are similar as in AspectZ [17]. The "insert" advice is used to reinforce some constraints to the specification, while the "replace" advice is used to modify some function in the base model.

2.3. The Aspect-Oriented Modeling Method

The aspect-oriented modeling framework is illustrated in Figure 1. The basic idea is to provide notations for separately modeling system functionality modules (the base model in Object-Z notation) and other crosscutting concerns (the aspect model in Object-Z extension), and to provide mechanism for systematically composing (or weaving) these models into a complete system model. The aspect-oriented approach to system modeling in Object-Z consists of the following steps:

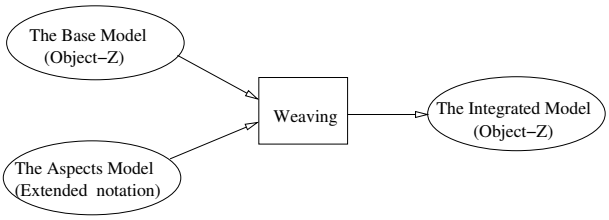


Figure 1. An aspect-oriented modeling framework

(1) Separating aspects from the basic functionality components of the system, and identifying the join points that the functionality components and aspects interact.

(2) Specifying the base model and the aspects in the extended Object-Z;

(3) For each join point in the base model, composing the behavior according to the advice defined in the aspect models.

- If it is an "*insert*" aspect, the predicate part of the advice will be inserted to the operation schema.
- If it is a "*replace*" aspect, the corresponding predicate in each join point *Ident* in base model will be replaced by the advice predicate whenever the *Ident* appears in the pointcut declaration.

In general, more than one piece of advice may apply at a join point. The different advice can come from different aspects or even the same aspect. The relative order in which such advice executes is domain dependent. The ordering is based on the fact that aspects are the primary units of crosscutting functionality. So *advice ordering* (or specificity), is resolved with the relative precedence of the aspects in which the advice is defined. We will illustrate this in the next section.

3. Case Study: The Bounded Buffer System

3.1. System Description

A bounded buffer has a limited capacity and provides services to producers and consumers. There are following requirements for the bounded buffer.

R1: The buffer shall have limited capacity.

R2: Buffer shall block get requests when it is empty.

R3: Buffer shall block put requests when it is full.

R4: The system shall allow only one producer or one consumer to access the buffer at a certain time.

R5: The products shall be consumed in the same order as they were produced.

The core functionality of the bounded buffer system includes buffer access, products producing and consuming. The first three requirements constrain capacity of the buffer. In the aspect-oriented framework, the above functionality and requirements are modeled as the base components. The exclusiveness requirement R4 and scheduling requirement R5 crosscut the core functionality of the bounded buffer, and therefore they are modeled as aspects.

3.2. The Basic Components

In addition to the previous *BoundedBuffer*, the basic components also include class schemas *Producer*, *Consumer*, and *BoundedBufferSystem* as follows.

Producer

PutItem

$inItem! : Item$

PutDone

$putDone! : Bool$

$Produce \mathrel{\widehat=} PutItem \fatsemi PutDone$

The operator *Produce* is defined as sequential composition of *PutItem* and *PutDone*.

Consumer

GetItem

$outItem? : Item$

GetDone

$getDone! : Bool$

$Consume \mathrel{\widehat=} GetItem \fatsemi GetDone$

The operator *Consume* is defined as sequential composition of *GetItem* and *GetDone*.

BoundedBufferSystem

$buffer : BoundedBuffer$

$pro : Producer$

$con : Consumer$

INIT

$buffer.INIT$

$Produce \mathrel{\widehat=} pro.Produce$
$Consume \mathrel{\widehat=} con.Consume$
$Put \mathrel{\widehat=} buffer.Put$
$Get \mathrel{\widehat=} buffer.Get$
$Contact \mathrel{\widehat=} (Produce \parallel Put) [] (Consume \parallel Get)$

The bounded buffer system consists of a bounded buffer *buffer*, a producer *pro* and a consumer *con*. Initially, the bounded buffer is in its initial state. Each of the operations in *BoundedBufferSystem* is a promotion of an operation defined in *buffer*, *pro* or *con*. The final operation, *Contact*, models behavioral composition of the above operations.

3.3. The Aspect Models

Aspect 1 The following is an aspect specification of exclusiveness of producing and consuming activities.

ExclusivenessAspect

$\Omega BoundedBufferSystem$

$flag : Semaphore$
$Pointcut\ PC1 : \{PutItem, GetItem\}$
$Pointcut\ PC2 : \{PutDone, GetDone\}$

INIT

$flag = 0$

*PC*1

$insert\ flag = 0 \land flag' = 1$

*PC*2

$insert\ flag = 1 \land flag' = 0$

The variable *flag* is a semaphore, which means checking the value of *flag* and changing it is done as an indivisible atomic action. The type of advices for both $PC1$ and $PC2$ is "insert". The advice for $PC1$ is to hold the semaphore *flag*, while the advice for $PC2$ is to release it.

Aspect 2 One way to ensure the scheduling requirement to the bounded buffer is to introduce queued buffer, which is specified by aspect *SchedulingAspect*.

SchedulingAspect

$\Omega BoundedBufferSystem$

$Pointcut\ PC3 : \{Put\}$
$Pointcut\ PC4 : \{Get\}$

*PC*3

$replace\ buf' = buf \frown \langle inItem? \rangle$

*PC*4

$replace\ buf = \langle outItem! \rangle \frown buf'$

In aspect specification *SchedulingAspect*, the type of advices is "replace". The "replace" advice is used to modify some function or operation in the base model, which is similar to that in AspectZ [17]. Note that we use implicit definition of functions. Therefore, when a "replace" advice is applied, the predicates with the concerned variables in the base model will be removed, and the advice predicate will be inserted.

3.4. Aspect Weaving

Figure 2 shows the relationship among components and aspects.

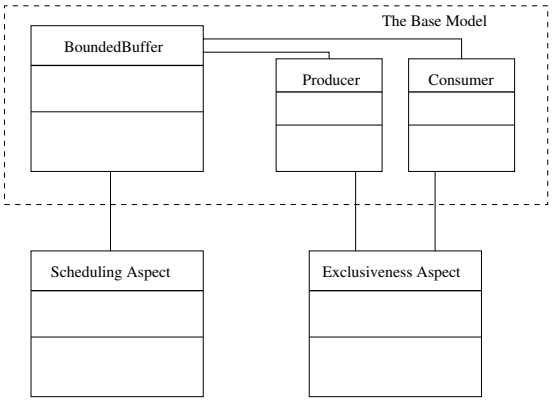


Figure 2. Relating components with aspects

We must be careful about the advice ordering when setting out to aspect weaving, for there are two aspects crosscut the base model. Advice ordering is domain specific.

For the bounded buffer system, the scheduling aspect has the precedence over the exclusiveness aspect. When the scheduling aspect is applied, the constraints of *Put* and *Get* will be modified. When exclusiveness aspect is applied, it will add to the base model new constraints as well as the variable *flag*.

4. Correctness and Reusability

4.1. Requirement Analysis

One of the major advantages of using a formal language like Object-Z is that it is able to reason about the specifications written in it. The requirement specification language is a linear temporal logic (LTL) [3, 11]. A *temporal formula* in LTL is constructed out of state formulas to which we apply the boolean connectives and temporal operators. Common temporal operators include $\Box$ (always), $\Diamond$ (eventually), and $\mathcal{U}$ (unless).

A *model* for a temporal formula p is an infinite sequence of states $\sigma : s_0, s_1, \ldots$, where each state s_j provides an interpretation for the variables mentioned in p. The semantics of a temporal formula p in a given model σ and position j is denoted by $(\sigma, j) \models p$. We define:

- For a state formula p, $(\sigma, j) \models p \;\Leftrightarrow\; s_j \models p$
- $(\sigma, j) \models \neg p \;\Leftrightarrow\; (\sigma, j) \not\models p$
- $(\sigma, j) \models p \vee q \;\Leftrightarrow\; (\sigma, j) \models p$ or $(\sigma, j) \models q$
- $(\sigma, j) \models \Box p \;\Leftrightarrow\; (\sigma, i) \models p$ for all $i \geq j$
- $(\sigma, j) \models \Diamond p \;\Leftrightarrow\; (\sigma, i) \models p$ for some $i \geq j$
- $(\sigma, j) \models p \mathcal{U} q \;\Leftrightarrow\; (\sigma, k) \not\models q$ for all $k > j$, or $(\sigma, k) \models q$ for some $k > j$ and $(\sigma, i) \models p$ for all $j \leq i < k$

The reader is referred to [11] for detailed presentation of temporal logic.

Formal semantics of Object-Z component like OZ structure in [16] will be used as model to interpret temporal logic formulas. A model $\sigma : s_0, s_1, \ldots$ satisfies a temporal formula if $(\sigma, 0) \models p$. A temporal formula p that is *valid* over an Object-Z component P specifies a property of P. Using the semantics of Object-Z, we can analyze the properties of the bounded buffer system model.

Property 1 (Synchronization) *Buffer shall block get requests when it is empty, and block put requests when it is full.*

$\Box((\mathit{PutItem} \rightarrow \mathit{card} < \mathit{MAX}) \wedge (\mathit{GetItem} \rightarrow \mathit{card} > 0))$

Proof: When the operation *PutItem* is invoked, it produces an *inItem*! in parallel with the operation *Put*. According to the semantics of parallel composition, *inItem*! is equated to *inItem*? of *Put*. The pre-condition of *Put* is exactly $\mathit{card} < \mathit{MAX}$. It is therefore true that $\mathit{PutItem} \rightarrow \mathit{card} < \mathit{MAX}$. Similarly, we can see $\mathit{GetItem} \rightarrow \mathit{card} > 0$. $|\Box$

Property 2 (Exclusiveness) *The system shall allow only one producer or one consumer to access the buffer at a certain time.*

$\Box(\neg \mathit{PutItem} \vee \neg \mathit{GetItem})$

Proof: According to the exclusiveness aspect model, pre-condition for invoking the operation *PutItem* is $\mathit{flag} = 0$. When *PutItem* is invoked, it sets the value of *flag* to 1. Checking the value of *flag*, changing it is done as an indivisible atomic action. Note that the pre-condition for invoking the operation *GetItem* is $\mathit{flag} = 0$. Hence, $\mathit{PutItem} \rightarrow \neg \mathit{GetItem}$, which logically equals to $\neg \mathit{PutItem} \vee \neg \mathit{GetItem}$. $|\Box$

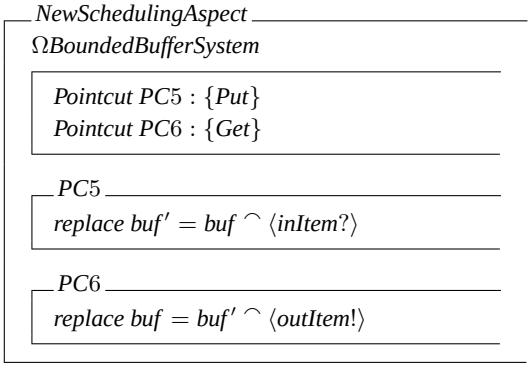
Property 3 (Scheduling) *The products shall be consumed in the same order as they were produced.*

$\forall x \bullet \Box(x \in (\mathit{buf} \frown \langle \mathit{new} \rangle) \wedge x \neq \mathit{new} \rightarrow ((\mathit{new} \neq \mathit{outItem}!)\mathcal{U}(x = \mathit{outItem}!)))$

Proof: According to the scheduling aspect model, each time the operation *GetItem* is invoked, *outItem*! chooses the head of *buf*, while each time the operation *PutItem* is invoked, *inItem*? is attached to the rear of *buf*. In other words, the buffer is a first-in-first-out queue. $x \in (\mathit{buf} \frown \langle \mathit{new} \rangle) \wedge x \neq \mathit{new}$ indicates that current buffer is of the form $b_1 \frown \langle x \rangle \frown b_2 \frown \langle \mathit{new} \rangle$. So the product x is consumed before *new* is done. $|\Box$

4.2. Reusability

Another benefit of the modeling method is that it will facilitate the reusability of models. Suppose an epicurean prefers to consuming the freshest product each time. The scheduling strategy can be specified by the following aspect model.



If the aspect *SchedulingAspect* in Figure 2 is replaced by *NewSchedulingAspect*, a new bounded buffer system model that satisfies the epicurean scheduling requirement is constructed. All the other components and the exclusiveness aspect remain untouched and are reusable.

5. Concluding Remarks

This paper has proposed a formal aspect-oriented modeling method based on Object-Z. Concepts of join point, pointcut, advice and aspect are introduced to Object-Z. The modeling steps include separately specifying system functionality and other crosscutting concerns, and weaving these specifications into a complete system model according to advice predicates. The two advice types in Object-Z, i.e. *insert* and *replace*, are counterparts of *before*, *after* and *around* advices in AspectJ [6]. Formal semantics of Object-Z schemas makes it possible to rigorously analyze system properties. Aspect-oriented models enjoy good traceability for property analysis, and reusability of system specification.

Our initial experience shows formal methods are promising in promoting aspect-oriented software development for better understandability, reliability, and reusability. Currently, we are investigating more language features of aspect orientation in Object-Z. Future research interests include more complex case studies, tool support for aspect-oriented modeling, design and verification.

Acknowledgments

The authors thank one anonymous reviewer for the insightful comments, which help improve the presentation of this paper.

References

[1] L. Bergmans and M. Aksits. Composing crosscutting concerns using composition filters. *Communications of the ACM*, 44(10):51 – 57, 2001.

[2] J. Bonér. What are the key issues for commercial AOP use: how does AspectWerkz address them? In *AOSD '04: Proceedings of the 3rd international conference on Aspect-oriented software development*, pages 5–6, New York, NY, USA, 2004. ACM Press.

[3] E. A. Emerson. Temporal logic and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B*, pages 995–1072. Elsevier Science Publishers, 1990.

[4] A. Garcia, C. Sant'Anna, E. Figueiredo, U. Kulesza, C. Lucena, and A. von Staa. Modularizing design patterns with aspects: a quantitative study. In *AOSD '05: Proceedings of the 4th International conference on Aspect-oriented software development*, pages 3–14, New York, NY, USA, 2005. ACM Press.

[5] G. George, R. France, and I. Ray. Design high integrity systems using aspects. In *Proceedings of the 5th IFIP TC-11 WG 11.5 Working Conference on Integrity and Internal Control in Information Systems*, pages 37–57, 2002.

[6] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold. An overview of AspectJ. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP), LNCS 2072*, pages 327–353. Springer-Verlag, 2001.

[7] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP), LNCS 1241*, pages 220–242. Springer-Verlag, 1997.

[8] D. K. Kim, I. Ray, R. B. France, and N. Li. Modeling role-based access control using parameterized UML models. In M. Wermelinger and T. Margaria, editors, *Proceedings of 7th International Conference on Fundamental Approaches to Software Engineering (FASE 2004), LNCS 2984*, pages 180–193. Springer, 2004.

[9] K. Lieberherr, D. Orleans, and J. Ovlinger. Aspect-oriented programming with adaptive methods. *Communications of the ACM*, 44(10):39 – 41, 2001.

[10] P. Maes. Concepts and experiments in computational reflection. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 147–155. ACM, 1987.

[11] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, 1992.

[12] B. Nuseibeh. Crosscutting requirements. In *AOSD '04: Proceedings of the 3rd international conference on Aspect-oriented software development*, pages 3–4, New York, NY, USA, 2004. ACM Press.

[13] A. Rashid, A. Moreira, and J. Araújo. Modularisation and composition of aspectual requirements. In *Proceedings of the 2nd International Conference on Aspect-Oriented Software Development (ASOD 2003)*, pages 11–20. ACM, 2003.

[14] G. Smith. *An Object-Oriented Approach to Formal Specification*. PhD thesis, University of Queensland, St. Lucia 4072, Australia, 1992.

[15] J. Spivey. *The Z Notation: A Refernce Manual (2rd Edition)*. Prentice Hall, UK, 1992.

[16] K. Winter and G. Smith. Compositional verification for Object-Z. http://citeseer.ist.psu.edu/winter03compositional.html, 2003.

[17] H. Yu, D. Liu, L. Yang, and X. He. Formal aspect-oriented modeling and analysis by AspectZ. *Proceedings of 17th International Conference on Software Engineering and Knowledge Engineering, Taipei*, pages 175–180, 2005.

Customizing Aspect-Oriented Variabilities using Generative Techniques

Uirá Kulesza[1] Carlos Lucena[1] Paulo Alencar[2] Alessandro Garcia[3]

[1]*Software Engineering Laboratory – Computer Science Department – PUC-Rio – Brazil*
{uira, lucena}@inf.puc-rio.br

[2]*Computer Systems Group – Computer Science Department – University of Waterloo – Canada*
palencar@csg.uwaterloo.ca

[3]*Computing Department – InfoLab 21 – Lancaster University – UK*
garciaa@comp.lancs.ac.uk

Abstract

With the emergence of aspect-oriented (AO) techniques, crosscutting concerns can be now explicitly modularized and exposed as additional variabilities in program families. Hence, the development of highly customizable software family architectures requires the explicit handling of crosscutting variabilities through domain engineering and application engineering steps. In this context, this paper presents a generative model that addresses the implementation and instantiation of variabilities encountered in AO software family architectures. The use of our model allows for an early specification and preparation of AO variabilities, which in turn can be explicitly customized by means of domain engineering activities. All the variabilities of the architecture are modeled using feature models. In application engineering, developers can request an instance of the AO architecture in a process of two stages: (i) the definition of a feature model instance which specifies the resolution of variabilities for the AO family architecture; and (ii) the definition of a set of crosscutting relationships between features.

1. Introduction

Aspect-oriented (AO) techniques have been proposed as an approach which aims to separate and modularize crosscutting concerns [7, 8]. Crosscutting concerns are concerns that often crosscut several modules in a software system. It encourages modular descriptions of complex software by providing support for cleanly separating the basic system functionality from its crosscutting concerns. Hence, AO techniques can be used now to exploit variabilities relative to crosscutting concerns, thereby enhancing the reusability and customizability of software family architectures.

Aspect-oriented programming (AOP) [8] proposes the *aspect* abstraction and new composition mechanisms which allow for the implementation of crosscutting variabilities. Aspects are modular units that extend the functionalities of classes in well-defined execution points, the so-called *join points*. AspectJ [1] is the most popular programming language that enriches the Java language with AO extensions. It provides constructions to specify more reusable and variable aspects, such as, abstract aspects and abstract *pointcuts*.

Recent work [2, 9-13] has focused on the use of AO techniques to enable the implementation of flexible and customizable software family architectures. In these research works, aspects are exclusively used to modularize crosscutting variable (optional or alternative) features encountered in the programming of frameworks or software product lines. However, there is a lack of support for software family architects and application developers to respectively manage and instantiate AO variabilities in different development stages. It is important to define models which allows to handle not only orthogonal variabilities enabled by classical OO mechanisms (such as, OO framework hotspots), but also the new ways of variabilities supported by AOP.

In this context, this paper presents a generative model [5] which provides explicit means to instantiate aspect-oriented family architectures. Using our model, AO variabilities are prepared to be customized through domain engineering activities. All the variabilities of the architecture are modeled using feature models. In application engineering, developers can request an instance of the AO architecture in a process of two stages: (i) the definition of a feature model instance which specifies the resolution of variabilities for the AO family architecture; and (ii) the definition of a set of crosscutting relationships between features. These crosscutting relationships are used to customize

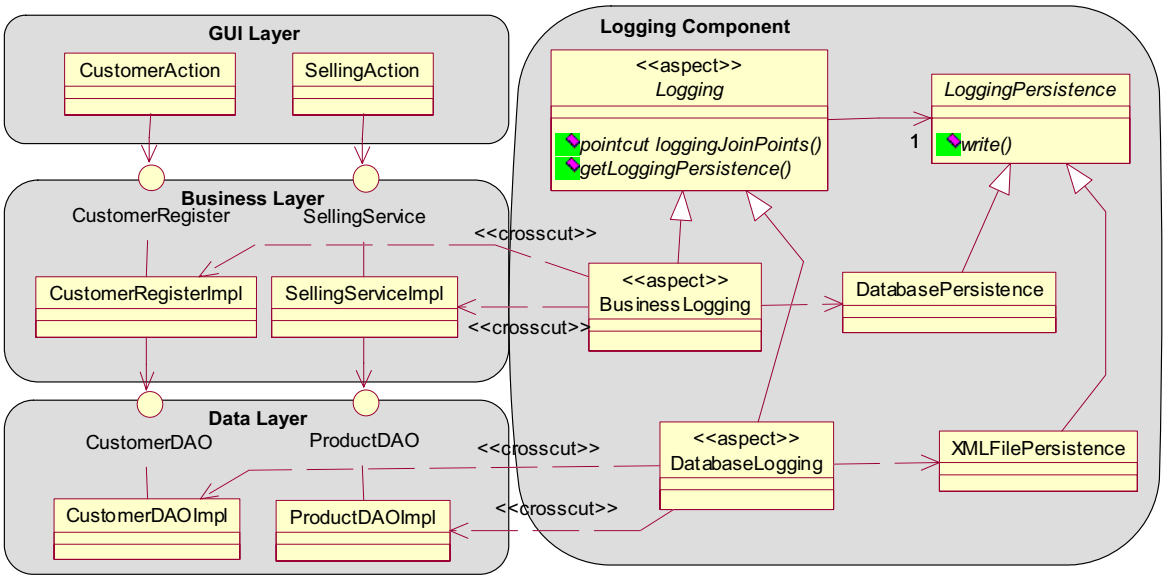


Figure 1. Logging Aspects

abstract pointcuts which define how aspects can affect system classes.

The remainder of this paper is organized as follows. Section 2 presents an overview of the variability mechanisms supported by AOP. Section 3 describes our generative model. Section 4 exemplifies our model application to the customization of logging aspects. It describes both domain implementation and application engineering activities. Section 5 presents our conclusion and points to directions for future work.

2. Variabilities in AOP

AO programming languages offer constructions to specify a set of join points. The join point model defined in such languages allows the aspect implementation to extend the class functionalities in specific points. Each aspect specifies a set of pointcuts and advices to implement these extensions. Pointcuts have a name and are collections of join points. Advices are a special method-like construction of aspects which are used to attach new crosscutting behaviors along the aspect pointcuts. These mechanisms can address the implementation of optional and alternative crosscutting features encountered in the implementation of software architectures.

As discussed in Section 1, AO languages, such as AspectJ [1], also support the definition of abstract aspects which can contain both abstract pointcuts and methods. These constructions enable to postpone the implementations of pointcuts and methods to concrete subaspects. Each subaspect can customize these elements considering a particular implementation of interest. Thus, more reusable aspects can be specified using these mechanisms.

Figure 1 presents an example of a `Logging` abstract aspect. It defines the `loggingJoinPoints()` abstract pointcut and the `getLoggingPersistence()` abstract method. The former is used by the subaspects to specify the join points in an application which will be logged. The latter allows subaspects to define a specific persistence mechanism to accomplish the logging. Figure 1 also shows two subaspects which implement in different ways the logging for a web application. The `BusinessLogging` aspect defines that the logging of the business services will be realized in the database. The `DatabaseLogging` aspect specifies the logging of the database accesses in a XML file. Figure 2 shows the AspectJ source code of the `Logging` abstract aspect and the `BusinessLogging` subaspect.

```
public abstract aspect Logging  {
    private LoggingPersistence log =
                getPersistenceLogging();

    abstract pointcut loggingJoinPoints();
    after(): loggingJoinPoints() {
       this.log.write(thisJoinPoint);
    }
    public abstract LoggingPersistence
                        getLoggingPersistence();
}
public aspect BusinessLogging extends Logging {
    pointcut loggingJoinPoints():
       execution(CustomerService+.*) ||
       execution(SellingService+.*);

    public LoggingPersistence
                    getLoggingPersistence(){
      return new DatabasePersistence();
    }
}
```

Figure 2. Source Code of Logging Aspects

The subaspects presented in Figures 1 and 2 were manually codified to customize the Logging feature for a specific web system. Different crosscutting features could also be implemented as abstract aspects and be reused in the implementation of subaspects. An automated mechanism could support the generation of these subaspects. It requires not only configuring existing functional variabilities (such as, the alternative persistence in the logging example), but also generating specific pointcuts in the subaspects. In this paper, we propose an aspect-oriented generative model which aims to generate these subaspects, including the customization of their pointcuts.

3. Approach Overview

Our approach is centered on the concepts of generative programming. Generative Programming (GP) [5] addresses the study and definition of methods and tools that enable the automatic generation of software from a given high-level specification language. It has been proposed as an approach based on domain engineering [5]. GP promotes the separation of problem and solution spaces, giving flexibility to evolve both independently. To provide this separation, GP proposes the concept of a generative domain model.

A generative domain model is composed of three basic elements: (i) problem space – which represents the concepts and features existent in a specific domain; (ii) solution space – which consists of the software architecture and components used to build members of a software family; and (iii) configuration knowledge – which defines how specific feature combinations in the problem space are mapped to a set of software components in the solution space. Two new activities need to be introduced to domain engineering methods in order to address the goals of GP:

- development of proper means to specify specific members of the software family. Domain-specific languages (DSLs) must be developed to deal with this requirement;
- modeling of the configuration knowledge in detail in order to automate it by means of a code generator.

We have defined an AO generative model following the model presented by Czarnecki and Eisenecker [5]. However, we propose the extension of that generative model to support the instantiation and customization of AO architectures. It allows configuring and generating specific crosscutting and non-crosscutting variabilities. Our generative model is composed by the following elements:

(I) *a feature model* – this model works as a configuration domain-specific language (DSL) responsible to specify and collect the features to be instantiated in the software family architecture. It is used to collect information to configure both the crosscutting and non-crosscutting variabilities. A set of crosscutting relationships between features is used to help the customization of aspects pointcuts.

(II) *an AO architecture* – it defines the main components of a software family architecture. This architecture defines a set of variabilities which need be customized to define a complete application. Crosscutting variabilities are implemented as aspects in this architecture. Each component of the architecture is specified as a set of classes, aspects and templates. The latter ones define elements that will be customized during the instantiation of the architecture. We also provide guidelines to implement these AO architectures by means of a base OO framework and a set of aspects which define optional and alternative crosscutting features existing in the OO framework. More details on these guidelines are provided in [9];

(III) *a configuration model* – it specifies the mapping between the features existing in the crosscutting feature model and the components (or their respective sub-elements, such as, class, aspect or templates) of the AO architecture. The configuration model is used to support the decision of which components must be instantiated and what customizations must be realized in those components considering a specific application.

There are several activities involved in the process of development of the elements of our generative approach. These activities are organized under the perspectives of domain implementation and application engineering. Next section details them in the context of the Logging example presented in Section 2.

4. Customizing AOP Variabilities: a Working Example

In this section, we illustrate our generative model by showing the customization of an AO architecture. We explore the customization of the Logging subaspects presented in Section 2. This is an illustrative example which allows to show how aspects can be customized using our approach. The same strategies, that we are going to show in the Logging example, can be used to customize other and different reusable aspects in more complex architectures, such as frameworks or product lines [2, 9, 11].

4.1. Domain Implementation

We first present the domain implementation activities which prepare an AO architecture to be automatically instantiated.

Activity 1*: AO Architecture Implementation.*

The first activity of the domain implementation is to implement an AO family architecture that addresses a

set of variabilities in a specific domain. The Logging aspect example (presented in Section 2) defines two variabilities: (i) *the logging pointcuts* – which represent the execution join points in the web system that will be logged; and (ii) *the logging persistence mechanism* – which defines alternative persistence ways to store the logging information. These variabilities are addressed by the aspect/class hierarchy presented in Figure 1.

The Logging subaspects are the only elements which need to be customized during the instantiation of a web system. In our approach, every element (class, aspect, interface or configuration file) which need to be customized during application engineering is implemented as a code template. Code templates allow us to represent structure and behavior of specific classes and aspects that we want to generate. Java Emitter Templates (JET), a generic template engine of the Eclipse Modeling Framework (EMF) [4], has been used to specify our templates. Thus, to specify the general structure of our Logging subaspects, we defined the `ConcreteLogging` JET template. It is used to generate specific logging subaspects. The `loggingJoinPoints` pointcut and `getLoggingPersistence()` method are customized based on information collected by the feature model. Our templates are processed by our code generator during architecture customization.

Activity 2*: Representation of the Variabilities in the Feature Model.*

After implementing all the variabilities of an AO architecture, the next activity is to represent them in a feature model. We use the feature model proposed in [6], which allows modeling mandatory, optional and alternative features, as well as their respective cardinality. The feature modeling plugin (fmp) [3] supports the modeling of feature models in Eclipse platform.

In order to support the customization of aspects in our approach, we have extended the feature model proposed in [6]. We can assign the *crosscutting* or *joinpoint* property to specific features. A *crosscutting feature* is used to represent aspects which can extend the behavior of other system features. A *joinpoint feature* is used to specify specific execution points in the system which can be extended by aspects. Crosscutting relationships between these elements can be defined in the application engineering (Section 4.2) in order to customize aspects to affect specific parts of the system.

Figure 3 shows the feature model of the Logging example using the fmp plugin. We first represent the business and data services which are provided by the web system. These features are all represented as mandatory (symbol ●), because they do not represent variabilities in the web system. They were only modeled because they represent possible features which the application engineer can desire to log their

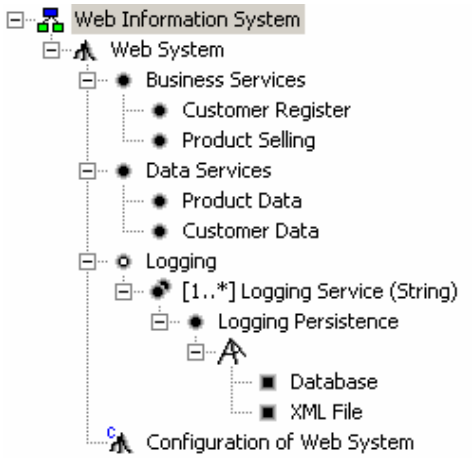


Figure 3. Logging Feature Model

execution. Since they are all candidates to be extended by crosscutting features, we call them *joinpoint features*.

Figure 3 also presents the logging variabilities. The logging feature is optional (symbol ○). It is composed by a set of logging services. Each logging service represents a possible logging subaspect to be created. The definition of a logging service feature involves the choice of one between two alternative persistence mechanisms: Database or XML File features. Each logging service needs also to be configured to extend specific services of the web system. Because of that, they are characterized as *crosscutting features*.

Activity 3*: Specification of the Configuration Model.*

The last activity of the domain implementation is the configuration model specification. The configuration model represents the configuration knowledge [5] in generative programming. It is used mainly to define how a specific configuration of features is mapped to a configuration of architecture components. All the information specified in the configuration model is used by our code generator to enable the automatic customization of AO architectures during application engineering.

Our configuration model is composed by three different elements: (I) description of dependency relationships between the architecture model's components (and sub-elements) and the features specified in the feature model; (II) definition of valid crosscutting relationships between *crosscutting* and *joinpoint* features; and (III) specification of the

mapping between *joinpoint* features and specific joinpoints in classes of the AO architecture. All these elements are being implemented as wizards of Eclipse [14] plugins.

The first element of our configuration model are the dependency relationships between architecture components (and sub-elements, such as, classes, aspects and templates) and features. They are used to define the mapping between the feature model and the architecture components. They allow to specify which components must be instantiated when specific features are selected. The following guidelines are used when defining the dependency relationships: (i) if a component (or sub-element) must be instantiated to every product of the product line, then no dependency relationships needs to be specified; (ii) if a component (or sub-element) depends on the occurrence of a specific feature, a dependency relationship must be created between them.

The dependency relationships are used by our code generator to decide which classes and aspects will be included in a product based on feature model instances defined by application engineers. In case of templates, the dependency relationships define if they will be processed and included in the final product generated. Every template element depends on specific features which provide knowledge necessary for their instantiation.

Figure 4 shows a set of dependency relationships between the Logging component and its respective feature model. It shows that the logging component will be instantiated only if the logging feature is selected by the application engineer. When the logging feature is selected, every element inside the logging component which does not have a dependency relationship with any feature will be automatically instantiated in the architecture. This is the case of the `Logging` abstract aspect and the `LoggingPersistence` class. Figure 4 also presents that the `ConcreteLogging` aspect template has a dependency relationship with the logging service feature. It means that a new different concrete logging aspect will be created for each logging service feature specified. Finally, `DatabaseLogging` and `XMLFileLogging` classes will be instantiated only if the database and XML file features, respectively, were requested by the application engineer.

The second element defined in our configuration model is the potential relationships between crosscutting and joinpoint features. This information is used by our code generator to check if the application engineers have specified valid crosscutting relationships. It allows to restrict the set of valid crosscutting relationships. Figure 4 shows the set of valid crosscutting relationships for the Logging example. It shows that every logging service feature can extend the following features: register and selling services, and product and customer data services features.

The third and last element of our configuration model is the mapping between the joinpoint features and the concrete joinpoints in classes of the AO architecture. This information is used by our code generator to customize pointcuts during the generation of aspects. The mapping involves the identification of which parts of classes (e.g.: constructor execution and method call) correspond to specific joinpoint features. In our particular implementation, the mapping refers to specific and valid AspectJ joinpoints. Figure 4 shows the joinpoint mapping for the Logging example. The business and data services joinpoint features of the web system are mapped to specific joinpoints existing in the implementation of its components.

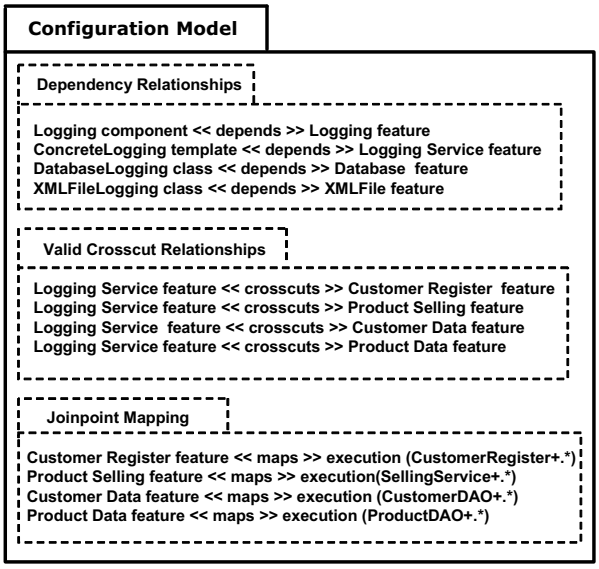


Figure 4. Logging Configuration Model

4.2. Application Engineering

In application engineering, developers request an instance of the AO architecture by specifying all desired variabilities. This request is composed of two activities: (i) choice of variabilities in a feature model instance; and (ii) choice of valid crosscutting relationships between features. This latter step is used to enable the customization of aspect pointcuts. A tool uses the information collected by these steps and the configuration model to generate an instance of the AO architecture.

Figure 5 shows a feature model instance of the Logging example. The application engineer is requesting two different logging services. The first one

is used to log information about the registration and selling business services. The other one logs information about the database access of the selling and data services. Each of them uses a different way to persist the logging information. The crosscutting relationships between features are specified separately from the feature model instance. For the Logging example, four crosscutting relationships must be defined: (i) the "Business Services" logging service crosscutting feature is related with the register and selling business services joinpoint features; and (ii) the "Data Services" logging service feature is related with the product and customer data services joinpoint features.

Using the feature model instance, the configuration model and the AO architecture of the Logging example, our code generator creates two Logging subaspects which affect the business and data services of the architecture. The complete algorithm of our code generator is described in [10].

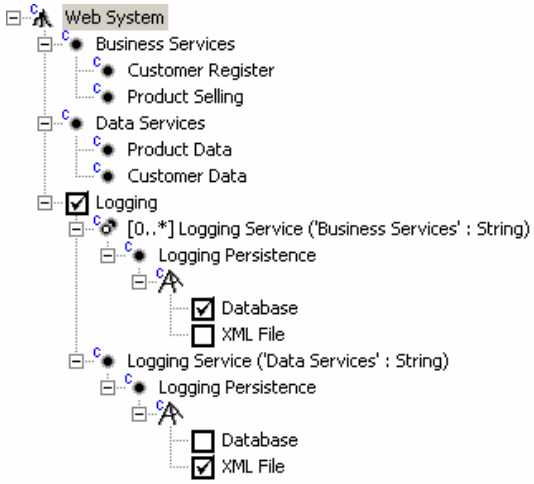


Figure 5. Logging Feature Model Instance

5. Conclusions and Future Work

This paper presented an aspect-oriented generative model which addresses the instantiation of variabilities encountered in AO architectures. We also described a set of domain implementation and application engineering activities which are adopted to prepare AO architectures to be automatically instantiated. To the best of our knowledge, the only research work which explores the instantiation of AO architectures is the Framed Aspects approach [13]. It proposes the integration between Frame and AOP technologies. The main difference between our and the Framed Aspects approach, is that they define many of the decision steps about the instantiation process in the template code of frames by means of meta-tags. In our approach, the decisions related to the architecture customization process are described separately by our configuration model. It makes easier to adapt or evolve the decisions related to the architecture customization. We also use feature model instances to gather all information necessary for the resolution of AO variabilities.

We are currently implementing a tool, as an Eclipse plug-in [14], which supports all the models presented in the paper. Also, new case studies involving software families from different domains are being realized to validate our approach. We are also exploring the instantiation of aspect libraries using our approach in these case studies. Finally, we are also refining a set of guidelines to modularize the implementation of framework variabilities using aspects [9].

Acknowledgments. The authors have been partially supported by CNPq, FAPERJ and European Network of Excellence on AOSD (AOSD-Europe).

References

[1] AspectJ Team. The AspectJ Programming Guide. http://eclipse.org/aspectj/.

[2] V. Alves, et al. "Extracting and Evolving Mobile Games Product Lines". In 9th International Software Product Line Conference (SPLC'05), September 2005.

[3] M. Antkiewicz and K. Czarnecki. FeaturePlugin: Feature modeling plug-in for Eclipse, OOPSLA'04 Eclipse Technology eXchange (ETX) Workshop, 2004.

[4] F. Budinsky, et al. Eclipse Modeling Framework. Addison-Wesley, 2004.

[5] K. Czarnecki, U. Eisenecker. Generative Programming: Methods, Tools, and Applications, Addison-Wesley, 2000.

[6] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration using feature models. In Proceedings of the Third Software Product-Line Conference, September 2004.

[7] R. Filman, T. Elrad, S. Clarke, M. Aksit. Aspect-Oriented Software Development, Addison-Wesley, Boston, 2005.

[8] G. Kiczales, et al. "Aspect-Oriented Programming". Proc. of ECOOP'97, LNCS 1241, Springer, Finland, June 1997.

[9] U. Kulesza, et al. "Improving Extensibility of Object-Oriented Frameworks with Aspect-Oriented Programming", Proceedings of ICSR´2006, Turin, Italy, June 2006.

[10] U. Kulesza, et al. "Instantiating and Customizing Product Line Architectures using Aspects and Crosscutting Feature Models". Proceedings of the Workshop on Early Aspects, OOPSLA'2005, October 2005, San Diego.

[11] U. Kulesza, et al. "A Generative Approach for Multi-Agent System Development". In "Software Engineering for MAS III". Springer, LNCS 3390, pp. 52-69, 2004.

[12] M. Mezini, K. Ostermann: "Variability management with feature-oriented programming and aspects". Proceedings of FSE'2004, SIGSOFT, pp. 127-136, 2004.

[13] N. Loughran, A. Rashid. "Framed Aspects: Supporting Variability and Configurability for AOP". Proceedings of ICSR'2004, pp. 127-140, 2004.

[14] S. Shavor, et al. The Java Developer's Guide to Eclipse. Addison-Wesley, 2003.

Collaboration Support Model of Software Development Experiment

Saeko Matsuura and Hiroki Kurihara
Shibaura Institute of Technology
370 Fukasaku Minuma-ku
Saitama-shi Saitama Japan
+81 48 687 5094
{matsuura@se,p02059@sic}.shibaura-it.ac.jp

ABSTRACT
We have been planning and conducting software development experiments based on group work as a software engineering education. The goal of software development is to create a high-quality software system that meets user requirements. On the other hand, a software development always involves a group work. So collaboration between the members decides the efficiency of the process and the quality of the final products. In order to make the software development experiment practical, it is important that the students learn both the meaning of the collaboration and the practical way of collaborating in a software development at university education. Since 2002, we have been planning and conducting group-work-based software development experiments as an approach to PBL. This paper describes how the software development process was designed and conducted in order to let the student learn the practical software engineering.

KEY WORDS
Software Development Experiment, PBL, Collaborative Work, Inspection, Software Development Process Model

1. Introduction

We have been planning and conducting software development experiments based on group work as a software engineering education. The goal of software development is to create a high-quality software system that meets user requirements. On the other hand, a software development always involves a group work. So collaboration between the members decides the efficiency of the process and the quality of the final products. In order to make the software development experiment practical, it is important that the students learn both the meaning of the collaboration and the practical way of collaborating in a software development at university education. Since 2002, we have been planning and conducting group-work-based software development experiments [1,2] as an approach to PBL [3]. The aim of this class is to master software development and project management technologies based on object-oriented development. With a group of 5 to 10 students, a moderately complicated software system can be developed over half a semester. The experiment comprises the following phases: requirement analysis, system analysis, system design, and implementation/test.

This paper describes how the software development process was designed and conducted in order to let the student learn the practical software engineering. The remainder of the paper is organized as follows. Section 2 presents the problems of how to teach the students quality improvement means of software products through their collaborative work. Section 3 provides an outline of the experiment. Section 4 describes our design of a software development process. Section 5 shows our collaboration support system. Section 6 discusses the trial results.

2. Problems of Learning a Software Quality Improvement Means

To a person who does not have much experience of software development, it is difficult to judge the validity of artifacts correctly. Once the students learn a modeling language such as UML, they get to be able to define several artifacts using diagrams. However, their artifacts resulted in having little completeness and the consistency between diagrams was lacked, so that the regression of work has occurred frequently. It is a main reason why the delay of process in a software development was caused. Inspection [4,5,6] is a widely acknowledged effective quality improvement means in software development by detecting defects involved in software artifacts and removing them through cooperative work. In a software development experiment at university education, the students need to experience a task of inspection and should become to be able to judge the validity of artifacts. However, the students who do not have much experience of software development project do not know how to inspect the artifacts at all. In this experiment, we provide guidelines of how to inspect artifacts through cooperative work so that they become feeling that an importance of inspection is real. Moreover, we provide an appropriate support tool for their collaborative work in the inspection.

3. Software Development Experiment

The various aspects of the experiments are as follows: prerequisite knowledge of the students, problems, group composition and working plan.

3.1 Prerequisite Knowledge

The experiments were conducted in the second semester for third-year students within the department. The number of participating students in 2002, 2003, 2004 and 2005 was 121, 66, 84, and 69, respectively. All the participating students had earned 2 credits each for the lectures and programming in C and 1 credit each for classroom work and programming in Java. Further, a majority of the students had earned 1 credit for lectures related to object-oriented development and UML (Unified Modeling Language) [7]; thus, they gathered some experience in conducting requirement analysis, system analysis, and system design with regard to small-scale exercises. However, they lacked experience with regard to work of implementation based on design documents, work of test planning, and practical testing.

3.2 Topics

In 2002 and 2003, we focused on a "sales management system" (topic A) and a "vending machine control system" (topic B), which were proposed as common topics for modeling at the ISPJ Object-Oriented Symposium. In 2004, the former topic was replaced with "conference room reservation system for Shibaura Institute of Technology." In 2005, a "delivery system" (topic C) by LEGO MINDSTORMS robots was added to the topics. The problems were selected such that they could be developed by multiple students over half a semester (3 months) and provide a certain degree of complexity.

3.3 Group Composition

First, each student selected either topic A or B or C. Then, in 2002, the students were divided into 9 groups of 15–17 students for topic A and 11–12 students for topic B. Based on the results of the experiments conducted in the first semester of 2003, 2004 and 2005 [3], students were divided into groups of 5 to 10. The number of groups in 2003, 2004 and 2005 was 7, 9 and 10, respectively. One leader and 1 or 2 subleaders were appointed for each group.

3.4 Plan

The schedules were planned by considering a time period of 3 to 4 weeks for each phase. The time for each class was 180 min; this was equivalent to 2 credits, and 14 classes were conducted. The groups made presentations (with an explanation and demonstration) in the final class. In 2003, 2004 and 2005, the final inspection was conducted at the end. Each experiment was supervised by 2 instructors and 2 Teaching Assistants (TAs) (4 TAs in 2004 and 6 TAs in 2005).

4. Design of Software Development Process

4.1 Phase

The software development method in our experiment involves object-oriented development using UML and Java. In this experiment, we divide the software development process into five phases: requirement analysis, systems analysis, system design, implementation, and testing. The goal of each phase is to create the following products, as listed in Table 1.

Table 1. Goals and products for each phase.

Phase/Goal	Contents of Products
Requirements Analysis	Requirement descriptions Use case diagram (*) Use case descriptions User interface sketches System architecture plan Conceptual model [Activity diagrams (*)]
The goal is to analyze user requirements so as to make clear functions which the user needs and the system performance and architecture.	
System Analysis	Use case diagram (*) Use case descriptions Class diagram (*) Sequence diagrams (*) [State diagrams (*)] [Object diagrams (*)]
The goal is to make clear the system components and their responsibility so that they can meet the results of requirement analysis.	
System Design	Class diagram (*) Sequence diagrams (*) Package diagrams (*) [Component diagram (*)] [Deployment diagram (*)] Integration testing specification Unit testing specification Testing plan
The goal is to define the program architecture so that the target system can be implemented by the specific system architecture, including the operational environment and the programming language.	
Implementation	List of source codes Source codes Installation guide User manual
The goal is to define the executable modules that meet the program architecture defined in the system design by the specific programming language.	
Testing	List of testing programs Testing programs [Package diagram (*)] List of testing cases List of testing results
The goal is to verify that the implemented programs meet the required functions and performance. The verification is performed in reverse order of the development process.	

[] denotes optional products. (*) denotes a UML diagram.

4.2 Task and Category of Tasks

In order to create valid products at each phase, the instruction guidelines state the tasks that must be performed during each phase. Each concrete work item is selected from a set of tasks. As shown in Figure 1, the tasks are divided into four categories: work done at the beginning of a phase (direction setting), work for describing the product (definition), work done for discovering mistakes or conflicts present in the defined product (verification), and work for creating the product based on the revised

definition (wrap-up). A concept of category was introduced for the purpose of clarifying the meaning of each task.

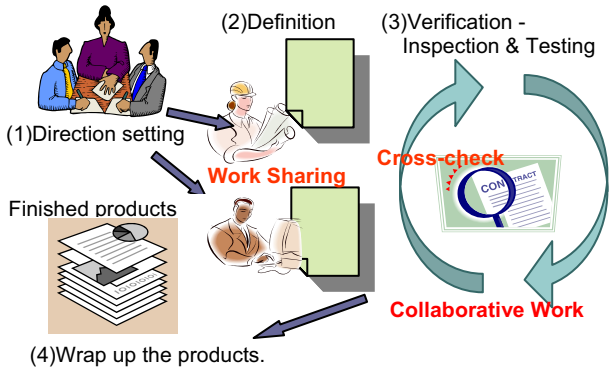


Figure 1. Category of Tasks

As shown in Figure 2, a phase comprises a row of tasks belonging to different categories. The verification tasks comprise the inspection of artifacts and program testing.

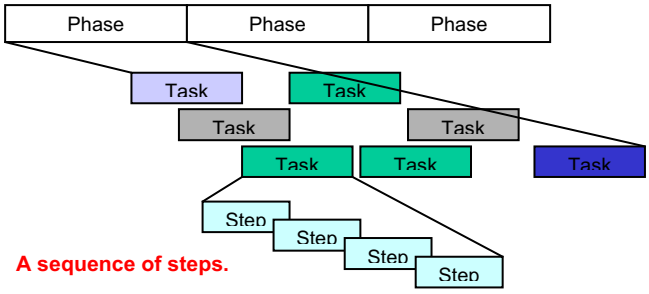


Figure 2. Composition of a phase

For example, at the system analysis phase, in order to make clear the system components and their responsibility, the following tasks are performed in appropriate order. As "direction setting" tasks, object candidates are detected from use case description and conceptual model that were defined at the requirement analysis phase. As "definition" tasks, a class diagram is elaborated through describing both sequence diagrams for every use cases and state diagrams for some objects. As "verification" tasks, the consistency and the validity between all of the sequence diagrams, state diagrams and the class diagram are inspected by members of a group. As "wrap up" tasks, the artifacts shown in Table 1 are defined as intermediate products at each phase. The underline parts represent example tasks at the system analysis phase. Each phase is performed as follows: "direction setting" → "definition" ↔ "verification" → "wrap up". The symbol ↔ denotes iteration of tasks. It is a feature of an object oriented development that some products are evolved from the ones at the previous phase. Not only the consistency within each product, but also the consistency between several products must be checked at the "verification" task. Moreover, the other members except a member who defined the artifacts should perform the task. Thus, the inspection is important collaborative work to improve the quality of artifacts.

The metrics for measuring each task based on the characteristics of the product were introduced in order to allow teachers to objectively monitor the individual work situations. Each student performed a process of concrete work. As a result, each task represented a sequence of his intended steps. The goal is to devise an efficient process for creating high-quality products. Students are required to understand the purpose of each task and perform a valid sequence of steps for each task.

4.3 Inspection Guideline

The target artifacts to be inspected are the products defined by "definition" tasks at each phase. We must separately inspect each product and its components for the internal validity. Moreover, the consistency between some products needs to be inspected. For example, the consistency of a class diagram is checked by sequence diagrams and state diagrams related to it. We provided the inspection guidelines and demonstrated to the students the manner in which the artifacts could be modified. Table 3 shows a guideline of the inspection at the system analysis phase.

4.4 Work Plan and Work Reports

A group prepares a work plan based on the tasks involved in each phase. A work item is an instance of a task for each group. Once the work plan is made, the students are required to submit a work report for each work item they are in charge of at least once a week. In the work report, the students enter the work period, state of progress (delayed, normal, ahead of schedule, finished), work history (content, work time), a sequence of steps, problems, and comments.

5. Collaboration Support System

5.1 Group Work Support System

In order to support the work model for collaboration between members, a group work support system was developed and applied to the experiment [1,2] in these three years. This system achieves awareness [8] within asynchronous groupware (i.e., understanding issues such as the work situation of cooperating workers and work history) by providing a forum for reporting, notifications, discussions, and questions related to work content. Through this forum, students can share information even beyond the class time and determine the work situation of other students. In order to support work beyond the class time, the system must be realized as an environment that can be accessed at any time, irrespective of whether students are present at school or not. This system has the ability to manage four forums as a web application: periodic work reports based on work items that specify the person in charge and the work period, minutes of meetings, discussion via the bulletin board, and a repository for artifacts and products. Our group work support system has some useful collaborative supports. However, this paper

describes inspection support system that the task itself has a strong effect on improving the quality of products.

5.2 Inspection Support System

The goal of inspection in the software development is to improve the quality of artifacts through iterative tasks that perform by the several developers. Among them, one who has defined the artifact plays a role of a client and the others play roles of reviewers. The task of inspection must proceed according to the following inspection guideline.

- The inspection team must contain a few reviewers who are not the client.
- Comments of each reviewer must be recorded for each target artifact separately.
- The client must answer each comment separately. The answer must contain the following decision of inspection: revising the artifact according to the comment, rejecting the comment and reserving the decision. Moreover, the reason of the decision must be clarified.
- All of the comments must be recorded so as to be referred later.
- In case of inconclusive debate about the inspection, the member may consult with an adviser. In this experiment, a teacher or TA plays an adviser.

The inspection support system offers a template in which the members can input a request or the comments or the decision according to the inspection guidelines.

6. Discussion

In 2005, the inspection support system was applied to the experiment. Based on the logs of inspection task and the results of a questionnaire administered after the experiment was finished, we discuss how a task of inspection could deepen the student's awareness of software development. The total number of answers was 60.

6.1 Effectiveness of Our Design

The software development process in our experiment has several important keywords which were mentioned in section 4. The following answers indicate the understanding and the effectiveness of these keywords.

Q: Did you perform each task at each phase after understanding the following keywords?: five phases, four categories of tasks, a set of tasks at each phase, a sequence of steps, metrics of a task and contents of inspection task.
A: Enough (3%), Pretty good (53%), Neither enough or not (22%), Seldom (17%), Not at all (0%)

Q: Which keyword was difficult to understand? You can select two or more answers.

a. Five phases (3%)
b. Four categories of tasks (13%)
c. A set of tasks at each phase (8%)
d. A sequence of tasks (14%)
e. Metrics of a task (46%)
f. Contents of inspection task (9%)
g. Nothing (9%)

Q: After the experiment was finished, have you understood the above mentioned important concepts about software development process?
A: Enough (8%), Pretty good (83%), Neither enough or not (8%), Seldom (0%), Not at all (0%)

Q: Why could you deepen your awareness of software development? You can select two or more answers.

a. All tasks that must be performed at each phase had been prescribed. (16%)
b. Metrics of each task had been provided. (8%)
c. All tasks had been categorized into four types. (10%)
d. We performed a task of inspection at each phase. (16%)
e. We experienced the software development for three months. (31%)
f. We had to plan our work plan by selecting appropriate tasks which were prescribed. (11%)
g. In the work report, we had to report value of each task metrics. (2%)
h. In the work report, we had to report a sequence of steps in which we had accomplished the task. (6%)

6.2 Analysis of Inspection Logs

At the requirement analysis phase, there were many comments on use case descriptions as follows: Errors in prior and posterior conditions of a use case (11%). A lack of essential steps in a use case description (14%). Duplicated steps in the several use case descriptions (7%). The others were comments on differences requirement readings (17%) and unifying vocabulary in the description (6%).

At the system analysis phase, there were many related comments on the degree of understanding of UML as follows: Errors or lacks of operations in a class definition (43%). Errors or lacks of classes in a class diagram of the system (16%). Lacks of alternative sequence diagrams or exception sequence diagrams for a use case (13%). The others were comments that vocabulary and notation were not unified in the description.

In this experiment, we had provided the guidelines of inspection to the students. However, as a result of observing inspection logs, we can not always say that their comments were valid and useful.

The ratio of valid comments in the inspection logs that recorded on the inspection support system was 44%. This is caused by lacks of their understanding of UML. We could see the fact by the result of an examination which was performed at the end of system analysis phase. The examination was to test how to read a sequence diagram being described in their submitted product. Moreover, the consistency between the sequence diagram and the class diagram was also examined. In full marks, 80 or more points were 32%, 60 or more points were 26%, and less

than 60 points were 42%. Table 2 shows the number of the valid comments on inspection task for each group.

Table 2. The number of comments.

Group	A1	A2	A3	A4	B1	B2	B3	C1	C2	C3
Number	116	70	27	9	38	8	12	9	21	2

There were some different ways to decide a role of the inspection task as follows. A group leader decided each role (30%). A role was decided in turn (25%). All members always attended all of the inspection tasks (25%). In the case of the first way, almost all they say that it was a good way for deciding a role. In the case of the second way, almost all they say that it was a bad way for deciding a role. The reason is that nothing was inspected when all the members had poor knowledge of UML. It may be a good way to assign a role of a reviewer to a member who has richer knowledge of UML than the other members. Actually, as shown in Table 2, for the sake of such assignment, the group A1 could give many valid comments to the client.

7. Conclusion

The goal of software development is to create a high-quality software system that meets user requirements. We must devise an effective process in which all members of a group can collaborate with each other. The students could understand several important keywords of software development through experience of our software development process on this experiment. They seem to have strongly recognized the importance of inspection for improving the quality of products. However, lack of the modeling ability using UML ended in giving few effects for improving the quality of products. Not only performing a task of inspection, but also recording inspection process is important for improving the quality of products. In order to let the students know this fact, we think that iterative development process needs to be elaborated.

References:

[1] Matsuura, S., Practical Software Engineering Education based on Software Development Group Experiments, Proc. of E-Learn 2005, pp.1701-1708, 2005.

[2] Matsuura, S., An Evaluation Method of Project Based Learning on Software Development Experiment, Proc. of the 37th ACM Technical Symposium on Computer Science Education, pp.163-167, 2006.

[3]Alfonso, M.I. and Mora, F., Learning Software Engineering with Group Work, Proc.of the 16th Conference on Software Engineering Education and Training,2003.

[4] M. E. Fagan, Design and Code Inspections to Reduce Errors in Program Development, *IBM Systems Journal*, 182-211, 1976.

[5] Travassos, G.H., Shull, F., Fredericks, M. and Basili, V.R., Detecting defects in Object-Oriented Designs: Using Reading Techniques to Increase Software Quality, Proc. of OOPSLA, pp.47-56, 1999.

[6] L. G. Votta, Does Every Inspection Need a Meeting?, *ACM SIGSOFT Software Engineering Notes*, Vol. 18, No. 5, 107-114, 1993.

[7] UML: http://www.omg.org/uml/

[8] Dourish, P. and Bellotti, V., Awareness and Coordination in Shared Workspaces, Proc. of the ACM CSCW'92, ACM, pp.107-114, 1992

Table 3. Inspection guidelines.

Inspection Target	Contents of inspection task
Use case diagram	Check on UML Notation Trace of Functional Requirements Comparing with the requirement description specified at the requirement analysis phase, it is checked that every use case meets the functional requirements. As the need arises, add a proper new use case, or delete the redundant use case. Check a Use Case Diagram using Work Flows It is checked that some use cases meet a work flow that is performed by an actor. As the need arises, add a proper new use case as an activity, or delete the redundant activity. Trace of Non Functional Requirements Comparing with the requirement description specified at the requirement analysis phase, it is checked that every use case meets the non functional requirements. As the need arises, modify some use case descriptions properly.
Sequence diagrams for each use case description	Check on UML Notation Trace of a Main Flow in a Use Case Description Comparing with the main flow description and the sequence diagram, it is checked that the use case description is enough to perform the use case. As the need arises, modify a use case descriptions or a sequence diagram properly. Extract some new proper objects. Change a class name or an operation name properly.

	Trace of an Alternative Flow in a Use Case Description Comparing with a sequence diagram for a main flow and a sequence diagram for the alternative flow, it is checked that there is no inconsistency between a class compositions. As the need arises, modify a use case descriptions or a sequence diagram properly. Extract some new proper objects. Change a class name or an operation name properly. Trace of an Exceptional Flow in a Use Case Description Comparing with a sequence diagram for a main flow and a sequence diagram for the exceptional flow, it is checked that there is no inconsistency between a class compositions. As the need arises, modify a use case descriptions or a sequence diagram properly. Extract some new proper objects. Change a class name or an operation name properly.
State diagrams for each object	Check on UML Notation Validity of the State of an Object It is checked that all of the states are enumerated for every object which must have a state. As the need arises, add a new proper state or delete an inadequate state, or change a name of a state properly. Validity of the State Transition It is checked that all of the state transition are valid from a view point of the component definition. As the need arises, add a new proper activity /action /event/guard, or delete an inadequate activity /action /event/guard, or change a name of activity /action /event/guard.
Class diagram	Check on UML Notation Validity of a Name of a Class Comparing with the concepts grasped at the use case analysis, it is checked that all the class names are valid and sufficient to the system. As the need arises, add a proper new class name, or delete an in adequate class name, or change a name of a class. Consistency between a Class Diagram and Sequence Diagrams Comparing with all of the sequence diagrams and a class diagram, it is checked that every operation exists in the correct class. As the need arises, change a name of an operation or a role of the operation properly. Consistency between a Class Diagram and State Diagrams Comparing with all of the state diagrams and a class diagram, it is checked that every operation exists in the correct class. As the need arises, change a name of an operation or a role of the operation properly. Validity of the Responsibility of a Class (Name) Comparing all attributes and operations with the class, it is checked that their name is valid in the class from the view point of the class responsibility. As the need arises, change a name of a class/attribute/operation properly Validity of the Responsibility of a Class (Role) It is checked that both the number of attributes and the number of operations in a class are valid from the view point of the class responsibility. As the need arises, divide the responsibility into several proper responsibilities, or put them into one class. Change a name of a class/attribute/operation properly. Validity of the Relationship between Classes in a Class Diagram It is checked that all the relation between classes are valid from the view point of the role and multiplicity. As the need arises, modify the relation properly.

Enhancing Semantic Interoperability in Collaborative Systems

Flavio De Paoli and Marco Loregian
DISCo, University of Milano-Bicocca, Italy
{depaoli,loregian}@disco.unimib.it

Abstract

A key successful factor for a knowledge management system is to promote and support semantic communication among people. This goal can be accomplished by integrating personal knowledge management tools with sharing tools that allow for merging semantic perspectives to create unified views, adapting behaviors to context, and finally learning from conversations to promote knowledge diffusion and acquisition. CADO is an open framework that provides for different levels of integration to support collaboration among members of an organization.

1. Introduction

Workers in a modern organization need to organize, share, and retrieve their personal knowledge. Many knowledge management (KM) systems suggest the use of ontologies to support the process of classification by means of metadata associated with documents. An issue is that such ontologies are often personal and need to be integrated to let groups of people share (part of) their knowledge. This knowledge exchange may occur on permanent bases (e.g., among people working on the same project or belonging to the same community of interest) or occasionally (e.g., during formal meetings or informal conversations).

In the project MILK, a solution was developed to provide such kind of workers with a centralized environment to let them share ontologies, documents and documents' profiles [4, 6]. The goal of this paper is to move from a centralized system to a peer-to-peer one that lets a worker create his/her personal installation of the MILK KM system and then integrate it with the ones of the other workers. The CADO framework (*Context-aware Applications with Distributed Ontologies*, [7]) provides for the basic mechanisms and policies to address these issues. According to the MILK features, information navigation and retrieval across knowledge bases is accomplished according to the *context* in which it occurs. The *context* is a situation composed of a technological and social environment, and application objectives. This is derived from a well-known definition of context: "the location, identity and state of people, groups and computational and physical objects" [3].

We are interested in clusters of users that can communicate directly or remotely by means of their devices. In the former case, they share the same physical space and perceive the same environment. In the latter, they do not share a physical space, therefore they are in what we call a "virtual room", which means that participants can concur in the creation of a common context even when being in different locations.

A cluster of workers in a particular room for a project meeting is an example of direct communication that can occur by wireless connections (e.g., Bluetooth and WiFi). This communication is augmented by the CADO framework that lets them share the knowledge stored in their laptops. CADO enforce semantic interoperability to provide users with rich contextualized descriptions to promote

awareness of available knowledge related to people and documents involved in the cluster.

The context is explicitly defined by a set of different (but interconnected) ontologies, each one describing a specific aspect that participates in the definition. For example, the context could be defined by involved devices, social goals, communication channels, people's roles and interests. Note that we have said "could" instead of "is" to outline that CADO allows the user or the application to define what a context is by selecting the interesting topic ontologies among the available ones. The merging of such ontologies identifies the actual context model on which the application can rely.

The next section describes the CADO architecture. Section 3 discusses the semantic interoperability issues. Section 4 is dedicated to related works and finally Section 5 draws some conclusion and outlines future works.

2. Framework architecture

The CADO framework defines a set of distributed components (*peers*) that can connect with each other, according to their type, to form *clusters*. Peers belonging to a cluster can run on the same *node* (i.e., the same computer) or be spread over different nodes. Components are classified according to the layered architecture described in Figure 1 that assumes two computers running all the needed components (e.g., two laptops of collaborating colleagues). Every layer provides for interoperability at different level of abstraction, from physical to conceptual. The *physical* interoperability layer lets the two nodes communicate by addressing issues like message dispatching, fault tolerance, and user authentication. The *semantic* interoperability layer provides support for data processing by providing an integrated view over the involved ontologies. The *context* interoperability layer supports the selection of data and behavior according to specific situations. Finally, the *interaction management* layer supports the definition of the application logic and of the actual interfaces and interaction.

Primary components in CADO are *Interaction Managers* (IM), *Context Managers* (CM), *Ontology Managers* (OM) and *Ontology Adapters* (OA). The set is completed by additional components that supply specific services such as the MILK Metadata Management System (MMS) components [6], and Lightweight-Directory-Access-Protocol (LDAP) components.

Figure 2 shows, with a sample situation, how peers can connect and how they are layered. The interaction layer is composed of interaction managers that support various devices (some with high computing capabilities, like PCs and laptops, and some with low capabilities, like digital pens and mobile phones). Interaction managers are connected to context managers. Context managers are connected to ontology managers that in turn are connected to ontology adapters. At different layers, peers are connected to form a multi-tier application (vertical connections). At every layer, peers can communicate and coordinate to each other (horizontal connections). Oval lines indicate clusters of peers forming (virtual) rooms. In room 1, we assume four devices (a laptop, a mobile phone, a large screen, and a digital pen) that share the same context and are coordinated by the same context

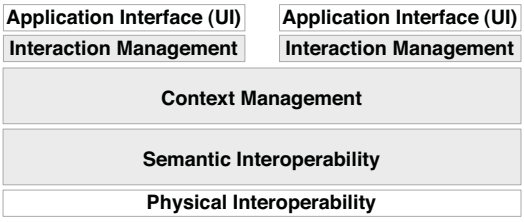


Figure 1. Framework architecture.

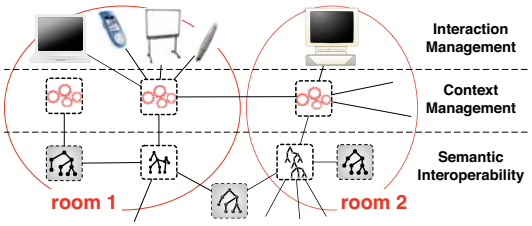


Figure 2. Sample configuration.

manager. This could be the situation in a meeting room where workers, carrying some personal devices, are discussing in front of a large screen (e.g., a Smart whiteboard). Room 2 is an example of remote peers.

Interaction Managers are user agents with application-specific or device-specific capabilities that provide users with interaction features and perform cluster discovery and set up. An IM controls the user interface layout and the content information displayed according to context information supplied by context managers. Inside a (virtual) room, IMs behave according to the technological and social perspective. For example, an IM controls the the large screen that acts as a shared display in the face-to-face meeting in room 1 (Figure 2).

Context Managers enact clustering mechanisms and define specific "policies" to support the behavior of the participants. Such policies are specified by a set of ontologies [7]. In the example of Figure 2, the four devices in room 1 are coordinated by the central CM according to its perception of the environment. CMs are contacted by IMs when they fall in range according to casual discovery protocol (e.g., the one provided by JXTA technology) or when two or more persons decide to cluster. IMs and CMs communicate to each other to establish the *context* for the cluster, and elect a coordinating CM that will be in charge of routing information among participating IMs. Connection and cluster capabilities are constrained by device capabilities. Devices with advanced connectivity support and high performance computing power (e.g., PCs, laptops) take care of simpler devices (e.g., sensors, RFID readers, digital pens, interactive screens) by driving their behavior to supply users with rich interactive environments.

A clustering example is shown in Figure 3, in which a newcomer actor joins a cluster. The joining protocol can be summarized as follows [7]: when an IM gets in contact with a set of CMs, peers configurations are exchanged so that the IM knows the available technological contexts (i.e., where CMs are located and the IMs they already cluster). The IM selects a CM according to application policies (or under explicit direction of the user) and the user

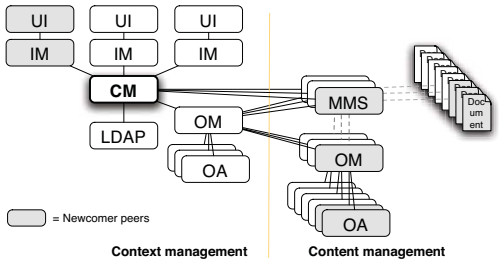


Figure 3. Example of cluster.

is authenticated by the CM contacting an LDAP peer (if required). The IM sends its configuration to the CM, specifying the MMS component in charge of managing user's documents. The CM will provide an integrated view over all the involved MMSs supplying IMs with document profiles. Profiles include URIs through which IMs can retrieve the actual files. The MMS supplies the CM with information on its reference ontologies. Such ontologies will be integrated with the CM ontologies (Figure 3) according to the semantic interoperability features described in the next section.

3. Semantic interoperability

According to a well known definition, an *ontology* is an explicit specification of a conceptualization [9]. An ontology is composed of a hierarchical taxonomy, additional relations between concepts, concept descriptions (attributes), axioms (constraints) and instances. CADO users adopt ontologies to cover different aspects of the domains of interest, that can be independent (e.g., work practices and product specifications). We call **topic ontology** the part of an ontology that covers a particular aspect in a domain, collecting a set of concepts connected with a *is-a* relation and with other topic-specific relations (later referred to as *intra-topic relations*). Therefore, a domain is covered by ontologies on different topics, that are interconnected via domain-specific relations (later referred to as *inter-topic relations*). In CADO, we consider *distributed ontologies* that are built from individual representations: "[a distributed] ontology is divided into sev-

eral component ontologies and [...] each ontology [is constructed] individually (perhaps in parallel) by different developers in a distributed environment" [14]. In other words, distributed ontologies are modular and managed by peer components spread over a network.

Since knowledge is manifold, different users exploit different topic ontologies to cover their entire knowledge and expertise. Therefore, a collective vision over a topic within a group of individuals, such as a work team or a community, can only be achieved by merging (different) topic ontologies. Ontology merging is the "creation of [a] single coherent ontology that includes the information from all the sources" [12]. In CADO only *virtual* merging is employed, in that no physical ontology is produced by merging topic ontologies, but only a temporary *view* over different sources seen as a whole.

Given two (or more) ontologies to be merged, corresponding concepts in the taxonomies of the ontologies have to be identified: this requires a similarity measure for taxonomies (i.e., a matching criterion). When similarities are identified, the corresponding concepts have to be merged preserving relations — taxonomy *is-a*, concept-specific attributes, ... — and tracking origins to remember the source ontologies. In summary, according to literature [11], ontology merging consists of (1) finding places in the ontologies where they semantically overlap — either extensionally, i.e., concepts with the same label or synonyms, or intensionally, i.e., concepts with the same place in the taxonomy — (2) check the consistency, coherency and non-redundancy of the result. Iteration of such steps might be required.

In the current implementation [7], a basic ontology merging has been implemented as default mechanism. It starts with the merging of topic ontologies. For every topic ontology the activities are: (1) visit the taxonomy to identify overlapping concepts; (2) built the new taxonomy including all the concepts in the original taxonomies; (3) add any other intra-topic relation to deliver the merged topic ontology. The ontology merging is then completed by adding inter-topic relations according to the original ontologies. During merging, simple checks are performed to avoid cycles and replications. Note that tackling the merging problems by considering topic ontologies ensure an higher degree of confidence, than considering generic taxonomies due to the high coupling of involved concepts. CADO allows for merging customization by overriding similarity and check procedures with tailored matching criteria (e.g., translations, synonym analysis, WordNet-mediated mapping).

In the CADO framework, the merging tasks are performed by ontology *managers* that rely on ontology *adapters* to access single modules. **Ontology Adapters (OA)** manage a single ontology module, such as an individual topic ontology; supply a standard interface to access the managed ontology as an OWL ontology (can be wrappers of exiting ontologies); manage local ontology evolution. **Ontology Managers (OM)** accomplish the merging of different ontologies; rely on modules managed by OAs. Such components can be connected to form a hierarchy with OAs as terminal components . Ontology Managers identify and select candidate ontologies according to following policies:

- **Manual selection policy**: OMs are explicitly supplied with ontology URIs at runtime.
- **Static selection policy**: OMs are configured to merge a given set of ontologies, which are identified by URIs.
- **Automatic selection policy**: OMs encompass application logics to identify ontologies of interest.

Static selection policy could be adopted to refer to normative ontologies (e.g., standards, reference ontologies of an organization) and to personal ontologies. An automatic selection policy could be adopted for looking for specific topic ontologies (e.g., device and spatial ontologies) according to the definition of context given by the context managers. The logics can be programmed by extending the default mechanisms or by supplying a sample taxonomy (e.g. the one adopted by the actual user of the system). In the latter case, the default is to look for some similar concepts (by exact matching) in candidate ontologies accessed by reachable OAs.

The described merging activity can affect also the merged ontologies that can *learn* from the others some new concepts. This mechanism allows for knowledge diffusion and promote ontology evolution. The process is managed by the OAs according to the following policies:

- **Normative policy**: an ontology can only be modified by authorized (usually human) administrators. This policy is usually adopted for reference ontologies (e.g., an organization's ontology) that are maintained off-line.
- **Selective policy**: an ontology can be modified on external requests (from OMs), but only by authorized users and possibly with explicit authorization from the owner. This policy could be adopted to deal with sensitive ontologies that reflect user's knowledge representation.
- **Plastic policy**: an ontology can be freely modified by OMs. This policy is adopted by flexible nodes that need to reflect external events. For example an ontology related to the social interactions should adopt such a policy.

4. Related Work

A requisite for sharing is to make people able to understand each other, which means that every participant is asked to contribute to the creation of an accessible common knowledge base. The problem of achieving semantic interoperability among different actors (and their devices) by exploiting online resources (i.e., ontologies [5]) is a hot topic in literature [8], grounding for example in Artificial Intelligence research on ontology mapping and merging. Several tools and systems have been proposed for concept mappings and ontology merging. Popular examples are the PROMPT suite [12] and Chimaera [11] that are semi-automatic merging tools. Important issues are consistency and resolution of semantic conflicts. Among possible strategies, we can mention the SCROL ontology [13] and collaborative design [10]. To tackle these problems, CADO provides only basic mechanisms (see Section 3) that can be replaced or customized to meet expected levels of accuracy and automatization.

From an architectural perspective, a good example of semantic interoperability is provided by Aberer et al. [1, 2] that proposes a framework addressing semantic agreement between peers as a local issue rather than global and centralized. Agreement is achieved by making expert users explicitly specify mappings between individual schema. The transitive propagation of manually expressed mappings is defined as *semantic gossiping*. Automatic processes to establish semantic mappings are not envisioned, whereas in CADO this option has been considered (e.g., according to [13]). Further, (portions of) topic ontologies are propagated gossip-style, in a selective way that does not necessarily require domain expert intervention. Moreover, the CADO users (and system designers) can specify policies to regulate the access to ontologies (for Ontology Adapters) and decide how ontologies have to be clustered (for Ontology Managers).

For space reasons, this paper does not cover the context management and application interoperability (i.e., interaction management) layers with details. Both of them have been designed to meet openness requirements. Openness at context management level means that how reasoning is enacted (e.g., by means of which inference engine, or constraint validation technique) is not strictly imposed by the framework, but it is also customizable (also according to policies adopted at ontology management level).

5. Conclusion and Future Work

The CADO framework overcomes the traditional peer-to-peer frameworks that allow for file exchange and repository replication. CADO add semantic layers with the main purpose of enabling knowledge circulation and, at the same time, to support personalized knowledge representation and organization. System designers can customize the default mechanisms of CADO to accommodate specific policies for clustering, context definition, ontology merging, profiling and retrieval of knowledge. A first level of customization can be achieved

by means of configuration files that can be edited by the final users, a deeper customization can be achieved by means of plug-in techniques that enable for the replacement of the framework logics. The great advantage is the high flexibility of CADO that can fulfil a variety of requirements in a single and coherent environment.

The MILK system is an example of a new generation of applications with richer knowledge representation and advanced interaction protocols. The CADO framework can be considered to be an evolution of the MILK approach to encompass distributed representation of knowledge and more various and contextualized interaction patterns and information presentation. By decoupling information presentation from interaction, context, and knowledge base management, the framework is open to the design of new interactions and to the development of device-specific, as well as situation-specific interfaces.

Future work will finalize the current prototype that is based on JXTA p2p technology and Protégé/OWL ontology managers. The goal is to deliver a framework that can be used by researchers in different fields to experiment advanced solutions on the more important aspects: knowledge management (e.g., ontology management, knowledge navigation, knowledge retrieval), social interaction (e.g., collaborative work, knowledge sharing, rich visualization), and finally ubiquitous and pervasive computing (e.g., new devices, embedded devices, seamless transition between contexts).

Acknowledgement This work was partially supported by the MILK project (IST-2001-33165).

References

[1] K. Aberer, P. Cudré-Mauroux, and M. Hauswirth. A framework for semantic gossiping. *SIGMOD Rec.*, 31(4):48–53, 2002.

[2] K. Aberer, P. Cudré-Mauroux, and M. Hauswirth. The chatty web: emergent semantics through gossiping. In *WWW '03*, pages 197–206, New York, NY, USA, 2003. ACM Press.

[3] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In *HUC '99*, pages 304–307, London, UK, 1999. Springer-Verlag.

[4] A. Agostini, S. Albolino, R. Boselli, G. De Michelis, F. De Paoli, and R. Dondi. Stimulating knowledge discovery and sharing. In *GROUP'03*, pages 248–257. ACM Press, 2003.

[5] H. Alani. Ontology Construction from Online Ontologies. In *WWW2006*, (in print) 2006.

[6] R. Boselli, R. Dondi, and F. De Paoli. Knowledge organization and retrieval in the MILK system. In *SEKE 2003*, pages 372–376, 2003.

[7] F. De Paoli and M. Loregian. Context-aware Applications with Distributed Ontologies. In *Ubiquitous Mobile Information and Collaboration Systems (UMICS 2006)*, volume in print of *LNCS*. Springer-Verlag, 2006.

[8] M. Ehrig, S. Staab, and Y. Sure. Bootstrapping ontology alignment methods with APFEL. In Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, editors, *ISWC 2005*, volume 3729 of *LNCS*, pages 186–200. Springer, 2005.

[9] T. R. Gruber. A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5(2):199–220, 1993.

[10] C. W. Holsapple and K. D. Joshi. A collaborative approach to ontology design. *Commun. ACM*, 45(2):42–47, 2002.

[11] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *KR2000*, pages 483–493, 2000.

[12] N. F. Noy and M. A. Musen. PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In *AAAI-2000*, pages 450–455. AAAI Press / The MIT Press, 2000.

[13] S. Ram and J. Park. Semantic Conflict Resolution Ontology (SCROL): An Ontology for Detecting and Resolving Data and Schema-Level Semantic Conflicts. *IEEE Trans. Knowl. Data Eng.*, 16(2):189–202, 2004.

[14] E. Sunagawa, K. Kozaki, Y. Kitamura, and R. Mizoguchi. An environment for distributed ontology development based on dependency management. In *ISWC 2003*, volume 2870 of *LNCS*, pages 453–468. Springer, 2003.

Combining AI Techniques into a Legal Agent-based Intelligent Tutoring System

Ig Bittencourt, Marcos Tadeu, Evandro Costa
Federal University of Alagoas - Instituto de Computação
Tabuleiro dos Martins, Postal Code 57.072-970 Maceió - AL
Email: {ibert, mts, evandro}@tci.ufal.br

Abstract

Computer based learning gets more and more important in higher education. Particularly, in Legal domain, students have little chance to deal with realistic situations. One way to alleviate this problem is to provide Law students with real cases, rules and different viewpoints of which a given body of knowledge is often recognized as important to their successful learning. We propose a novel approach to ITS applied to Legal domain in order to address each of the above concerns. Then, we define an agent-based architecture to support multiple views of domain knowledge, aiming to improve the quality of student-ITS interactions and the learning success of the students. Each tutoring agent from the system contains a knowledge-based system that combines case-based reasoning and rule-based system. In addition, each agent adopts the reinforcement learning Algorithm aiming to identifying the best pedagogical strategy by considering the student profile. This paper focuses on both architecture and the mentioned Artificial Intelligence techniques into a Legal System. Finally, an example scenario is shown to demonstrate the feasibility of our approach.

1 Introduction

Providing Law students with real cases, rules and different viewpoints of a given body of knowledge is often recognized as important to their successful learning. In fact, these requirements are hot topics and seem to be more realistic to be explored in the context of Legal Intelligent Tutoring System research. However, little research has been done on these aspects.

We propose a novel approach to ITS applied to Legal domain in order to address each of the above concerns. Then, we define an agent-based architecture to support multiple views of domain knowledge, aiming to improve the quality of student-ITS interactions and the learning success of the students. Here, learning is considered as problem-solving, where students solve problems and are individually assisted and guided by the system during the solution process. The system may also solve problems. CBR has been used for checking the similarity with old cases to justify new problems and RBS for evaluating the rules of Normative Knowledge. In addition, each agent adopts the reinforcement learning Algorithm aiming to identifying the best pedagogical strategy by considering the student profile.

The use of a hybrid solution to the problem solving has been motivated due the structure of the juridical system [1]. Legislation is the main Legal research, where magistrates making their decision based on the code and laws, originating case solutions. One of the best ways to solve Legal problems based on legislation is using rules, due this, rules is approached in the ITS. For this reason, Case Based Reasoning and Rule Based System make necessary to the ITS.

Some related works were developed taking into consideration legal tutoring or hybrid reasoning. In [1], is approached an intelligent learning environment designed to help beginning law students learn basic skills of making arguments with cases. In [11] was found an ITS for Legal domain, using rule-based system and approach problem-based learning as pedagogical strategy. Our proposal refers to a novel ITS approach applied to Legal domain, using hybrid reasoning (CBR and RBS), besides the modeling of multiple views of domain knowledge, providing two-way interaction in the problem solving. Others Legal approaches were found, however, were not focused as comparative works. These works have total relevance and some of them that can be cited are [10, 5].

In our approach, the idea is to engage Law students into interactions with ITS based on the resolution of Legal problems and their consequences on other tutorial activities, concerning the Civil Law. The starting point of these interactions occurs when ITS submits a penal situ-

[1] Civil Law, also known as Continental Law or Roman Law has been used in the system

ation to Law students. Then, they will learn two fundamental but different skills of Legal problems. First, know how to identify relevant cases and Legal concepts (normative knowledge, for instance) of the cases. Second, know how to use them effectively as examples justifying position in a Legal argument. When using an automated information retrieval system, one needs to use a query or a set of queries that captures the issues and the intended use of the cases in an argument[2]. Therefore, Case-based reasoning has been used to know what kind of cases can result better solutions and rule-based system to find better concepts, and consequently giving support for better explanations of the problem.

2 System Architecture and Implementation

Basic components in the architecture represents the classical ITS approach, however, details about specific agents in these components are explained in the next subsections.

The **Interface Agent** is responsible for communicating the student, providing access to Legal information such as jurisprudence and normative knowledge. The kinds of screens displayed are about student credentials, student information, problem, solution, solution evaluation and argumentation. In addition, a **Broker Agent** was inserted for mapping the information between the agents.

We adopt an ontology that represents the base of the Legal knowledge to be used within Intelligent Tutoring System.

2.1 Expert Module

The Expert Module was modeled as shown in Figure 1. In this subsection, details of the Domain Architecture are presented.

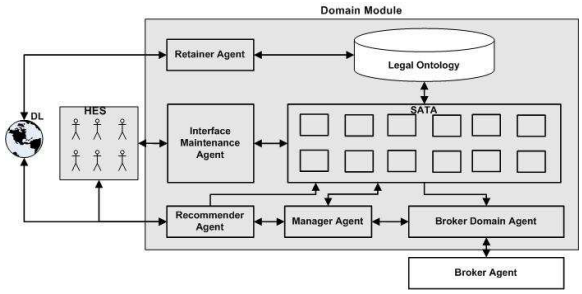


Figure 1: Expert Module Architecture

The ontology approached in our ITS was developed by using Protégé [12] tool, providing flexibility and a lot of plug-ins.

Figure 2 shows a relevant part of the legal ontology. In this figure, it is possible to see some concepts and their relationships. Ontology has been developed take into consideration MATHEMA Model, described into 2.1.1.

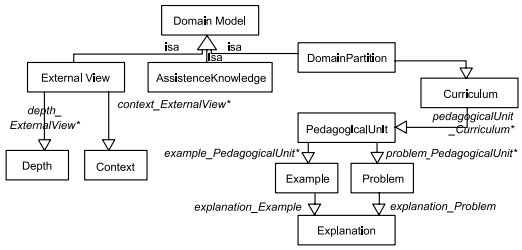


Figure 2: Ontology Description

Based on the Legal ontology developed to the ITS, a problem is defined as a 3-Tuple $\langle P, I, F \rangle$, where:

- P: represents the real penal situation;
- I: represents a set of interpretations (solutions) to the problem P. The related interpretations are based on:
  - Prosecutor view;
  - Lawyer view.
- F: P x I: represents the theoretical fundamentation in the relation P x I, that can be a Doctrine, Jurisprudence or web site.

2.1.1 Agents

- **SATA** is a society of artificial tutoring agents that represents an agent-based ITS acting in a specific domain. Each agent in SATA play pedagogical roles. These agents are responsible for problem solving and providing information of the student profile and pedagogical strategies. In the problem resolution, cooperation is one of the features of the SATA. Cooperation is approached by considering a multidimensional view, where the expert knowledge is distributed in each agent from SATA. The idea is the division of the domain knowledge into many sub domains, represented by agents. This division tries to orient a search in the expert knowledge distributed. In addition, each agent has a curriculum manager that chooses problems for the student, but it is optional.

SATA Reasoning: This subsection approaches the reasoning presents in SATA structure. Rule-based system and case-based reasoning are two known approaches adopted in the Expert Module Agents in the Intelligent Tutoring Systems (ITS). They are natural

alternatives in knowledge representation. Rules usually represent general knowledge, whereas cases encompass knowledge accumulated from specific (specialized) situations. Each approach has advantages and disadvantages. Due to their interchangeable nature, rules and cases can be integrated and thus produce effective ITS. Below characteristics of CBR and RBS reasoning are described.

CBR (Case-Based Reasoning) Reasoning

The CBR Reasoning is responsible for evaluating the similarity between the jurisprudence (inserted in the case base) and the penal situation sent by Law student.

The knowledge representation was done as a relational representation, with n attributes $A = \{a_1, a_2, ..., a_n\}$ where each attribute has a weight $W = \{w_1, w_2, ..., w_n\}$, for details of the knowledge representation and similarity functions, see [8]. The similarity function between two cases is defined as:

$$SIM(C_1, C_2) = \sum_{i=1}^{n} \left(w_i * sim(a_{i_{C_1}}, a_{i_{C_2}}) \right)$$

In both cases, the case retrieval was divided into two levels. First, it is used the similarity function for simple attributes (numeric and boolean, for instance) and more important index. Second, it is compared the complex attributes such as strings. This division turns the case retrieval faster than the sequential retrieval method. Adaptation and retention are not necessary because the jurisprudence can not be changed.

RBS (Rule-Based System) Reasoning

RBS (Rule-based System) Reasoning is responsible for the rules evaluation in the Legal ontology. The rules were modeled by considering the normative knowledge, which enables the whole validation of a penal situation. When the Law Student describes a problem, the system tries to infer about the features and map them in doctrine concepts defined in the ontology. In addition, were modeled 49 rules to infer of the domain. Follow an examples of rules developed in the Jess [6] environment and integrated within protégé[12].

```
(bind ?article new Article) (defrule concept
    ("corporalLesion")
    ?article getInstance() )
```

The interactions between the Law students and the ITS in the problem solving can happen in two ways: a) when the student submits a penal situation to tutoring system and b) when the tutoring system submits a penal situation to the student.

In both cases, can be use a hybrid mechanism of reasoning, CBR (Case-Based Reasoning) and RBS (Rule-Based System) working together with the legal ontology to solve problems submitted by the student or by the tutoring system. The interactions between the techniques can be structured as follows:

- CBR reasoning and RBS reasoning return the solution, exploiting the jurisprudence and normative knowledge researches;
- CBR reasoning returns the solution. This situation happens when the RBS reasoning is unable to infer from the given problem;
- RBS reasoning returns the solution. This situation happens when the CBR reasoning is unable to infer from the given problem;
- none of them. If this situation occurs, the system will send a message to the student, requesting better description from the given problem.

When the *student submits the tutoring system to a penal situation*, ITS tries to solve the penal situation integrating CBR and RBR. This interaction algorithm was implemented as follows

Algorithm 1 The EvaluationStudentProblem Algorithm

Initialize $Evaluate(studentProblem)$;
Initialize $RBSInfer()$;
$rbsSolution \leftarrow$ try infer from NormativeKnowledge;
Initialize $CBRCycle()$;
$casesBase \leftarrow$ select jurisprudence from Ontology;
Execute Retrieve from CBRCycle;
Select similarCase;
Select similarityValue;
$MountSolution(rbsSolution, similarCase)$;

Second, when the *tutoring system submits a penal situation to the student*, the student describes the solution as far as her/his concerning. After that, ITS evaluates the student solution according to the algorithm below.

Algorithm 2 The StudentSolutionEvaluation Algorithm

Initialize $Evaluate(studentSolution)$;
Initialize $CBRCycle()$;
$casesBase \leftarrow$ select casesSolution from Ontology;
Execute Retrieve from CBRCycle;
Select similarCase;
Select similarityValue;

- **HES (Human Expert Society)**: represents an external knowledge source integrated in SATA, which provides a kind of maintenance necessaries in the SATA, such as inclusion and exclusion of agents, besides give support to Law students. The support to the Law student is possible through the recommender agent.

- **Interface Maintenance Agent**: represents the communication between HES and SATA.
- **Manager Agent**: Responsible for all the flow interaction ITS-student. The tutoring process can happen as: a) guided, where the ITS gives assistance to the student about the curriculum or b) free, where the student choose the curriculum he/she intends to study or search about Legal Concepts as jurisprudence, Doctrine and Legislation. The characteristics of this agent are:
  - choose which SATA Agent will tutoring the student or give assistance;
  - manage kinds of pedagogical strategy are given to the student, like hints, warnings and others;
  - choose which SATA Agent(s) will solve a problem sent by the student.
- **Broker Domain Agent**: responsible for delegating actions to the Recommender Agent, Explainer Agent and SATA, besides send information to the Broker agent.
- **Recommender Agent**: responsible for providing supports to the Law student, providing additional researches aiming the improvement of learning process. The following are the kind of supports provided for the Recommender Agent:
  - HES (Human Expert Society) can be recommended to give support to the Law student;
  - information in a Digital Library can be sent to support the learning process;
  - an agent in SATA to supervise the student is recommended to provide better resolution about the problems and to help him/her with doubts concerning the domain knowledge.
- **Retainer Agent**: one of the biggest problems found in Case-Based Reasoning systems for the Legal domain refers to the jurisprudence that updates everyday. To solve this problem, a Retainer Agent in the intelligent tutoring system for evaluating of new jurisprudence in virtual libraries was added.

2.2 Pedagogical Module

Some researches have been doing into intelligent tutoring system, exploring Reinforcement Learning in pedagogical activities [3, 13, 9]. The Pedagogical Module was modeled as shown in Figure 3.

In order to improve the quality of the solutions and student learning, a Reinforcement Learning Algorithm was used concerning the strategies above.

- Increase the Problem Difficulty Degree;

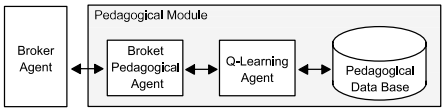


Figure 3: Pedagogical Module Architecture

- Decrease the Problem Difficulty Degree;
- Same Difficulty Degree;
- Change the Level;
- Change the issue;
- Change the Problem Issue to past issue.

Q-Learning algorithm is used after the definition of the student profile (features of the student profile was not focused), due this, in student-ITS interaction, the algorithm, through rewards, define and select the best strategy to use in a specific situation. The selection of strategy takes into consideration groups of profiles as shown in Figure 4.

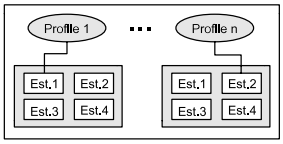


Figure 4: Q−Learning Strategy

2.2.1 Q-Learning Structure

The goal of the agent in a Reinforcement Learning Problem is to learn an action policy that maximizes the expected long term sum of values of the reinforcement signal, from any starting state [4]. In the present work, the problem is defined as a Markov Decision Process (MDP) solution.

The chosen of better strategies has been modeled as a 4-tuple (S, A, T, R), where:

- S: set of pair of strategy and MATHEMA Context;
- A: finite set of strategies;
- $T : S \times A \rightarrow \Pi(s)$: state transition function represented for the probability value, signalizing the betters strategy to be chosen;
- $R : S \times A$: is described as a utility value, defined for the similarity of the attributes, mapped as a reward function.

We use in the e-learning environment a proposal approached in [4], that implements an algorithm which is used in the action choice rule, which defines which action must be performed when the agent is in state s_t. The heuristic function included

$$\pi(s_t) = \begin{cases} argmax_{at} & \left[\hat{Q}(s_t, a_t) + \xi H_t(s_t, a_t)\right] \\ a_{random} & \end{cases} \quad \begin{matrix} if q \leq p, \\ otherwise. \end{matrix}$$

- $H : S \times A \rightarrow R$ is the heuristic function;
- ϵ: is a real variable used to weight the influence of the heuristic function;
- q: is a random uniform probability density mapped in [0, 1] and $p(0 \leq p \leq 1)$ is the parameter which defines the exploration divided for exploitation balance;
- a_{random} is a random action selected among the possible actions in state s_t;

Then, the value heuristic $H_t(s_t, a_t)$ that can be defined as:

$$H(s_t, a_t) = \begin{cases} max_a & \hat{Q}(s_t, a) - \hat{Q}(s_t, a_t) + \eta \\ 0 & \end{cases} \quad \begin{matrix} if a_t = \pi^H(s_t), \\ otherwise. \end{matrix}$$

Algorithm 3 The Heuristics Algorithm

Initialize $Q(s, a)$
Repeat:
  Visit the s state
  Select a strategy using the choice rule
  Receive the reinforcement $r(s, a)$ and observe the next state $s^{'}$.
  Update the values of $H_t(s, a)$.
  Update the values of $Q_t(s, a)$ according to:
  $\hat{Q}(s, a) = \hat{Q}(s, a) + \alpha \left[r + \gamma max \hat{Q}(s^{'}, a^{'}) - \hat{Q}(s, a)\right]$
  Update the $s \leftarrow s^{'}$ state
Until some stop criteria is reached,
where $s = s_t, s^{'} = s_{t+1}, a = a_t e a^{'} = a_{t+1}$

3 An Illustrative Example

To illustrate the functionality of the system we show a learner working with problem-solving of a penal situation approaching multiple views of the knowledge.

Imagine that the student works the first time in the ITS. The student answers a set of question about Legal issue and then, the level knowledge of the student is defined. Below, we exploit an example where the student submits a problem to the system.

3.1 Case

Problem: *John arrives in his home and see Maria and Joseph (John´s brother), sleeping in the bed, naked. Then John overdrew his gun and shot against Maria, which died.*

Solution: The solution is divided into two views: The Prosecutor view, where tries to increase the punishment and the Lawyer view that tries to decrease the punishment.

Prosecutor View:

- Normative Knowledge - Qualified Homicide: Art. 121, §2º, IV;
  Doctrine - Qualified Homicide can be used when happens a crime through research that makes difficult or impossible the defense or the offended person, by the fact of the victim been sleeping.
- Jurisprudence - Summary: JURI. Qualified Homicide. Research that turn defense of the offended person impossible. Victim Sleeping. [...]

Lawyer View 1:

- Normative Knowledge - Self-Defense: Art. 23;
  Doctrine - Self-Defense can be used when the author has his honor stained for the victim;
- Jurisprudence - Summary: Homicide - Self-Defense of the honor - Accused that, [...].
- Site: `http://jus2.uol.com.br/doutrina/texto.asp?id=980`;

Lawyer View 2:

- Normative Knowledge - Privileged Homicide: Art. 121, §1º;
  Doctrine - Privileged Homicide can be used when the author act through strong emotion;
- Jurisprudence - Summary: JURI. Qualified Homicide. Cohabitation. Condemnation for Privileged Homicide.

In the case, three solutions were returned to the ITS. Three SATA Agents were used to solve the case, where each solution represented one agent. The table 1 with the agents characteristics follows below.

Table 1: Agents characteristics

Agent	Context
SATA Agent 121_2	Crime versus live
SATA Agent 121_1	Crime versus live
SATA Agent 23	About the Crime

Other characteristic present in the example is that all SATA Agents knew solve the case using the hybrid reasoning. On the other hand, only the SATA Agent 23 found a site to guide the student with others knowledge of the solution.

The Figure 5 shows the solution retrieve of the system.

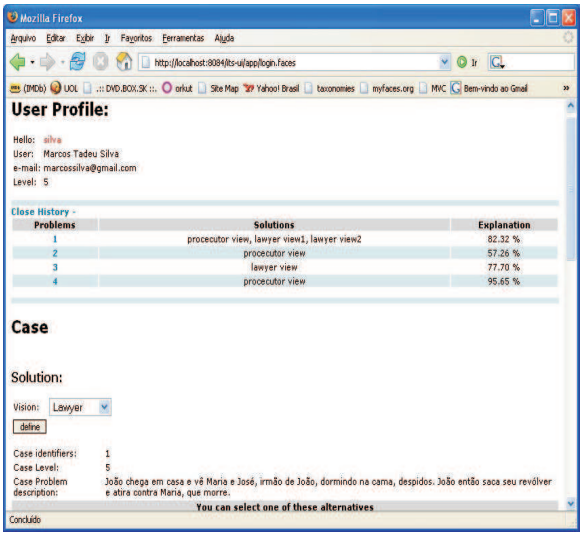


Figure 5: Prototype Resolution

4 Final Remarks and Future Work

This paper describes a hybrid ITS which provides human learning by problem posing to Law students and giving those appropriate tutorial feedbacks. The prototype has been used with three types of knowledge domain (Jurisprudence, Normative knowledge and doctrines). At the moment, we described the Case-Based Reasoning model and Rule-Based System that integrates Jurisprudence, Normative knowledge and doctrines and the application of the corresponding Legal concepts in the problem solving process. Technologies such as JADE[14], JESS [6], Protégé[12] were used on the development of the prototype. Each one of these tools can be considered as the state of art in its application domain.

Now, It has been developed the version 2.0 of the ITS. We have planned to this new version: a) create the strategy structure to the pedagogical model in others parts of the tutor; b) create the student modeling structure to the student model, providing a holistic view of each individual student is stored, allowing the tutor to be highly personalized [7]. Finally, we are planning to evaluate the current system with undergraduate students to improve the system's robustness.

5 Additional Authors

Alan Silva and Rômulo Nunes (Federal University of Alagoas, email: {alan,romulo}@tci.ufal.br)

References

[1] Vincent Aleven. Using background knowledge in case-based legal reasoning: a computational model and an intelligent learning environment. *Artif. Intell.*, 150(1-2):183–237, 2003.

[2] Vincent Aleven and Kevin D. Ashley. What law students need to know to win. In R. P. Jones and K. V. Russell, editors, *Proceedings of the Fourth International Conference on Artificial Intelligence and Law*, pages 115–129, Abingdon, UK: Carfax Publishing Co, 1993. University of Melbourne.

[3] Amy L. Baylor and Yanghee Kim. Simulating instructional roles through pedagogical agents. *International Journal of Artificial Intelligence in Education*, 15, 2005.

[4] Reinaldo A. Cosda BIANCHI. *Uso de heurísticas para a aceleração do aprendizado por reforço*. Tese de doutorado, Escola Politécnica, Universidade de São Paulo, 2004.

[5] Tharam Dillon Vivian Vossos George Vossos, John Zeleznikow. An example of integrating legal case based reasoning with object-oriented rule-based systems: Ikbals ii. In *ICAIL '91: Third International Conference on Artificial Intelligence and Law*, New York, NY, USA, 1991. ACM Press.

[6] Jess the rule engine for the java platforme, 2003.

[7] Vive Kumar and D. Brokenshire. An ontological framework to collect and disseminate user model data. In *I2LOR: ELearning for the Future: from Content to Services, 2nd Annual Scientific Conference of LORNET research network, Vancouver, Canada, accepted for publication*, 2005.

[8] Rosina Lee. *Pesquisa Jurisprudencial Inteligente*. Tese de doutorado, Programa de Pós-Graduação em Engenharia de Produção, Universidade Federal de Santa Catarina, 1998.

[9] Kimberly N. Martim and Ivon Arroyo. Agentx: Using reinforcement to improving the effectiveness of intelligent tutoring system. In *Intelligent Tutoring System, Maceió, Alagoas, Brazil*, pages 564–572, 2004.

[10] A. Oskamp, R. F. Walker, J. A. Schrickx, and P. H. van den Berg. Prolexs divide and rule: a legal application. In *ICAIL '89: Proceedings of the 2nd international conference on Artificial intelligence and law*, pages 54–62, New York, NY, USA, 1989. ACM Press.

[11] Georges Span. Lites, an intelligent tutoring system for legal problem solving in the domain of dutch civil law. In *ICAIL '93: Proceedings of the 4th international conference on Artificial intelligence and law*, pages 76–81, New York, NY, USA, 1993. ACM Press.

[12] Stanford. Protégé ontology editor and knowledge acquisition system, 2000.

[13] Joel Tetreault and Diane Litman. Using reinforcement learning to build a better model of dialogue state. In *11th Conference of the European Chapter of the Association for Computational Linguistics (EACL), Trento, Italy (to appear in April)*, 2006.

[14] Tilab. Java agent development framework, 2005.

Using Conditional Probability to Measure Rule-based Knowledge Similarity

Chin-Jung Huang [1], Min-Yuan Cheng [2]

[1]*Department of Mechanical and Computer-Aided Engineering, St. John's University, Taiwan*
[2] *Department of Construction Engineering, NTUST, Taiwan*
jimrong@mail.sju.edu.tw

Abstract

In the process of rule based knowledge accumulation, due to** various knowledge sources and various expert comments in the knowledge base**, specific knowledge elements in the knowledge base may be in duplicate, in conflict, or inconsistent. The application of wrong information may even lead to wrong decisions. This study proposes the O-A-RV structure for rule-based knowledge and integrates conditional probability, vector matrices, and artificial intelligence to establish the conditional probability knowledge similarity algorithm and develop the knowledge similarity calculation system, which together can quickly and accurately calculate knowledge similarity matrices and determine the relationship among knowledge items. Moreover, according to knowledge relationships, through the inference of value-added treatment such as merging, integration, deletion, innovation and additions, the accuracy of the knowledge itself can be securely ensured and wrong decisions be avoided.

Keywords: Rule-based Knowledge, Conditional Probability, Vector Matrices, Artificial Intelligence, Similarity

1. Introduction

In establishing a rule-based knowledge base, experts are always alert to whether there may exist in the base any logic or structure errors. That is, they insist on the verification of redundancy rules, conflict rules, circularity rules and incompleteness rules. However, in the process of knowledge accumulation, due to different knowledge sources in the knowledge base and different opinions that may be held by different experts, specific information in the knowledge base may be in duplicate, in conflict or inconsistent. This may cause problems such as information unsuitable for use. More importantly, the application of wrong knowledge may lead to wrong decisions.

Knowledge value-added treatment should be done according to relationships among knowledge items, which are determined on the basis of knowledge similarities. So, how to accurately calculate similarity becomes the most basic and necessary task.

2. Literature Review

According to statistics done by McGill, Koll and Noreault in 1979, then-current methods for measuring similarity were continually growing and already numbered more than 60 types including inner product, Dice coefficient, cosine coefficient, Jaccard coefficient, overlap coefficient, etc. [1]. However, the most popular method today remains one based on the distance between the two end-points of two vectors and the angle between the two vectors. Distance in a geometric distance model is usually represented by Euclidean distance; and the angle by dot product [2].

In 1999 and 2001, Zhang put forward the similarity measure method integrating distance and angle. In this method, distance similarity uses an exponential function, with the bottom between 0.7-0.97, and angle similarity a cosine function [4] [5]. In 1997, Frank proposed that the retrieval of different information may require different known similarity measuring methods; for example, after retrieval, the file similarity measure is transformed to the vector of numeric value and then the similarity between two vectors is calculated [3].

3. Measuring Rule-based Knowledge Similarity

3.1 Knowledge Representation in the O-A-RV Format

The syntax of rule-based knowledge representation is IF <antecedent> THEN <consequent>. The antecedent and the consequent, whatever they are, can be represented as a sentence. This paper proposes an improved O-A-RV structure comprising the following four components: Object (O), Attribute (A), Relationship Operator (R) and Linguistic Value (V), as shown in Fig. 1. Some examples of sentences represented by the O-A-RV structure are shown in Table 1.

The attributes of each object in the antecedent and consequent in knowledge, after proper transformation and mapping into numerical types, can be described by the three components in the O-A-RV structure. The RV component of a null component will be Null. The attribute representation of antecedent objects forms an antecedent

vector, while that of consequent objects forms a consequent vector. If the antecedent has n object attributes, it requires 3*n components for representation as described in Eq. (1). The consequent vector is expressed in the way of the antecedent vector. The antecedent and consequent vectors are then combined to form the knowledge vector.

3.2 Transform Mapping of O-A-RV

The possible data types of O-A-V components in a knowledge sentence are nominal, ordinal, or interval and ratio, as shown in Table 2.

Table 2. Data Types of O-A-V Components

Data Type	Operation Property
Nominal	Unable to compare its magnitude. Unable to do arithmetic alculations.
Ordinal	Finite, with order relationships, able to compare its magnitude, unable to do arithmetic alculations.
Interval and ratio	Numerical, able to compare its magnitude and do arithmetic calculations

(1) Transformation of the O-A component

When the data type of the O-A component in a knowledge sentence is nominal, its transformation is 0 or 1 respectively, determined by whether there is an identical character or not, as shown in Table 3.

Table 3. Transformation of O-A Component

Object (O)	Attribute (A)	Mapping Value (O)	Mapping Value(A)
Existing identical character		1	1
No identical character		0	0

(2) Transformation of RV-components

(a)When the data type of V is nominal, its transformation is 0 or 1 according to the nature of the O-A part, determined by whether or not there is an identical character.

(b)When the data type of V is ordinal, it is directly assigned to transform a specific value between 0 and 1 according to Table 4.

Table 4. Direct Assignment of Specific value for Ordinal Data Transformation

Description of V	Transformation of V
small, very small, very slow	VS 0
Small, slow	S 0.25
Medium, common speed	M 0.5
Large, fast	L 0.75
large, very large, very fast	VL 1

(3) If the data type of V is interval and Ratio, it is to keep its original numerical value. Furthermore, if all the Relationship operators (R) of the component are the equal sign, it is to normalize Eq. (2), so as to map the value of V into the range from 0 to 1.

$$V_{norm} = \frac{V - V_{\min}}{V_{\max} - V_{\min}} \quad (2)$$

V_{norm}: the normalized value of V, ranging between 0 and 1
V: the value of V before normalization
V_{max}: the maximal element in component V
V_{min}: the minimal element in component V

(4) If the data type of V is interval and ration, but not all the relationship operators (R) are the equal sign, the transform mapping of RV component is determined by the conditional probability theory.

Conditional probability, $P(B \mid A) = \frac{P(B \cap A)}{P(A)}$ is the probability that the event B occurs, under the condition that the event A occurs too. P(A) is the probability of the event A, $P(A \cap B)$ is the probability that both the event A and event B occur.

Let x be the value of testing cases (T), y the value of knowledge cases (K), then the calculations of the transform mapping values of RV component, under the different situations of T>x or K>y , are shown in Table 5.

Table 5. V_1: Transforming Mapping for x > y

V_1 / Value / V_2	K > y	K = y	K < y
T > x	$\frac{\max - x}{\max - y}$	0	0
T = x	$\frac{1}{\max - y}$	0	0
T < x	$\frac{x - y}{x - \min}$	1	1

Definition R = V_{max}- V_{min}
R:the distributed range of the component V
V_{max}:the maximal element of the component V
V_{min}:the minimal element of the component V

Wherein V_1 is the relationship between x and y, V_2 between K and y, V_3 between T and x, max is the maximal element of the component V with a value R further added, min is the minimal element of the component V with a value R further subtracted.

For the testing cases of $x_1 < T < x_2$ and $y_1 < K < y_2$, which fall within a certain range respectively, wherein V_1 is the relationship among x_1 and y_1, y_2, V_2 is the relationship among x_2 and y_1, y_2, max is the maximal element of the component V with a value R further added, and min is the minimal element of the component V with a value R further subtracted. Due to the pages of paper are limited, so that transformational value of the RV component for other cases can not list in this paper.

3.3 Knowledge Similarity Calculation

After the proper transform mapping and normalization for the knowledge antecedent and consequent, it is able to represent them into the O-A-RV format, and then form the antecedent, consequent and knowledge vectors. If the antecedent and consequent vectors have different dimensions, they are represented into the maximal dimension between them, with those augmented dimensions assigned with zero values. If the antecedent or consequent sentences contain the processing of logic operators like AND or OR, it is described as follows:

IF (a_1 AND a_2) THEN c_1, the antecedent vector is formed by six components, [O_{a1} A_{a1} $R_{a1}V_{a1}$ O_{a2} A_{a2} $R_{a2}V_{a2}$].IF (a_3 OR a_4) THEN c_2, first to divide it into two knowledge operations, IF a_3 THEN c_2 and IF a_4 THEN c_2, with their antecedent vectors represented as[O_{a3} A_{a3} $R_{a3}V_{a3}$] [O_{a4} A_{a4} $R_{a4}V_{a4}$] respectively; after the complete calculations, then to merge them into the original knowledge form IF (a_3 OR a_4) THEN c_2。

For *m* knowledge representations, each with a *n*-dimensional antecedent, and an *l*-dimensional consequent, then the antecedent matrix, consequent matrix and knowledge matrix are described by equations from (3):

$$K=[k_{ij}]_{m*(n+l)}=\begin{bmatrix} k_{11} & k_{12} & \cdots & \cdots & k_{1(l+n)} \\ k_{21} & k_{22} & \cdots & \cdots & k_{2(l+n)} \\ \vdots & \vdots & & & \\ \vdots & \vdots & & & \\ k_{m1} & k_{m2} & \cdots & \cdots & k_{m(l+n)} \end{bmatrix}$$

$$=\left[\begin{array}{cccccc:cccc} \multicolumn{5}{c:}{\text{(Antecedent Matrix)}} & \multicolumn{4}{c}{\text{(Consequent Matrix)}} \\ a_{11} & a_{12} & \cdots & \cdots & a_{1n} & c_{11} & c_{12} & \cdots & \cdots & c_{1l} \\ a_{21} & a_{22} & \cdots & \cdots & a_{2n} & c_{21} & c_{22} & \cdots & \cdots & c_{2l} \\ \vdots & \vdots & & & & \vdots & \vdots & & & \\ \vdots & \vdots & & & & \vdots & \vdots & & & \\ a_{m1} & a_{m2} & \cdots & \cdots & a_{mn} & c_{m1} & c_{m2} & \cdots & \cdots & c_{ml} \end{array}\right] \quad (3)$$

When rule-based knowledge is represented into knowledge vectors, such as the two knowledge vectors, $\vec{k}_i=(k_{i1},k_{i2},k_{i3},\cdots,k_{i(n+l)})$ and $\vec{k}_j=(k_{j1},k_{j2},k_{j3},\cdots;k_{j(n+l)})$, their Euclidean Distance $D(\vec{k}_i,\vec{k}_j)$, Length $\left\|\vec{k}_i\right\|$, Inner Product $<\vec{k}_i,\vec{k}_j>$, are defined in equations from (4) to (6) respectively.

$$D(\vec{k}_i,\vec{k}_j)=\left\|\vec{k}_i-\vec{k}_j\right\|=\sqrt{\sum_{v=1}^{n+l}\left|\vec{k}_{iv}-\vec{k}_{jv}\right|^2} \quad (4)$$

$$\left\|\vec{k}_i\right\|=\sqrt{\sum_{v=1}^{n+l}\left|k_{iv}\right|^2} \quad (5)$$

$$<\vec{k}_i,\vec{k}_j>=\sum_{v=1}^{n+l}k_{iv}k_{jv} \quad (6)$$

The present research proposes a knowledge similarity (KKS), which is simpler than the one in [5] and much easier to understand. It is the multiplicity between the Distance Similarity (DS) and Angle Similarity (AS).The KKS between two knowledge representations with *m* dimensions will take m^2 times of pairwise calculation, which are described in equations from (7) to (10).

$$\mathrm{KKS}_{ij}=DS(\vec{k}_i,\vec{k}_j)\cdot\theta S(\vec{k}_i,\vec{k}_j) \quad (7)$$

$$DS(\vec{k}_i,\vec{k}_j)=1-\left(\frac{D(\vec{k}_i,\vec{k}_j)}{\alpha\cdot\max_{\forall i,j}D(\vec{k}_i,\vec{k}_j)}\right) \quad (8)$$

$$\text{Constant Coefficient } \alpha=\frac{1}{1-DS_{\min}} \quad (9)$$

$$\theta S(\vec{k}_i,\vec{k}_j)=\frac{<\vec{k}_i,\vec{k}_j>}{\left\|\vec{k}_i\right\|\left\|\vec{k}_j\right\|}=\cos\theta \quad (10)$$

The KKS value should be in the range from 0 to 1 since both the Distance Similarity (DS) and Angle Similarity (AS) are ranging from 0 to 1. When the constant coefficient in Eq. (9) is 1, then DS_{min} is 0, DS_{min} is the minimal Distance Similarity, which can be set by users.

The larger the KKS of two knowledge vectors, the more similar are the two knowledge vectors. Since KKS is in the range between 0 and 1, when KKS=1, it means the two knowledge vectors are entirely identical; when KKS=0, it means they are totally different. In the same way, the pairwise Antecedent Similarity (AAS), Consequent Similarity (CCS), Antecedent Consequent Similarity (ACS) can be described by equations from (11) to (13).

$$\mathrm{AAS}_{ij}=DS(\vec{a}_i,\vec{a}_j)\cdot\theta S(\vec{a}_i,\vec{a}_j) \quad (11)$$

$$\mathrm{CCS}_{ij}=DS(\vec{c}_i,\vec{c}_j)\cdot\theta S(\vec{c}_i,\vec{c}_j) \quad (12)$$

$$\mathrm{ACS}_{ij}=DS(\vec{a}_i,\vec{c}_j)\cdot\theta S(\vec{a}_i,\vec{c}_j) \quad (13)$$

3.3.1 Knowledge Similarity Matrix

Equations from (14) to (16) define the knowledge similarity matrix, antecedent similarity matrix and consequent similarity matrix. KSM, ASM and CSM might not be symmetric.

(1) KSM (Knowledge Similarity Matrix):

$$\mathrm{KSM}=[KKS_{ij}]_{m*(n+l)},s.t.\ i=1\ \text{to}\ m,\ j=1\ \text{to}\ n \quad (14)$$

(2) ASM (Antecedent Similarity Matrix):

$$\mathrm{ASM}=[AAS_{ij}]_{m*n},\ s.t.\ \ i=1\ \text{to}\ m,\ j=1\ \text{to}\ n \quad (15)$$

(3) CSM(Consequent Similarity Matrix):

$$\mathrm{CSM}=[CCS_{ij}]_{m*l},\ s.t.\ i=1\ \text{to}\ m,\ j=1\ \text{to}\ l \quad (16)$$

3.3.2 Conditional Probability Knowledge Similarity Algorithm (CPKSA)

This study proposes the O-A-RV structure for rule-based knowledge and integrates conditional probability, vector matrices, and artificial intelligence to establish the conditional probability knowledge similarity algorithm (CPKSA), the architecture of which is shown in

Fig. 2, operated in the following steps:
Input: typical rule-based knowledge
Output: knowledge similarity matrix, antecedent similarity matrix, consequent similarity matrix
Step1: Represent typical rule-based knowledge in the O-A-RV structure
Step2: Transform and map each O-A-RV component into numerical values; represent them as knowledge matrices.
Step3: Calculate pairwise similarity
Step4: Save the matrices of similarity
Step5: Stop

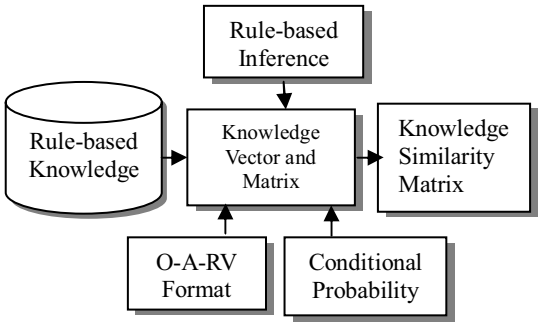


Fig. 2. CPKSA Architecture

3.3.3 An Example of Knowledge Similarity Calculation

Five instances of rule-based knowledge are shown in Table 6. If the RV component is not null, the null value of this component is converted to 0. If the RV component is null, the component value that is not null is converted to 1. The knowledge similarity matrices are given in the Table 7.

4. Conclusions and Future Work

To sum up, the present research reaches the following three conclusions:
1. A new knowledge vector representation for rule-based deterministic knowledge can be proposed on the basis of the O-A-RV structure, which comprises four components: object, attribute, relationship operators and linguistic values.
2. A calculation method for knowledge similarity can be proposed by integrating the distance and angle. Also a conditional probability knowledge similarity algorithm (CPKSA) can be proposed.
3. The knowledge case most similar to the testing case can be quickly retrieved from the knowledge base by applying CPKSA, and used for all types of case-based reasoning (CBR) to help decision making and prediction.

Future work is applying the conclusions to real areas of knowledge reasoning, decision assistance and predicting.

5. References

[1] S. M. Chen, M. S. Yeh and P. Y. Hsiao (1995), "A comparison of similarity measures of fuzzy values," Fuzzy Sets and Systems, Vol.72, Issue: 1, pp.79-89.
[2] B. J. Falkowski (1998), "On certain generalizations of inner product similarity measures," Journal of the American Society for Information Science, Vol. 49, No.9, pp.854-858.
[3] Frank Candocia and Malek Adjouadi (1997), "A Similarity Measure for Stereo Feature Matching," IEEE Transactions on Image Processing, Vol. 6, No. 10, pp.1460–1464
[4] J. Zhang, Edie M. Rasmussen (2001), "Developing a new similarity from two different perspectives," Information Processing and Management, Vol. 37, No. 1, pp. 279-294.
[5] J. Zhang and R.R. Korfhage (1999), "A distance and angle similarity measure," Journal of the American Society for Information Science, Vol.50, No. 9, pp.772-778.

6. Acknowledgements

The authors would like to thank the financial support of the National Science Council of Taiwan government and grant number: NSC-94-2213-E-129-013.

Sentence =	Attribute(A)	Relationship Operator (R)	Linguistic Value (V)	Object (O)

Fig. 1. The components of a sentence

Table 1. Representation of the O-A-RV structure of a sentence

Sentence	O	A	R	V
The temperature of the engine is more than 100 °c.	Engine	temperature	>	100 °c

$$\text{Antecedent Vector} = [O_1 \ A_1 \ R_1V_1 \ O_2 \ A_2 \ R_2V_2 \ O_3 \ A_3 \ R_3V_3 \ldots\ldots O_n \ A_n \ R_nV_n] \quad (1)$$

Table 6. Representation of Knowledge Instances

No.	antecedent			consequent		
K1	=M	>190	=Null	=M	=VL	=Null
K2	=M	=Null	=45	=M	=Null	=VS
K3	=M	=Null	<55	=M	=Null	=S
K4	=F	>180	=Null	=F	=VL	=Null
K5	=F	=Null	>70	=F	=Null	=VL

Table 7. Knowledge Similarity Matrix (KSM)

	K1	K2	K3	K4	K5
K1	1.00	0.11	0.11	0.35	0.00
K2	0.30	1.00	0.84	0.00	0.09
K3	0.28	0.84	1.00	0.00	0.17
K4	0.35	0.00	0.00	1.00	0.11
K5	0.00	0.11	0.19	0.11	1.00

Reverse Engineering of Rule-based Systems

Daniel Wakounig, Abdelhamid Bouchachia
Department of Informatics
University Klagenfurt, Austria
E-mail: {daniel,hamid}@isys.uni-klu.ac.at

Abstract

This paper is concerned with comprehension of legacy rule-based systems (RBS). It shows how static analysis stemming from reverse engineering techniques can be applied in discovering the models underlying RBS. A theoretical background of the approach is presented before a prototype called RETOTS is briefly described. To evaluate the approach, RETOTS is applied on real-world legacy RBS and various aspects are discussed.

1. Introduction

Being one of the most visible knowledge-based systems, RBS have been for long time very attractive in artificial intelligence research and industry. Currently, RBS are a well established technology and applied in various decision making domains like medical diagnosis, mechanical engineering, finance, traffic management, intelligent tutoring, etc. RBS consist of rules of the form: *If condition Then action*, where *condition* is the premise of the rule and can be either simple or complex. A condition may take the form of: $attribute = value$ and the consequence of the rule, *action*, can also take different forms. For instance, an action is an assignment (e.g., assigning a new value/state to a state or variable, assigning an entity to a class).

Despite their wide deployment in many operational environments, RBS still lack analysis and verification mechanisms to enable automatic and easy update. Basically, the challenge with these systems is related to *quality* and *maintenance* issues. The quality aspect has been studied by many authors in the past [11, 12]. It is mainly concerned with knowledge consistency and conflict analysis relying on various verification and validation (V&V) methods such as inspection and testing [5, 13], be it static or dynamic testing. Note that dynamic testing is usually preceded by static analysis that aims at removing anomalies.

Validation is about repairing and removing contradictions. On the other hand, maintenance entails the activities of knowledge optimization, and evolution [8]. The task of optimization seeks to make knowledge compact. This is usually done by alleviating all types of redundancy and rendering the knowledge base transparent. The second aspect, that is evolution, is about future update of the readily available rules. This operation has to be done carefully to avoid jeopardizing the existing rule base. RBS Evolution is a key, but unfortunately not well studied, that ensures a continuous growth of consistent and transparent knowledge. Furthermore, it is worth stressing that incremental update of RBs is an important aspect since RBS might be equipped with incomplete and insufficient knowledge at a given time in order to be capable of handling all required reasoning situations. However, when adding or removing rules from an operational system, special attention must be payed to rule inconsistencies and conflicts that may result from the update operation. Such phenomena must be resolved using appropriate analysis tools. Here, analysis can be seen as a reverse engineering (RE) process. Making such tools available will help dealing also with legacy systems.

As a general definition, RE is the process of analyzing an existing system to identify its components and their interrelationships and create an abstract representation of the system. RE is usually undertaken in order to redesign the (legacy) system for better maintainability. In the current work, we are particularly interested in static analysis, as a RE task, to comprehend an undocumented RBS. RE of RBS is a retrospective analysis that aims at discovering the hidden and undocumented structure of the rule base, the relationship between rules, the type of objects (or variables) manipulated by the rules, and the possible combinations of objects with rule conditions and their ramifications.

The approach investigated here is a typical static analysis approach that aims at conducting RE of an operational legacy RBS. The main aims are: **(1)** Understanding the structure of rules, **(2)** Understanding the dependencies between rules over types/objects, **(3)** Locating local contexts in the rule base, **(4)** Visualizing rules for special purposes like update and track of change, and **(5)** Equipping the existing system with an interface to enable interrogation and

analysis of the rule base content.

The class of RBS we are investigating is a particular one. Here, rule conditions and conclusions report on types (instances) rather than on strict equality (feature=value). More details on such systems will follow.

The rest of the paper is organized as follows. Section 2 describes how RE techniques are applied for RBS. Section 3 introduces a theoretical basis of the analysis approach before the processing analysis steps are explained in Sec. 4. The application of the approach on real-world legacy system is discussed in Sec. 5. Finally, Sec. 6 outlines some future work and concludes the paper.

2. Reverse Engineering Techniques for RBS

One of the basic techniques of RE is static analysis which aims at mining the source code to obtain the knowledge representing the abstractions of the software structure. The knowledge is language dependent and concerns the usage of variables, types, methods, classes and represented in the form of entities and relationships before its storage in a queriable and visualizable database.

To the best of our knowledge, analysis of legacy RBS has not been the subject of any research work from the perspective of RE. It is, however, indirectly discussed in the literature related to rule-based testing and verification. Usually analysis of software products, be it from a forward engineering perspective or RE perspective, relies on data and control flow graphs. Such techniques need adaptation to fit the characteristics of RBS. The basic idea is how rules can be represented as a graph reflecting rule dependencies. This idea has been discussed under the name of physical rule flow graph (PRFG) [12], where the nodes represents rules and edges represent the relationship "is-executed-after". Thus, PRFG represents the order of rule executions which is different from the logical order. However, when analyzing of a RB, we do not have an execution order, hence only a static analysis is possible. This latter consists of a logical rule flow graph (LRFG) which shows the rules and their logical combinations. The nodes in LRFG represent the rules and the edges represent the relationship "contributes-to-fire". Let us show this using an example (viz. 1):

$$\begin{aligned} R_1 &: \text{if a then assert b} \\ R_2 &: \text{if b then assert c} \\ R_3 &: \text{if b and c then assert d} \end{aligned}$$

R_1 causes R_2 to fire and both R_1 and R_2 contribute to fire R_3. Explicitly said, a rule R_i contributes to fire a rule R_j if the right hand side (RHS) conclusions of R_i matches the LHS conditions of R_j. This idea was extended by several authors in different systems [10, 5, 13, 12]. It is also worthwhile to point out that such techniques have been used for testing and redundancy elimination as in [11, 9]. However, in the aforementioned research work, the rule conditions and the conclusions report on a precise and specified feature value (i.e. strict equality between an instance feature and a value). The present work deals with a very particular type of rules. They have the generic form:

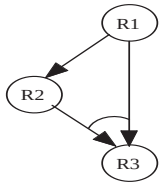


Figure 1. Graphical representation of rule flow example

$$R_i: \quad \text{if } \psi(O_1,T_1)\cdots\psi(O_m,T_m) \quad \text{then} \quad \phi(O_1,T_1)\cdots\phi(O_n,T_n) \tag{1}$$

The condition and conclusion parts involve operations on objects. These objects, O_i, are instances of types, denoted as types, T_j, in the sense of object-oriented paradigm. The functions ψ and ϕ are existensional and creational operations respectively, where the former is a Boolean function that takes two forms: existence of an object or its absence and is formally defined as:

$$\psi(O_i,T_j) \equiv \begin{cases} \exists\, O_i^j & \text{if the object } O_i \text{ of type } T_j \\ & \text{exists in the working memory} \\ \neg\exists\, O_i^j & \text{if } O_i \text{ of type } T_j \text{ does not exist} \\ & \text{in the working memory} \end{cases} \tag{2}$$

The latter function allows to change the state of the working memory by adding, removing, or changing objects. It can be defined as follows:

$$\phi(O_i,T_j) \equiv \begin{cases} \text{Create an object } O_i \text{ of type } T_j \\ \text{Remove the object } O_i \text{ of type } T_j \\ \text{Change the object } O_i \text{ of type } T_j \end{cases} \tag{3}$$

The generic actions create, remove and change are modeling primitives which are not implemented using the same wording as will be explained below. Since the conditions and conclusions of rules involve typed objects, such RBS are called **typed** RBS. Formally, we define typed RBS as:

Definition 1 (Typed RBS) *Any RBS based on rules of the form (1) is called typed system.*

The objective of this research is to reverse engineer typed RBS where rules have the form (1) and which are implemented in the following way:

$$R_i: \quad \text{If not exists Sensor} \quad \text{Then assert new HTSensor()} \tag{4}$$

This rule checks if an object $Sonsor$ is alive, an object of (sub)type $HTSensor$ is then created. It is clear now that the available techniques dealing with analysis of rule bases systems are not appropriate to the legacy RBS we are reverse-engineering. In the rest of this paper, the analysis of this legacy system is presented.

3. Rule Dependencies

Being the main component of the system to be reverse-engineered, rules dependencies aim at showing how rules interact between each other over the facts in the working memory. Facts are simply typed objects. To discover the dependencies, we adopt a static analysis technique that is the logical rule flow graph (LRFG) but adapted to our typed RBS. Rule dependencies can be categorized into two types:`Direct dependencies` and `Transitive dependencies`. These can be formally defined as follows:

Definition 2 (Direct dependency)
Let $\mathcal{R}=\{R_1, R_2, \ldots, R_N\}$ be the set of rules in the system's rule base. A rule $R_i \in \mathcal{R}$ is said `directly dependent` *on $R_k \in \mathcal{R}$, denoted as $dd(R_i, R_k)$, if and only if $\mathcal{T}(P_i) \cap \mathcal{T}(C_k) \neq \emptyset$, where P_i and C_i are the premise (condition) and the conclusion part of the rule R_i respectively, and $\mathcal{T}(.)$ is a function that returns the types of the objects referenced by its parameter. Furthermore, a direct dependency takes one of the following six forms:*

1. *$dd(R_i, R_k)$ is labeled as **creates-required** $\Longleftrightarrow$ R_k creates an instance ($\phi(O_i, T_j)$), O_i, of a type, T_j, required by R_i (i.e., $\psi(O_i, T_j) = \exists\, O_i^j$).*
2. *$dd(R_i, R_k)$ is labeled as **creates-not-allowed** $\Longleftrightarrow$ R_k creates an instance, O_i, of a type, T_j, which is not allowed (i.e., $\psi(O_i, T_j) = \neg\exists\, O_i^j$) by R_i.*
3. *$dd(R_i, R_k)$ is labeled as **modifies-required** $\Longleftrightarrow$ R_k modifies an instance of a type required by R_i.*
4. *$dd(R_i, R_k)$ is labeled as **modifies-not-allowed** $\Longleftrightarrow$ R_k modifies an instance of a type not allowed by R_i.*
5. *$dd(R_i, R_k)$ is labeled as **deletes-required** $\Longleftrightarrow$ R_k deletes an instance of a type required by R_i.*
6. *$dd(R_i, R_k)$ is labeled as **deletes-not-allowed** $\Longleftrightarrow$ R_k deletes an instance of a type not allowed by R_i.*

Definition 3 (Transitive dependency)
A rule $R_i \in \mathcal{R}$ is said `transitively dependent` *on a rule R_z, denoted as $td(R_i, R_z)$ if and only if there exists a sequence of direct dependencies such that $[dd(R_i, R_j), dd(R_j, R_k), \ldots, dd(R_x, R_y), dd(R_y, R_z)]$ among a subset of rules $\{R_j, \ldots, R_y\} \subset \mathcal{R}$.*

Now, if the types of the objects belong to a type hierarchy. The nodes in the hierarchy tree represent types which are related by typing relationship (subtype, supertype). This leads to consider another class of rule dependencies, that is `type dependencies`. Two rules are type-dependent if they involve different types of the same hierarchy. In Example 1, the rule `CreateSensor` creates an instance of type `HTSensor`, and rule `DeleteSensor` requires an instance of type `Sensor` to be fired. `DeleteSensor` is type-dependent on `CreateSensor`, since `HTSensor` is a specialization of `Sensor`.

Example 1 (Type-dependency)

```
rule CreateSensor
  when
    not exists s:Sensor;
  then
    create HTSensor();
end rule;
rule DeleteSensor
  when
    exists s:Sensor
  then
    delete s;
end rule;
```

More formally, type dependency is defined as follows:

Definition 4 (Type order)
Let $\mathrm{T} = \{T_1, T_2, \ldots, T_M\}$ denote the set of types referenced by R_i. Types belong to the same hierarchy iff there exists an order relation $\leq_T$ that is defined on the set T such that: $T_i \leq_T T_j$ if and only if T_i is a subtype of T_j (i.e., T_j is a supertype of T_i)

Definition 5 (Type dependency)
Let $\mathrm{T} = \{T_1, T_2, \ldots, T_M\}$ denote the set of types referenced by R_i and an order relation $\leq_T$ defined on the set T. A rule $R_i \in \mathcal{R}$ is said `type-dependent` *on $R_k \in \mathcal{R}$, denoted as $td(R_i, R_k)$, if and only if there exists a type $T_j \in \mathcal{T}(P_i)$ and $T_m \in \mathcal{T}(C_k)$, where P_i and C_k are the premise and the conclusion part of the rules R_i and R_k respectively, such that: $T_j \leq_T T_m$ or $T_m \leq_T T_j$.*

4. Processing steps

To perform a static analysis of the legacy RBS involving rules of the form described in (1), we suggest an approach consisting of two steps:

- Transformation: aims at extracting knowledge from RBS and storing it in a database.
- Analysis: aims at discovering the dependencies between rules.

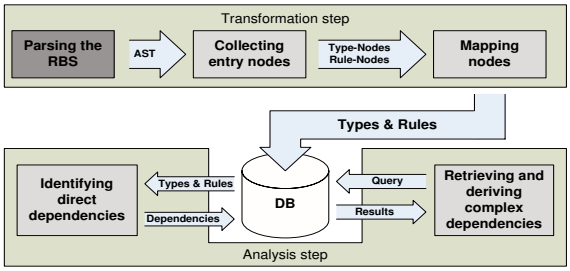


Figure 2. Processing steps

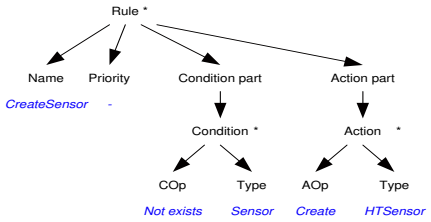


Figure 3. Mapping a RBS onto an AST

Each of these steps is realized by a module in a RE tool as portrayed in Fig. 2. The tool is called RETOTS standing for **R**everse **E**ngineering **TO**ol for **T**yped rule-based **S**ystems.

4.1. Transformation

RETOTS admits as input a rule base of a legacy system which is subject to RE. The first task executed by RETOTS is the transformation of the legacy code into structured knowledge that can be mined and analyzed. To achieve this purpose, RETOTS usually uses application-specific knowledge to decode the rule base. The transformation step consists of three stages: parsing, information collection, and mapping. Each of these steps is realized by a corresponding submodule as describe in the following.

4.1.1 Parser

The first task that is ever triggered by RETOTS during its working cycle is parsing. It consists of transforming the code into an abstract syntactic tree (AST). This tree is dependent on the RBS grammar (i.e., it is entirely application specific). Figure 3 shows the relevant nodes of such a tree.

4.1.2 Collector

Since the AST produced by the parser may be very large and contains irrelevant nodes, the task of the collector consists of traversing AST in order to select the most meaningful nodes. These nodes are associated with: `Rule definition` and `Type definition`.

The output of this step is two sets: the first consists of rule entry nodes and the second consists of types associated with the rule entry nodes. To achieve an application-independent Collector, knowing that AST is application specific, RETOTS is equipped with a *configurable* Collector so that the rule and type entry nodes are specified by the maintainer. In our brief example in Fig. 3 `Rule`-nodes are collected since they are stated as Rule-entry nodes of the legacy system.

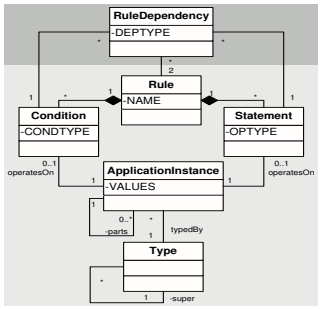


Figure 4. Generic representation

4.1.3 Mapper

Once the relevant nodes are collected, the mapper module transforms them into a generic representation that is stored in a database. This representation is shown in Fig. 4 (light gray part). The darker part is designed for rule dependency storage. Note that the mapper is partly application-specific since it needs some knowledge on how to interpret the nodes found by the collector and how to transform them.

4.2. Analysis

The second module of RETOTS is the analyzer which is responsible for the identification of the dependencies among the mapped results obtained and stored in a database by the transformation module. The analysis module consist of two sub modules: the *Direct Dependencies Analyzer* and the *Query Engine*. The former is responsible for finding the direct dependencies (strict match of types) and type-dependencies (semantic match based on typing hierarchy) using Definitions 2, and 5. The search engine is responsible for two tasks: (i) finding the transitive dependencies relying on Definition 3 and (ii) interpreting the user's queries. This latter task is initiated once the user interactively and via a system interface formulates his/her query. The user is allowed to express different types of queries on the database.

5. Evaluation: RETOTS at Work

RETOTS is being experimented on rule based model for a simulator called **SESAM** (standing for **S**oftware **E**ngineering **S**imulation by **A**nimated **M**odels) [2]. SESAM

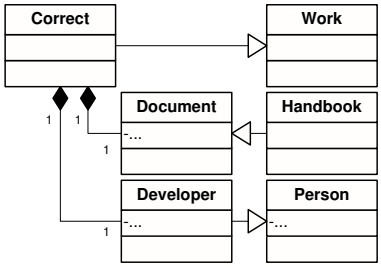


Figure 5. Excerpt of the type hierarchy

was developed at the University of Stuttgart (Germany). It is open source (LGPL) and available to download for free. Models for SESAM are written in the model description language BASE2 [6] which is similar to ILOG [1]. At our University, SESAM has been integrated in a project called AMEISE (A Media Initiative for Software Engineering) dedicated to management of software projects [3]. Actually we examined the rule base (*QA-Model*) which is used to simulate a software project [4]. We extended the SESAM simulator by wrapping and using it as a core simulation engine. Due to some extension requirements, we needed to adapt the actual QA-model. But, this turned out to be a difficult task due to lack of maintainability and tool support, hence this work was initiated. To give an insight into the BASE2 language, consider Example 2 where a rule and a type are displayed. Figure 5 shows an excerpt of the type hierarchy. Note that the total number of types is 267.

Example 2 (BASE2 type and rule example)

```
TYPE CORRECT conforming WORK is
  WHO : DEVELOPER;
  WHAT : DOCUMENT;
END TYPE;
RULE CORRECT_HANDBOOK CONDITIONS
  EXISTS dev : DEVELOPER;
  EXISTS hb : HANDBOOK;
  NOT EXISTS(CORRECT(dev,hb));
BEGIN
  CREATE CORRECT WITH
    WHO := dev;
    WHAT := hb;
  END CREATE;
END RULE;
```

The rule-base of QA-model is transformed into the generic representation and stored in a database for further processing. One can then submit certain queries to RETOTS to explore different aspects of the rule base. In the following, we will analyze the rule base considering the aspects: **(1)** Complexity of the rule base, **(2)** Dependency-based rule clustering, and **(3)** Change metrics.

As to the complexity of the rule base, the first metric one would be interested in is the number of instances per dependency category defined in Def. 2. Using RETOTS,

Category of Dependency	Number of Dependencies
Creates-required	40362
Modifies-required	91160
Deletes-required	38683
Creates-not-allowed	1966
Modifies-not-allowed	0
Delete-not-allowed	2140
Total	174311

Table 1. Dependencies found

# Rules	Types Creat.	Types Modif.	Types Del.
0	20	191	170
1	224	14	69
> 1	23	62	29

Table 2. Relationship Rule-Type

we have found the numbers shown in Tab. 1. These show how the rule base is complex. The total number of dependencies is 174311. The ”modifies-required” dependency type is the most frequent. Unexpectedly, the ”modifies-not-allowed” dependency type does never occur in the QA-model. Furthermore, to get an insight into the relationship between the number of rules and the number of types involved in the rules' conclusions, RETOTS outputs the numbers shown in Tab. 2. It is worth noticing that a significant number of types are not handled by any rule; e.g., there are 20 types (among the 267 available types) which are never used (i.e., by any rule) to create new instances. The reason for that can be either one of the following alternatives:(1)instances of such types are created during the initialization of the system independently of the rules, (2) such types are abstract types, hence instances are created by specialized (concrete) types, (3) such types are never instantiated and therefore never used, or (4) such types are never instantiated but accidentally used (consistency problem). Furthermore, there are 191 types whose instances get never modified by rules. Possible reasons for that are either instances of such types may be modified via their supertype or instances of such types are required (e.g. used as a condition flag) by some rules but never modified.

Finally, there are 170 types whose instances are never deleted by rules. This usually happens when these instances are to be returned by the system as results. The second aspect to be examined using RETOTS is related to dependency clustering. The basic idea here is to build clusters of inter-rules dependencies using the fuzzy C-Means (FCM) clustering algorithm [7]. The input to this algorithm is a set of feature vectors, where each feature represents a direct dependency category, hence the length (number of features) is the number of direct dependency categories. A vector, v, is

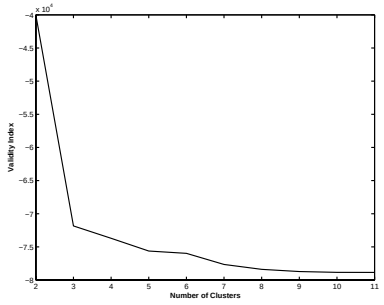


Figure 6. Dependency clustering

a realization of dependencies between two particular rules;

$$v_i(R_j, R_k) = \begin{cases} 1 & \text{if rules } R_j \text{ and } R_k \text{ are related to each other via a dependency of category } i \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

The aim is to gather vectors (pairs of rules) having a similar degree of coupling. In other words, dependency vectors belonging to the same cluster are of the same complexity. Understanding and maintaining rules involved in vectors of the same cluster require similar effort. One can then predict the maintenance costs and risks quite efficiently. However, fuzzy C-means requires, as input, the number of clusters to be known. This is not only the case of FCM but most of the clustering algorithms. To solve this problem we rely on a validity index that combines the within variance and the between-variance of clusters [7]. FCM is run on a range of values and the index is computed. The minimum value of this index corresponds to the optimal number of clusters (well separated and compact). Figure 6 shows the evolution of the validity index. Indeed, the optimal number of clusters would be in the set $\{8, 9, 10, 11\}$ since the curve flattens starting from cluster 8. Taking the time complexity of FCM into account, 8 clusters would be a completely acceptable value. This experiment allows to draw a correspondence between clusters and a set of complexity levels. These levels are simply coupling degrees. Higher coupled rules will be assigned to the same cluster and therefore a similar maintenance effort is expected. The last aspect we have examined is the quantification of the gain obtained over the application of RETORTS on the typed RBS. The gain is simply the number of rules on average to be discarded during the update of the rule base. It reflects the proportion of the rules to be examined. We have formulated this gain as follows:

$$gain = \frac{1}{|\mathcal{R}|} \sum_{i=1}^{|\mathcal{R}|} (|\mathcal{R}| - |\overline{R_i}|) \quad (6)$$

where $\mathcal{R}$ is the entire set of rules and $\overline{R_i}$ is the set of rules which are dependent on the rule R_i. The obtained gain value is 528,838. Recalling that the number of rules (626), on average 84.48% of the rule base content is not examined during the maintenance of a given rule.

6. Conclusion

In this paper, a maintenance approach of a legacy RBS is presented. Based on typed dependency analysis, a prototype is developed and evaluated on real working legacy system showing very interesting outcome. As a future work, further analysis aspects will be investigated especially with respect to rule and dependency clustering. Furthermore, transitive dependencies have to be investigated more thouroughly.

References

[1] Ilog website. http://www.ilog.com, ILOG Inc.

[2] Sesam website. http://www.iste.uni-stuttgart.de/se/research/sesam/index_e.html, Dep. of Software Engineering, Univaersity Stuttgart.

[3] Ameise website. http://ameise.uni-klu.ac.at, Dep. of Software Engineering, University Klagenfurt.

[4] Qa-model documentation. http://www.iste.uni-stuttgart.de/se/research/sesam/models/index_e.html, Dep. of Software Engineering, University Stuttgart.

[5] J. Kiper. Structural testing of rule-based expert systems. *ACM Transactions on Software Engineering and Methodology*, 1(2):168–187, 1992.

[6] R. Melchisedech and R. Reiing. Sprachbeschreibung fr base-2. internal report, Dep. of Software Engineering, University Stuttgart, 2001.

[7] J. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Plenum, New York, 1981.

[8] A. Bouchachia. Maintaining knowledge bases via incremental rule learning. In *Proc. of the Int. Workshop on Soft Computing for Information Mining(German Conf. on AI)*, pages 51–63, Ulm, Germany, 2004.

[9] R. Stachowitz, J. Combs, and C. Chang. Validation of knowledge-based systems. In *Proc. of the* 2^{nd} *AIAA-NASA-USAF Symposium on Automation, Robotics, and Advanced Computing for the National Space Program*, pages 1–9, Arlington, VA, 1987.

[10] Z. Chang, R. Stachowitz, and J. Combs. Validation of non-monotonic knowledge-based systems. In *Proc. of the* 2^{nd} *Inter. Conf. on Tools for Artificial Intelligence*, pages 776–782, 1990.

[11] T. Nguyen, W. Perkins, T. Laffey, and D. Pecora. Checking an expert system's knowledge base for consistency and completeness. In *Proc. of the* 9^{th} *Inter. jnt. Conf. on Artificial intelligence*, volume 1, pages 375–378, 1985.

[12] A. Preece, R. Shinghal, and A. Batarekh. Principles and practice in verifying rule-based systems. *Knowledge Engineering Review*, 7(2):115–141, 1992.

[13] J. Rushby and J. Crow. Evaluation of an expert system for fault detection, isolation, and recovery in the manned maneuvering unit. Nasa Contractor Report CR-187466, SRI Intenational, Menlo Park, CA.

A New Method of Value-Added Treatment Inference for Rule-based Uncertainty Knowledge

Chin-Jung Huang [1], Min-Yuan Cheng [2]

[1]*Department of Mechanical and Computer-Aided Engineering, St. John's University, Taiwan*
[2] *Department of Construction Engineering, NTUST, Taiwan*
jimrong@mail.sju.edu.tw

Abstract

In the process of knowledge accumulation, varied sources and expert comments lead to similarity, conflict or inconsistent data in knowledge base. Along with change of time and space, emerging of new technology, new law, new method and new evidence, the knowledge may not be suitable, the integral connotation and relation have not been shown, and wrong knowledge leads to wrong decision. And, certainty knowledge is one special case of uncertainty knowledge when the certainty factor (cf) equals to 1. Therefore, this study proposes the conditional probability knowledge similarity algorithm, to rapidly acquire the knowledge similarity matrix and determine the relationship of knowledge, which will be the source of increasing value-added treatment. This study also proposes a Reliable Factor (RF) to characterize knowledge with conflict, overlap or inconsistent data size, a certainty factor (cf) to denote confidence of knowledge, a Decision Factor (DF) to express the index of knowledge assistant decision. Based on knowledge relationship, a rule-based uncertainty knowledge Value-Added treatment Inference algorithm (RUKVATIA) is established to perform Value-Added treatment such as merge, integration, delete, innovate, and add, so that the relation mapping of knowledge and knowledge assistant decision index can be obvious, avoiding wrong decision effectively

Keywords: conditional probability, artificial intelligence, similarity, reliable factor, decision factor

1. Introduction

In the process of knowledge accumulation, varied sources and expert comments, and change of time and space, emerging of new technology, new law, new method and new evidence, lead to conflict, overlapping or inconsistent data in knowledge base. With change of time and space, emerging of new technology, new law, new method and new evidence, the knowledge may not be suitable, and the most important point is, wrong knowledge leads to wrong decision. Knowledge accuracy is short of verification reasoning, related knowledge with similar connotations are short of merging or integration reasoning, conflicted or overlapped knowledge are short of innovating or deleting reasoning. Without these Value-Added reasoning treatments, not only the knowledge accuracy, knowledge conflict or overlapping, or data size inconsistency and knowledge consistency remain to be verified, but direct citing of knowledge from the base becomes both risky and unable to assist decision. Furthermore, certainty knowledge is just one special case of uncertainty knowledge when the certainty factor (cf) equals to 1. So, it is necessary to build a type of Value-Added treatment reasoning for rule-based uncertainty knowledge in order to increase the added values of all rule-based uncertainty knowledge so that the knowledge management gets more effective.

According to the Conditional Probability Knowledge Similarity Algorithm (CPKSA) [1], so as to calculate the relative similarity of antecedent, consequent and rule-based knowledge, and further determine the relationship of antecedent, consequent and knowledge, and carry out Value-Added treatment reasoning such as merging, integration, innovation, adding or deleting. This study also put forward a Reliable Factor theory to characterize knowledge with conflict, overlap or inconsistent data size and embody the reliable extent of knowledge; this study uses the Certainty Factor to demonstrate certainty extent of knowledge; this study invents a Decision Factor to indicate the objective of assistant decision; this study sets up the Rule-based Uncertainty Knowledge Value-Added Treatment Inference Algorithm (RUKVATIA) to realize knowledge Value-Added treatment, so that knowledge of various disciplines become more integral, and related knowledge mapping and knowledge-assistant decision objective can be realized, and hence wrong decision can be avoided.

2. Literature Review

In 2005, Chin-Jung Huang integrated Conditional

Probability, Vector Matrices and Artificial Intelligence, to establish the Conditional Probability Knowledge Similarity Algorithm (CPKSA) can rapidly obtain knowledge similarity matrix. [1].

In 1996, Kunhuang Huarng proposed the object Knowledge Interchange Format (KIF) of knowledge reuse, in order to build the object-oriented expert system [4]. In 2001, M. Lynne Markus proposed the theory of knowledge reuse on the basis of 4 kinds of knowledge reusers and the purpose of knowledge reuse, to analyze factors and situations of successful knowledge reuse [6]. Thomas Housel used Business Process Reengineering (BPR), focusing on cost down, to analyze Knowledge Value Added (KVA), and meanwhile proposed 3 KVA methodologies including the Learning Time Approach, Process Description Approach and Binary Quary Method.

In 2001, Ron Hyson utilized perspective and efficacy to discuss 36 kinds of valuemetric enterprise models and to analyze the value added of increasing enterprise intelligent capital [7]. In 2002, Gopika Kannan pointed out, on the basis of Infotech survey, that organizational culture of intellectual enterprise, and knowledge management support systems and processes, are factors of increasing knowledge value added of enterprise human capital [3].

Key points of the above Knowledge Value Added (KVA) treatments are to improve efficiency and efficacy of spreading, disseminating, sharing and using knowledge, i.e., to raise frequency of knowledge use. This study directly discusses how to increase value added of knowledge itself, such as more integration, related mapping of knowledge, countermeasures of knowledge conflict and inconsistent data size, and knowledge accuracy.

In 1975, Shortliffe and Buchanan applied the theory of certainty factor (cf) in a medical expert system called MYCIN [8], where cf denotes how certain experts are about the knowledge; in 1994, Durkin proposed the cf combination algorithm [2].

3. Rule-based Knowledge Value-Added Treatment Inference Algorithm

3.1 Knowledge Relationship

The value-added treatment should target at those specially related knowledge sets, including those with same antecedents, consequents, or those with either a same antecedent or consequent, or the case that one consequent acts as another antecedent, or the one with contradictive or independent antecedents and consequents. According to the above mentioned knowledge similarity matrix, antecedent similarity matrix, consequent similarity matrix, antecedent consequent similarity matrix, it is able to specify the relationship of two knowledge instances. The present research proposes six types of interrelations of knowledge representations, including one-to-many, many-to-one, many–to-many, cause and effect, contradiction, independence, as shown in Table 1.

3.2 Significance of knowledge value-added treatment

The implication of value-added treatment is reflected by the overall expression of knowledge and the added value of its application, which subsumes the following essential implications:

(1)Knowledge Expression Aspect: make the knowledge expression more integrative and representative.

(2)Knowledge Relation Aspect: clearly present the relationship and the transform mapping of two knowledge instances.

(3)Knowledge Application Aspect: If the Reliable Factors of knowledge (RF) emerge obviously, then wrong decisions can be avoided.

The different value-added treatments for those specially related knowledge sets, without considering the input of new knowledge, are stated in the following:

(1)Merge: to merge those knowledge sets with same antecedents (one-to-many correspondence), or same consequents (many-to-one correspondence), or either same antecedents or consequents (many-to-many correspondence), provided that the antecedents and consequents are not contradictive, or inconsistent in value, so as to reveal more integrity and representability of knowledge.

(2)Integration: in order to present more integrity and representability of knowledge, integrate the knowledge sets associated in a cause and effect way, that is, in which one knowledge consequent acts as another knowledge antecedent (Z-type correspondence).

(3)Innovation: to adopt the innovating process for the knowledge sets with same antecedents (one-to-many correspondence), same consequents (many-to-one correspondence), either the same antecedents or consequents (many-to-many correspondence), or those with contradictive relationship, provided that the antecedents or consequents are contradictive or inconsistent in value. This will obviously present the reliable factors of the knowledge itself and to avoid wrong decisions, and to expand the coverage of knowledge application (both the width and depth).

(4)Deletion: It is able to delete the knowledge with identical meaning in the original knowledge base, so as to reduce the storing size and search time.

(5)Search: It is able to search the most similar historical knowledge (case) to the new knowledge (case) from the knowledge base, so as to complement with the KBR or CBR for decision making.

As new knowledge arises due to change of time and space, new technique, new regulation, new approach and new evidence, it may not conform with the knowledge in original knowledge base, or the knowledge in original knowledge base become unsuitable any longer, taking new knowledge into consideration, appending and deleting treatments have to be done when necessary:

(6)Append: run add operation to ensure knowledge

correctness, in regard to new knowledge derived due to new technique, new regulation, new approach, new evidence.

3.3 Theory of Reliable Factor (RF)

According to human concept of solving disputes and conflicts, the consensus (intersection) method is adopted, otherwise, voting is carried out, and the minority follows the majority. In this paper, Reliable Factor (RF) theory is put forward to deal with conflicts, or get the intersection consensus of conflicted people. For example, three experts have three different solutions to one problem, as shown in Fig. 1, in the end, the solution shared by three experts, or intersection of them, is chosen, so implementing plan in this way can gain support and approval of three experts, that is, the reliable factor is the highest.

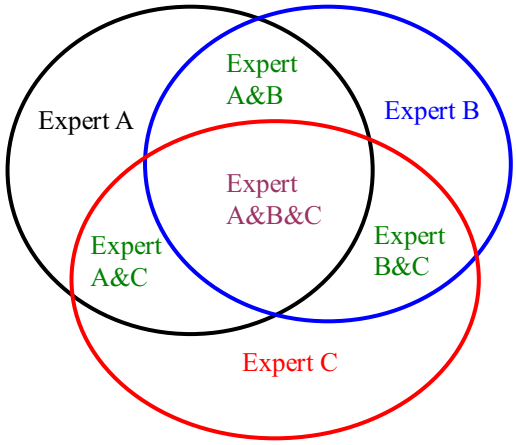


Fig. 1. Human concept of dealing with dispute or conflict

As for conflicted, overlapped or different data size knowledge, the Reliable Factor (RF) is proposed to correctly show the reliability of this knowledge.

Definition: RF of knowledge with same antecedent by equ. (1). And RF of knowledge with same consequent by equ. (2). Summary of RF reasoning result of examples is shown in Table 2.

4. Rule-based uncertainty knowledge Value-Added treatment reasoning

4.1 Theory of calculating certainty factor of uncertainty knowledge

(1)Net certainty factor (cf) of knowledge with single antecedent [5]:

Knowledge: IF < antecedent (a) > THEN <consequent (c) > {cf_a, cf_{rule}}In short, [a, c] {cf}

$$cf = cf_a * cf_{rule} \quad (3)$$

where cf_a: certainty factor of knowledge with single antecedent, indicating the probability of this antecedent, which is between 0 and 1.

cf_{rule}: certainty factor of rule reasoning, indicating the probability of this rule reasoning, which is between 0 and 1.

cf: net certainty factor (cf) of knowledge integrating single antecedent and rule reasoning, indicating the probability of this rule reasoning, which is defined between 0 and 1, as shown in Table 3 [5].

Table 3. Uncertain terms and their interpretation

Term	Certainty factor
Unknown	0 to 0.2
Maybe	0.4
Probably	0.6
Almost certainly	0.8
Definitely	1.0

(2)Net certainty factor of knowledge whose antecedent has logic operator AND [4]:

Knowledge: IF < antecedent a_1 > AND < antecedent a_2 > AND …AND < antecedent a_n >

THEN < consequent c > {cfa_1, cfa_2,…., cfa_n, cf_{rule}}

In short, [$a_1 \cap a_2 \cap \ldots \cap a_n$, c] {cf}

$$cf = \min[cf_{a_1}, cf_{a_2}, \ldots, cf_{a_n}] * cf_{rule} \quad (4)$$

(3)Net certainty factor of knowledge whose antecedent has logic operator OR [4]:

Knowledge: IF < antecedent a_1 > OR < antecedent a_2 > OR …OR < antecedent a_n >

THEN < consequent c > {cf_{a_1}, cf_{a_2},…., cf_{a_n}, cf_{rule}}

In short, [$a_1 \cup a_2 \cup \ldots \cup a_n$, c] {cf}

$$cf = \max[cf_{a_1}, cf_{a_2}, \ldots, cf_{a_n}] * cf_{rule} \quad (5)$$

(4)Reasoning of net certainty factor of knowledge with convergent relationship (multiple to one):

Knowledge 1: [a_1, c] {cf_1}

Knowledge 2: [a_2, c] {cf_2}

Reasoning knowledge 3: IF (a_1 OR a_2) THEN c {cf_3}, in short, [$a_1 \cup a_2$, c] {cf}

The net certainty factor of knowledge is shown in equation (6) [2]

$$cf = cf_{1,2} = cf(cf_1, cf_2) = cf_1 + cf_2 - cf_1 * cf_2 \quad (6)$$

The universal equation is $cf_{i,j} = cf_i + cf_j - cf_i * cf_j$ ($i \neq j$), and $cf_{i,j} = cf_{j,i}$

$$\text{Pro rata } cf_{i,j,k} = cf(cf_i, cf_j, cf_k) = cf_i + cf_{jk} - cf_i * cf_{jk} \quad (7)$$

4.2 Reasoning of net certainty factor of uncertainty knowledge with various relationships

(1) Reasoning of certainty factor of knowledge set with divergent relationship (one to multiple):

Knowledge 1: [a_1, c_1] {cf_1}

Knowledge 2: [a_1, c_2] {cf_2}

To infer the following knowledge:

Knowledge 3: [a_1, $c_1 \cup c_2$] {cf_3}

Knowledge 4:[a_1, $c_1 \cap c_2$] {cf_4}

Then certainty factors of knowledge 3 and knowledge 4,

cf_3 and cf_4, are obtained as in equation (8) and (9) respectively.

Knowledge 3:$cf_3=cf\,[a_1, c_1 \cup c_2] = \max[cf_1, cf_2]$ (8)

Knowledge 4:$cf_4=cf[a_1, c_1 \cap c_2]=\min[cf_1, cf_2]$ (9)

(2) Reasoning of certainty factor of knowledge set with cross relationship (multiple to multiple) (meanwhile with divergence and convergence mapping):

Knowledge 5: $[a_l, c_1]$ $\{cf_5\}$

Knowledge 6: $[a_1, c_2]$ $\{cf_6\}$

Knowledge 7: $[a_1, c_3]$ $\{cf_7\}$

Knowledge 8: $[a_2, c_1]$ $\{cf_8\}$

Knowledge 9: $[a_2, c_3]$ $\{cf_9\}$

To reason the following knowledge:

Knowledge 10: $[a_1, c_1 \cap c_2 \cap c_3]$ $\{cf_{10}\}$ (divergent)

Knowledge 11: $[a_1, c1 \cap c_3]$ $\{cf_{11}\}$ (divergent)

Knowledge 12: $[a_1 \cup a_2, c_1]$ $\{cf_{12}\}$ (convergent)

Knowledge 13: $[a_1 \cup a_2, c_3]$ $\{cf_{13}\}$ (convergent)

Then certainty factors of knowledge 10 thru 13 can be obtained as in equation (10) thru (13).

$cf_{10}=cf[a_1, c1 \cap c_2 \cap c_3] =\min[cf_5, cf_6, cf_7]$ (10)

$cf_{11}=cf[a_1, c_1 \cap c_3]=\min[cf_5, cf_7]$ (11)

$cf_{12}=cf[a_1 \cup a_2, c_1]=cf_{5,8}=cf(cf_5, cf_8)=cf_5+cf_8-cf_5*cf_8$ (12)

$cf_{13}=cf[a1 \cup a_2, c_3]= cf_{7,9}=cf(cf_7, cf_9)=cf_7+cf_9-cf_7*cf_9$ (13)

(3) Reasoning of certainty factor of knowledge with causal relationship (Z-type mapping):

Knowledge 14: $[a_l, c_1]$ $\{cf_{14}\}$

Knowledge 15: $[c_1, c_2]$ $\{cf_{15}\}$

To reason knowledge 16: IF a_l THEN (c_1 AND c_2) $\{cf_{16}\}$, in short, $[a_1, c_1 \oplus c_2]$ $\{cf_{16}\}$, then the certainty factor of knowledge 16 can be obtained as in equation (14).

$cf_{16}=cf[a_1, c_1 \oplus c_2]=cf_{14}*cf_{15}$ (14)

(4) Reasoning of certainty factor of knowledge with paradoxical relationship (abnormal mapping) (meanwhile with convergence and divergence mapping):

Knowledge 17 : $[a_l, c_1]$ $\{cf_{17}\}$

Knowledge 18: $[a_1, \text{NOT } c_1]$ $\{cf_{18}\}$

Knowledge 19: $[\text{NOT } a_1, c_1]$ $\{cf_{19}\}$

To reason the following knowledge:

Knowledge 20: IF a_l THEN (c_1 AND NOT c_1) $\{cf_{20}\}$, in short, $[a1, c1 \cap \sim c1]$ $\{cf_{20}\}$ (divergent)

Knowledge 21: IF (al AND NOT a1) THEN (c_1) $\{cf_{21}\}$, in short, $[a_1 \cup \sim a_1, c_1]$ $\{cf_{21}\}$(convergent)

Certainty factors of knowledge 20 and 21 can be obtained as in equation (15) and (16).

$cf_{20}=\min[cf_{17}, cf_{18}]$ (15)

$cf_{21}=cf_{17,19}=cf(cf_{17}, cf_{19})=cf_{17}+cf_{19}-cf_{17}*cf_{19}$ (16)

4.3 Model of uncertainty knowledge value-added reasoning

A theory of Decision Factor (DF) was proposed, where DF denotes certainty extent and reliable degree of knowledge, conducive in assistant decision.

Definition: decision factor = certainty factor * reliable factor

To show in symbols, DF=cf*RF (17)

where

certainty factor (cf): how certain the knowledge exists, between 0 and 1. cf=1 indicates that it is totally certain to exist, cf=0 indicates that it is totally certain not to exist. A bigger cf value means higher certainty of it.

Reliable Factor (RF): the reliable extent of the knowledge in respect to external conflicting knowledge, ranging between 0 and 1. RF =1 indicates that it is totally reliable, RF =0 indicates that it is totally unreliable. A bigger RF means it is more reliable.

Decision Factor (DF): a comprehensive expression of certainty extent and reliability of a knowledge, ranging between 0 and 1. A bigger DF means that the knowledge more certainly exists and is more reliable, and hence better for decision. DF =1 indicates that the knowledge is certain and totally reliable, DF =0 means that the knowledge doesn't exist certainly.

4.4 Value-Added treatment Inference of rule-based uncertainty knowledge with various relationships

In summary one of these 6 knowledge correlations and corresponding value-added treatments, the result is shown in fig. 2 and fig. 3.

4.5 Uncertainty knowledge Value-Added treatment Inference algorithm

The infrastructure of Rule-based Uncertainty Knowledge Value-Added Treatment Inference Algorithm (RUKVATIA), shown in Fig. 4, is described as follows:

Input: normal rule-based uncertainty knowledge

Output: rule-based uncertainty value-added knowledge

Step1: using Conditional Probability Knowledge Similarity Algorithm and rule-based knowledge similarity calculation system [1], can obtain the antecedent similarity matrix, consequent similarity matrix, knowledge similarity matrix, antecedent consequent similarity matrix, and consequent antecedent similarity matrix quickly and correctly find the knowledge sets which have various specific relationships and can be value-added treated

Step2: make RF reasoning according to knowledge relation and theory of reliable factor

Step34: make cf reasoning according to knowledge relation and theory of net certainty factor

Step4: make DF reasoning according to knowledge relation and theory of decision factor

Step5: make uncertainty knowledge value-added treatment reasoning according to knowledge relation

Step6: save the value-added rule-based uncertainty knowledge

Step7: end

5. Conclusion and future development

In summary of the above analyses, this study acquires

the following three conclusions:
(1) A Reliable Factor (RF) is proposed to characterize the knowledge with conflict, overlapping or inconsistent data size; and the certainty factor (cf) is also recommended to reflect how reliable a knowledge is; and the Decision Factor (DF) is put forward to indicate the objective of knowledge assistant decision.
(2) The Rule-based Uncertainty Knowledge Value-Added Treatment Inference Algorithm (RUKVATIA) is set up to make a fast Value-Added treatment of rule-based uncertainty knowledge, widely applicable in assistant decision of various kinds of knowledge.

In future, based on RUKVATIA, a WEB version of Uncertainty Rule-Based Knowledge Value-Added Treatment Distance System (URKVATDS) is developed, and the similarity algorithm can be used to collect quickly from knowledge base the knowledge case most similar to tested case, combining Knowledge-Based Reasoning (KBR) or Case-Based Reasoning (CBR) to conduct assistant decision or prediction in reality.

6. References

[1] Chin-Jung Huang, Ying-Hong Lin (2005), "Rule-based Knowledge Similarity Distance Calculation System -Using Conditional Probability Knowledge Similarity Algorithm," Proceeding of the 2005 International Workshop on Distance Education Technologies, Banff, Canada, pp.458-463.
[2] Durkin, J. (1994), "Expert Systems Design and Development," Prentice Hall, Englewood Cliffs , NJ.
[3] Gopika Kannan and K.B. Akhilesh (2002), "Human capital knowledge value added A case study in infotech, " Journal of Intellectual Capital, Vol. 3, No.2, pp.167-179.
[4] Kunhuang Huarng, Dick B. Simmons (1996), "An Object Knowledge Canonical Form for Knowledge Reus," Expert System With Applications, Vol. 10, No. 1, pp.135–146
[5] Michael Negnevitsky (2002), Artificial Intelligence, USA: Addison-Wesley Co.
[6] M. Lynne Markus (2001), "Toward a Theory of Knowledge Reuse: Types of Knowledge Reuse Situations and Factors in Reuse Success," Journal of the Management Information Systems, Vol. 18, No.1, pp.57-93.
[7] Ron Hyson (2001), "Adding value to enterprise modeling, " AI Expert, Vol.3, No.2, pp.26-33.
[8] Shortliffe, E.H. and Buchanan, B.G. (1975), "A model of inexact reasoning in medicine," Mathematical Biosciences, Vol. 23(3/4), pp.351-379.

7. Acknowledgements

The authors would like to thank the financial support of the National Science Council of Taiwan government and grant number: NSC-94-2213-E-129-013.

Definition: RF of knowledge with same antecedent ($0 \le RF \le 1$)

$$RF = \frac{total\ of\ knowledge\ complying\ corresponding\ consequents\ with\ same\ antecedent}{total\ of\ knowledge\ with\ same\ antecedent} \quad (1)$$

RF of knowledge with same consequent

$$RF = \frac{total\ of\ knowledge\ complying\ corresponding\ antecedents\ with\ same\ consequent}{total\ of\ knowledge\ with\ same\ consequent} \quad (2)$$

Table 1. Knowledge Relationships

Knowledge Relation	Relationship	Correspondence
1. Knowledge set with completely the same antecedents	One to many	Divergent
2. Knowledge set with completely the same consequents	Many to one	Convergent
3. Knowledge set with completely the same antecedents or consequents	Many to many	Interleaving
4. Knowledge set in which one antecedent acts as another consequent	Cause and effect	Z-shape
5. Knowledge set with contradictive antecedents and consequents	Contradiction	Abnormal
6. Knowledge set with independent antecedents and consequents	Independence	Parallel

Table 2. RF inference result

Example	Original knowledge	Value added inference knowledge
five knowledge sets with the same antecedent a1 in knowledge base	K1:IF a1 THEN V<42 K2:IF a1 THEN V>30 K3:IF a1 THEN V>52 K4:IF a1 THEN V<60 K5:IF a1 THEN V=50	K1:IF a1 THEN V≤30 (RF=0.4) K2:IF a1 THEN 30<V<42 (RF=0.6) K3:IF a1 THEN 42<V≤52 (RF=0.4,V≠50) K4:IF a1 THEN V=50 (RF=0.6) K5:IF a1 THEN 52<V<60 (RF=0.6) K6:IF a1 THEN V≥60 (RF=0.4)

Type 1: Value-adding treatment reasoning of knowledge set with the same antecedent (one to multiple)

original uncertainty knowledge					
antecedent	consequent	cf_a	cf_{rule}	cf	RF
a_1	c_1	cfa_1	cf_{rule1}	$cf_1 = cf_{a1} * cf_{rule1}$	RF_1
a_1	c_2	cfa_1	cf_{rule2}	$cf_2 = cf_{a1} * cf_{rule2}$	RF_2
a_1	c_3	cfa_1	cf_{rule3}	$cf_3 = cf_{a1} * cf_{rule3}$	RF_3

Reasoning

value-adding knowledge				
antecedent	consequent	cf	RF	DF
a_1	c_1 AND c_2	$\min[cf_1, cf_2]$	$\min[RF_1, RF_2]$	$\min[cf_1, cf_2] * \min[RF_1, RF_2]$
a_1	c_1 AND c_3	$\min[cf_1, cf_3]$	$\min[RF_1, RF_3]$	$\min[cf_1, cf_3] * \min[RF_1, RF_3]$
a_1	c_1 AND c_2 AND c_3	$\min[cf_1, cf_2, cf_3]$	$\min[RF_1, RF_2, RF_3]$	$\min[cf_1, cf_2, cf_3] * \min[RF_1, RF_2, RF_3]$
a_1	c_1 OR c_2 OR c_3	$\max[cf_1, cf_2, cf_3]$	$\max[RF_1, RF_2, RF_3]$	$\max[cf_1, cf_2, cf_3] * \max[RF_1, RF_2, RF_3]$
a_1	c_1	cf_1	$RF = RF_1$	$cf_1 * RF_1$
	c_2	cf_2	$RF = RF_2$	$cf_2 * RF_2$
	c_3	cf_3	$RF = RF_3$	$cf_3 * RF_3$

Fig. 2. Value-added treatment inference of knowledge sets with same antecedent

Example of Type 1:

original uncertainty knowledge					
antecedent	consequent	cf_a	cf_{rule}	cf	RF
a_1	c_1	0.8	0.8	0.64	0.33
a_1	c_2	0.8	0.6	0.48	0.67
a_1	c_3	0.8	0.9	0.72	0.67

Reasoning

value-adding knowledge					
antecedent	consequent	cf	RF	DF(RF≠1)	DF(RF=1)
a_1	c_1 AND c_2	0.48	0.33	0.16	0.48
a_1	c_1 AND c_3	0.64	0.33	0.21	0.64
a_1	c_1 AND c_2 AND c_3	0.48	0.33	0.16	0.48
a_1	c_1 OR c_2 OR c_3	0.72	0.67	0.48	0.72
a_1	c_1	0.64	0.33	0.21	0.64
	c_2	0.48	0.67	0.32	0.48
	c_3	0.72	0.67	0.48	0.72

Fig. 3. Example for the value-added treatment inference of knowledge sets with same antecedent

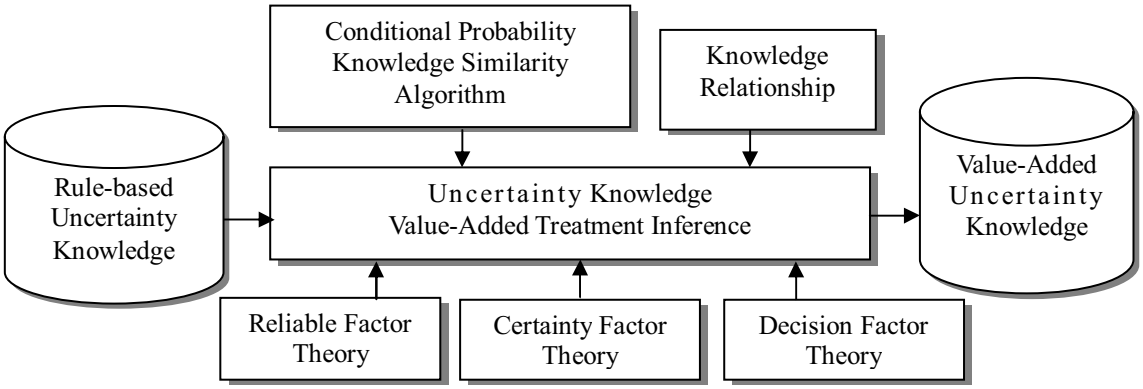


Fig. 4. RUKVATIA infrastructure

A Rule-Based Expert System for the Diagnosis of Convergence Problems in Circuit Simulation

Christopher W. Lehman D.CS
IBM
Colorado Springs, CO 80903

Mary Jane Willshire Ph.D.
Department of Computer Science
Colorado Technical University
Colorado Springs, CO 80919

Abstract Industry standard circuit simulators use mathematical techniques that do not always guarantee a successful simulation outcome, resulting in a significant loss of productivity as the circuit designer attempts to diagnose the source of the failure. This paper addresses the problem of improving the diagnosis of circuit simulation non-convergence through computer automation of the diagnostic process. SOAR – Simulation Output Analysis and Recommendations – is the software application that implements the proposed solution to the problem.

Circuit simulation convergence failure analysis heuristics can be encapsulated into a rule-based expert system environment that emulates the reasoning of experienced circuit designers. SOAR integrates expert system technology with a modern web-based framework. Utilizing a database to store and retrieve simulation information gives SOAR the ability to diagnose simulations ranging in size from gate-level circuits to full chip architectures with virtually no increase in processing time.

Keywords: simulation, circuit convergence, expert system, diagnostic systems, DC operating point/bias point analysis

I. INTRODUCTION

Traditional circuit design was characterized by building a circuit prototype, and then providing the prototype with various stimuli (such as input signals, temperature variations, and changes in power supply voltages). Laboratory equipment was used to make measurements on the circuit prototype as the stimuli were applied.

With the advent of Very Large Scale Integrated (VLSI) circuit design, prototypes are no longer practical. The breadboard prototype of the circuit will not have the same electrical, mechanical and parasitic components as the VLSI equivalent.

Software provides a solution to the problem. Computer programs that simulate the performance of an electronic circuit can be used to confirm the performance of the circuit prior to committing fabrication resources to the actual construction of the circuit. Circuit simulation software has therefore become a vital part of the IC design cycle.

The most prevalent circuit simulators use the Newton Raphson (NR) method (or some variation) for solving nonlinear systems of equations. The NR algorithm within the circuit simulator starts with an initial guess for each node voltage in the circuit and begins its iteration process. The iterations continue until the solution is reached within the specified allowed margin of error. The drawback to the NR method is that it will not converge on a solution unless the required initial guess to start the iterations is sufficiently close to the solution [1], [3], [4]. When non-convergence occurs, the circuit designer must begin the time-consuming process of determining the cause of the failure.

The knowledge associated with the diagnosis and elimination of convergence problems in circuit simulation includes written knowledge, composed of books, papers, memos and "how to" documents, and user experience and expertise. This knowledge can potentially be analyzed and rendered into a set of inference rules. In other words, rule-based representation of the available knowledge is possible [5].

II. RELATED WORK

A. Avoiding Convergence Failure

A search of the literature indicates that circuit simulation convergence issues are generally tackled by either 1) attempting to embellish existing techniques (algorithmic or numeric) for avoiding convergence failure, or 2) by providing heuristic solutions for non-convergence after the simulation fails [6].

It is possible to have a realistic circuit, but one or more models that represent the circuit elements may have difficulty in converging. Simulator engines therefore typically have built-in strategies for dealing with this type of circuit.

B. Identifying Convergence Failure After the Fact

When convergence failure occurs the designer must resort to trial and error in an attempt to a) identify the source of the problem, and b) make the necessary corrections prior to restarting the simulation of the circuit. As a result, vendors

Manuscript received May 05, 2006.

usually provide their customers with guidelines for troubleshooting convergence problems [1], [2].

In addition, the design community within an organization will often publish internal memos, guidelines and manuals that describe convergence problems that have previously been encountered.

A designer with many years of experience in circuit design will also have developed a certain amount of expertise in solving convergence problems. An experienced designer can be quite adept at solving convergence problems, but in many cases the expertise has not been formally captured.

C. Expert Systems and VLSI Design

Expert Systems have been utilized within the VLSI design cycle, especially during the 1980's when both technologies began to mature. The height of the Expert System/VLSI fusion was during the 1988 to 1991 time period [6]. However, researchers during this period were working in problem domains that were too broad [6]. Despite the rather gloomy outcomes of the early attempts at expert system solutions, expert systems continue to show promise in the areas of VLSI test, and for solving specialized VLSI and printed circuit board placement and routing problems [6].

III. Architecture of the Expert System and User Interface

A. Architectural Considerations

An important portion of the work described in this paper consisted of the task of choosing the most appropriate platform from which to build the expert system. The system needed to be low-cost, but flexible enough to work with large amounts of textual data. The C Language Integrated Production System (CLIPS) [7] was selected as the expert system environment because it is public domain software, the C source code is available for modification, it uses a simple but extensible command-line driven user interface, it contains a number of conflict resolution strategies, and the authors have previous experience with the tool.

As noted, the CLIPS user interface is constrained to command line invocation and output to either a file or a text-only environment. For academic purposes, this is perfectly acceptable. For the more general case, a more flexible solution is desirable. A web-based user interface was determined to be the most efficient method of deploying the application for the following reasons:

1 Web-based applications are accessible anywhere, at any time.

2 Software distribution can be controlled from the originating web page, and can also more easily track customer usage patterns and problems.

3 Validation of expert system knowledge bases can be a major challenge [8]. A web-based expert system's releases can be more readily controlled by the author.

Another major architectural consideration was the amount of data that the expert system would need to access. Simulation output files routinely consist of megabytes of textual information contained in structured patterns. Repeated searches over the entire file would be time-consuming and repetitive. On the other hand, programmatic extraction of pertinent data is possible because of the repeating textual structures. Once the data is extracted, it could be transformed into more regular structures such as database records. Database queries on simulation information were therefore determined to be feasible and would allow for faster processing of the available information about the simulation.

Given this analysis of the implementation features, the decision was made to utilize the Ruby on Rails (RoR) software environment for the construction of the user interface to the expert system.

Ruby on Rails [12] uses the Ruby programming language within an open-source framework for developing web-based applications. The Ruby language [9] combines the object-oriented programming power of Smalltalk with the scripting power of Perl.

B. SOAR: Simulation Output Analysis and Recommendations

The expert system that has been developed is called **SOAR**, that is, a program that provides for **S**imulation **O**utput **A**nalysis and **R**ecommendations. A block diagram of the overall SOAR architecture is shown in Figure 1.

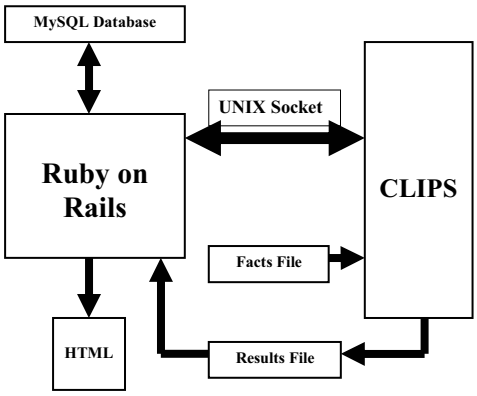


Figure 1 - SOAR Block Diagram

The primary design elements of SOAR are:

- The use of a UNIX socket as the means of communication between the Ruby on Rails application and the CLIPS expert system.
- The creation of files to transfer information into and out of the CLIPS expert system's text-based external interfaces.
- A reliance on the built-in database connection features of the Ruby on Rails application.
- A uniform "look and feel" of the Graphical User Interface through the use of Cascading Style Sheets coupled with HTML-embedded Ruby code.

1) User Interface

The simulation file is read and parsed, and the parameter, option and model information is extracted, reformatted and

placed in a MySQL database. The user has the option of viewing the extracted parameters, options and model information by clicking on buttons placed on the web page.

Once the user has finished inspecting the various database records, the "Start The Checks" button is pressed, and program control is passed to the CLIPS expert system. Control is passed back to the Ruby on Rails interface when CLIPS has completed its analysis, and a web page is displayed that contains recommendations returned by the CLIPS program.

2) Database

Ruby on Rails was specifically designed to interact with databases. Once a database is created, RoR can access the data almost immediately via a built-in code generation process that sets up the necessary hooks for basic operations on database tables, commonly referred to as CRUD – Create, Read, Update, and Delete. Remaining consistent with the open-source approach to the research, the MySQL database [10] was selected.

3) CLIPS Implementation

In an attempt to maintain consistent Object Oriented Programming (OOP) coding standards, the CLIPS rule set was written in COOL, the CLIPS Object Oriented Language, as defined in [11]. Circuit simulators, and their associated rule sets, are defined using the *defclass* declaration. Additional simulators and rule sets can be defined by declaring additional *defclasses*. When a rule set changes, only the associated classes are affected – the verification and validation of the CLIPS output is therefore confined only to the classes associated with the change, thus reducing the scope of the verification and validation effort.

The function, SOAR_sock_client.c, establishes a socket connection with the Ruby on Rails GUI. The user function was compiled and linked with the standard issue CLIPS modules, creating a custom executable. The CLIPS version used was 6.22.

When the CLIPS program is invoked, it begins to request information about the simulation. This is the "fact-gathering" stage of the operation. The facts to be extracted from the simulation are those associated with user-configurable options that direct the processing details of the simulation. These details include such features as error tolerance values, voltage and current constraints, algorithmic options during runtime, and options to regulate the simulation speed and accuracy. The options typically consist of a keyword and a value, although there are some options that when specified, act simply as flags to modify the behavior of the simulation. The Rails interface is used to capture a CLIPS request for a particular fact to be extracted. Rails sends the request to the database server, and then passes the result back to the CLIPS program. This process continues until all database queries are sent and processed, i.e. until all required facts are assembled. The appropriate rule sets are then applied to the fact set. When rule processing is complete, control is passed back to the RoR interface. The RoR interface displays the recommendations to the user in HTML table format. A partial rendering of a sample web page presented to the user is shown in Figure 2.

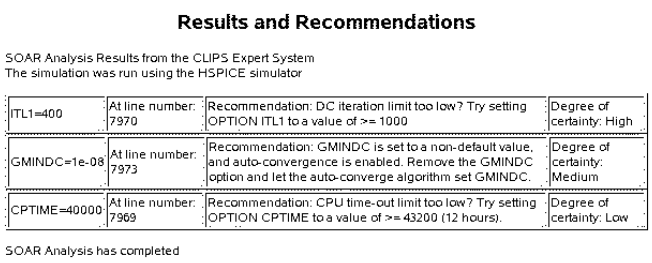


Figure 2 - Results Displayed to the User

IV. TEST CASES

The test cases were derived from actual circuits created for use in CMOS Static RAM (SRAM) memory chips, using many full-custom analog circuits in order to meet stringent design requirements; as a result, convergence problems are commonly encountered when simulating the various analog components of an SRAM design. Specific test case selection was based upon three general requirements:

1 The test cases needed to be based upon real problems encountered by real designers. Each test case represented a circuit that, at one point during the design cycle, failed to converge. Each test case was analyzed by SOAR, i.e. the simulation information was read into the SOAR application, and a solution was returned. When the recommended solution was implemented and the circuit was re-simulated, each test case then attained convergence.

2 The sizes of the circuit test cases were selected to be broad enough to represent the entire design cycle in order to demonstrate that SOAR would work with state of the practice designs. The test cases represent circuit sizes spanning three orders of magnitude, from small 16 node circuit simulations to full chip simulations of over 11,000 nodes.

3 The scope of the work was limited to DC operating point simulations of analog circuits, and the test cases were selected accordingly.

V. RESULTS

Table 1 presents the results of the individual test cases.

Table 1 - SOAR Test Case Results

Name	Test	Number of Circuit Nodes	Circuit Description	CLIPS Run Time (seconds)
Case1	No Errors	1161	tAA	25
Case2	All Errors	1161	tAA	25
Case3	ACCT	1161	tAA	27
Case4	ITL1	1161	tAA	25
Case5	CPTIME	1161	tAA	26
Case6	RELV	16	Trip Point	25
Case7	ABSV	16	Trip Point	27
Case8	ABSI	16	Trip Point	25

Case9	Error Tol.	16	Trip Point	25
Case10	KCLTEST	510	LVDetect	25
Case11	DV	25	Senseamp	30
Case12	DCON	11131	Read 100C	32
Case13	GMINDC	11131	Read 25C	36
Case14	CONVERGE	11258	tHA	32

Case 1 was a custom simulation file that deliberately tests for CLIPS "false positives"; all CLIPS rules are exercised, but no rules should generate a recommendation. Case 2 was also contrived in order to force all rules to detect and display a recommendation. The remaining test cases were used without modification from the original simulations.

Table 1 contains circuits with node counts that span three orders of magnitude. It is remarkable to note that the run times are virtually the same regardless of circuit node count. Transforming the circuit information into database records is a key element in the ability of the SOAR application to scale up to VLSI-sized circuits.

In all test cases, the simulation originally failed to converge. When the simulation was re-run using the solution recommended by SOAR, the circuit attained convergence. A 100 per cent accuracy was attained for the test case set.

VI. Future Direction of Research

Based upon the work thus far completed, a number of enhancements and new research directions can be contemplated.

Not all of the possible convergence problem classes are represented in the current version of SOAR. Additional queries and associated rule sets could be added for more comprehensive coverage of circuit simulation convergence scenarios.

Due to the OOP modular construction of the GUI and the CLIPS code, additional circuit simulators can be added without affecting the existing components of the system.

Database queries pertaining to model information and additional rule sets designed specifically to diagnose potential problems associated with model definitions could be added to the application.

Now that a method for transforming circuit simulation information into a database has been developed, analysis of a circuit is not confined to convergence problems alone. SOAR could be altered to act as a "best practices" guide for use by new college graduates who are just beginning to learn to be circuit designers.

VII. Conclusion

The research activity in this paper has addressed the problem of non-convergence associated with analog VLSI circuits, an ongoing problem for the popular simulation environments that use the Newton-Raphson root solving algorithm.

This paper has demonstrated that rule-based expert systems can successfully diagnose analog circuit simulation convergence problems for circuit complexities ranging from simple gate-level functions and all the way up to full chip design hierarchies. The research described in this paper:

1 Converted heuristics for diagnosing DC operating point convergence problems into a software application.

2 Implemented a process that transforms text-based circuit simulation information into database records.

3 Designed and implemented processes by which the simulation database records were converted into sets of facts suitable for use by a rule-based expert system.

4 For every non-convergent test case circuit, the recommendations made by SOAR allowed the circuit to converge.

5 The expert system successfully processed the massive amounts of information associated with state of the practice VLSI designs, thus demonstrating the ability of the application to scale up to the data-intensive rigors of modern circuit design.

6 The test set contained circuits with node counts that spanned three orders of magnitude, but the run times were virtually the same regardless of circuit node count. Transforming the circuit information into database records was a key element in the ability of the SOAR application to scale up to VLSI-sized circuits.

References

[1] Ron Kielkowski, *Inside SPICE* Second Edition. New York, New York, McGraw-Hill, Incorporated, 1998.

[2] Synopsys, Incorporated, *Convergence*. unpublished.

[3] Francky Leyn and Georges Gielen and Willy Sansen, "An Efficient Root Solving Algorithm with Guaranteed Convergence for Analog Integrated CMOS Circuits," ACM ICCAD98, pp. 304–307, 1998.

[4] William H. Press and Brian P. Flannery and Saul A. Teukolsky and William T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*. New York, New York, Cambridge University Press, 1993.

[5] Frederick Hayes-Roth, "Rule-Based Systems," Communications of the ACM, Volume 28, Number 9, pp. 921–932, 1985.

[6] C. W. Lehman, "A Rule Based System for the Diagnosis of Convergence Problems in Circuit Simulation," Doctoral dissertation, Dept. Comp. Sci., Colorado Technical University, Colorado Springs, CO, 2006.

[7] Joseph Giarratano and Gary Riley, *Expert Systems Principles and Programming*. Boston, Massachusetts, PWS Publishing Company, 1998.

[8] Avelino J. Gonzalez and Douglas D. Dankel, *The Engineering of Knowledge-Based Systems Theory and Practice.* Prentice-Hall, Incorporated, Englewood Cliffs, New Jersey, 1993.

[9] RubyCentral Home Page, Available: http://www.rubycentral.com, 2005.

[10] MySQL AB Home Page, Available: http://www.mysql.com, 2005.

[11] Chris Culbert and Gary Riley, *CLIPS Reference Manual Volume I Basic Programming Guide*. unpublished.

[12] Dave Thomas and David Heinemeier Hansson, *Agile Web Development with Rails*. Raleigh, North Carolina, The Pragmatic Bookshelf, 2005.

Using Data Mining Schemes for Improvement on System Performance in Virtual Environments

Shao-Shin Hung and Damon Shing-Min Liu
Department of Computer Science and Information Engineering
National Chung Cheng University
Chiayi, Taiwan 621, Republic of China
{hss, damon}@cs.ccu.edu.tw

Abstract

Object correlations are common semantic patterns in virtual environments (VE). They can be exploited for improve the effectiveness of storage caching, prefetching, data layout, and disk scheduling. However, little approaches for discovering object correlations in VE to improve the performance of storage systems. Moreover, current methods are presented for typical data mining datasets and not suitable for our virtual reality datasets. In this paper, we develop a class of view-based projection-generation method for mining various frequent sequential traversal patterns in the virtual environments. The frequent sequential traversal patterns are used to predict the user navigation behavior and help to reduce disk access time with proper placement patterns into disk blocks. Finally, we have done extensive experiments to demonstrate how these proposed techniques not only significantly cut down disk access time, but also enhance the accuracy of data prefetching.

1. Introduction

With the advent of advanced computer hardware and software technologies, virtual environments (VE) are becoming larger and more complicated. To satisfy the growing demanding for fidelity, there is a need for interactive and intelligent schemes that assist and enable effective and efficient storage management.

Unfortunately, it is not an easy task to exploit the intelligence in storage systems [8]. One primary reason is the system latency between VE applications and storage systems. In such a case, VE do not consider the problem of access times of objects in the storage systems. They always simply concerned about how to display the object in the next frame. As a result, the VE can only manage data at the rendering and other related levels without knowing any semantic information such as semantic correlations between data. This motivates a more powerful analysis tool to discover more complex patterns, especially semantic patterns, in storage systems. Therefore, the aim of our work is to decrease this latency through intelligent organization of the access data and enabling the clients to perform predictive prefetching [8].

In this paper, we consider the problem and solve this using data mining techniques [2]. Clearly, when users traverse in a virtual environment, some potential semantic characteristics will emerge on their traversal paths. If we collect the users' traversal paths, mine and extract some kind of information of them, such meaningful semantic information can help to improve the performance of the interactive VE. For example, we can reconstruct the placement order of the objects of 3D model in disk according to the common section of users' path. Exploring these correlations is very useful for improving the effectiveness of storage caching, prefetching, data layout, and disk scheduling.

This paper proposes *VSPM* (*Viewed-based Sequential Pattern Mining*), a method which applies a data mining technique called *frequent sequential pattern mining* to discover object correlations in VE. Specially, we have modified several recently proposed data mining algorithms called *FreeSpan* [4] and *PrefixSpan* [10] to find object correlations in several traversal traces collected in real systems. To the best of our knowledge, *VSPM* is the first approach to infer object correlations in a VE. Furthermore, *VSPM* is more scalable and space-efficient than previous approaches. It runs reasonably fast with reasonable space overhead, indicating that it is a practical tool for dynamically inferring correlations in a VE. Besides, we have also proposed a clustering method to cluster the similar patterns for reducing the access time. It will make similar objects much closer to be accessed in one time. This results in less access times and much better performance.

The rest of this paper is organized as follows. Related works are given in Section 2. In Section 3, we describe our problem formulation. The suggested mining and clustering mechanisms are explained with illustrative examples shown in Section 4. Section 5 presents our experiment results. Finally, we summarize our current results with suggestions for future research in Section 6.

2. Related Works

In this subsection, we will briefly describe related works about virtual environments, sequential pattern mining and pattern clustering, respectively.

2.1 Virtual Environments Methods

Since the navigation in virtual environments consists of many different detailed objects, e.g., of CAD data that cannot all be stored in main memory but only on hard disk. Many techniques were proposed for rendering complex models used today, including the use of hierarchical spatial structures, level-of-detail (LOD) management [18], hierarchical view-frustum and occlusion culling [11], working-set management (geometry caching) [18]. In additional, Massive Model Rendering (MMR) system [12] was the first published system to handle models with tens of millions of polygons at interactive frame rates. Besides, many *out-of-core* spatial data

structures [14], including *kd*-trees, quad-trees, *oct*-tree and *R*-trees [14] were presented. On the other side, it is desirable to store only the polygons and not to produce additional data as, e.g., *textures* or *pre-filtered points*. However, polygons of such highly complex scenes require a lot of hard disk space so that the additional data could exceed the available capacities [19]. To meet these requirements, an appropriate data structure and an efficient technique should be developed with the constraints of memory consumptions.

2.2 Sequential Pattern Mining Methods

Sequential pattern mining was first introduced in [8], which is described as follows. A sequence database is formed by a set of data sequences. Each data sequence includes a series of transactions, ordered by transaction times. This research aims to find all the subsequences whose ratios of appearance exceed the minimum support threshold. In other words, *sequential patterns* are the most frequently occurring subsequences in sequences of sets of items. A number of algorithms and techniques have been proposed to deal with the problem of sequential pattern mining. Many studies have contributed to the efficient mining of sequential patterns [4,10]. Almost all of the previously proposed methods for mining sequential patterns are *apriori*-like [4]. Sequential pattern mining algorithms, in general, can be categorized into three classes: (1) *Apriori-based* : *horizontal partition* methods and *GSP* [6] is one known representative; (2) *Apriori-based*: *vertical partition* methods and *SPADE* [5] is one example; (3) *projection-based pattern growth* method, such as the famous *FreeSpan* [10] and *PrefixSpan* algorithms [4].

2.3 Pattern Clustering Methods

The fundamental clustering problem is to partition a given data set into groups (clusters), such that data points in a cluster are more similar to each other (i.e., *intra-similar property*) than points in different clusters (i.e., *inter-similar property*) [9]. There is a multitude of clustering methods available in literature, which can be distinguished with respect to their algorithmic properties [13]. One, *partition algorithms* strive for a successive improvement of an existing clustering. These approaches need information with regard to expected cluster number, *k*. Representatives are: *k*-Means [13] and *k*-Medoid [13]. The other one, *hierarchical algorithms* create a tree of node subsets by successively merging (*agglomerative* approach) or subdividing (*divisive* approach) the objects. In order to obtain a unique clustering, a second step is necessary that prunes this tree at adequate places. Representatives are: *k*-nearest-neighbor and linkage [15].

Although there are many clustering algorithms presented above, they can not be applied to our data set directly. The reasons are as follows [18]. First is that our database is composed of many *transactions*. There is a finite set of elements, called *items* from a common item universe, are contained in a transaction. Every transaction can be presented in a point-by-attribute format, by enumerating all items *j*, and by associating with a transaction the binary attributes that indicates whether *j*-items belong to a transaction or not. Such representation is sparse that two random transactions have very few items in common. Common to this and other examples of point-by-attribute format for transaction data, is high dimensionality, significant amount to zero values, and small number of common values between two objects. Conventional clustering methods, based on this similarity measures, do not work well. Since transactional data is important in clustering profiling, Web analysis, DNA analysis and other applications, different clustering methods founded on the idea of *co-occurrence* of transaction data have developed. They are usually measured by *Jaccard* coefficient $SIM(T_1, T_2)= |T_1 \cap T_2| / |T_1 \cup T_2|$ [20,13].

3. Mining Traversal Histories and Problem Formulation

In this section, we extract the useful information in the access history in the form of sequential patterns. In order to mine for sequential patterns, we assume that the continuous client requests are organized into discrete sessions. *Sessions* specify user interest periods and a *session* consists of *a sequence of client requests* for data items ordered with respect to the time of reference. The client request consists of the objects which a client browse and traverse at will in the VE. We denote this type of clients request as *view*. A session consists of one or more views. In correspond to with terminologies used in data mining, a session can be considered as a *sequence*. The whole *database* is considered as a set of sequences. Formally, let $\sum = \{l_1, l_2, ..., l_m\}$ be a set of *m* literals, called *objects* (also called *items*) [16]. The *view v* is defined as snapshot of sets of objects which a user observes duration the period. A *view* (also called *itemset*) is an *unordered*, non-empty set of objects. A sequence is an *ordered* list of views. We denote a sequence *s* (also called *transaction*) by $\{v_1, v_2, ..., v_n\}$, where v_j is a view and ordered property is obeyed. We also call v_j an *element* of the sequence. An item can occur only once in an element of a sequence, but can occur multiple times in different elements. We assume, without loss of generality, that items in an element of a sequence are in lexicographical order.

A sequence $<a_1\ a_2\ ...\ a_n>$ is *contained in* another sequence $<b_1\ b_2\ ...\ b_m>$if there exist integers $i_1 < i_2 < \ldots < i_n$ such that $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \ ..., a_n \subseteq b_{i_n}$. For example, <(*a*)(*b, c*)(*a, d, e*)> is contained in <(*a*, *b*) (*b, c*)(*a, b, d, e, f*)>, since (*a*) $\subseteq$ (*a, b*), (*b, c*) $\subseteq$ (*b, c*), and (*a, d, e*) $\subseteq$ (*a, b, d, e, f*). However, the sequence <(*c*)(*d*)> is not contained in <(*c, d*)>)and vice versa. The former represents objects *c* and *d* being observed one after the other, while the latter represents objects *c* and *d* being observed together. In a set of sequences, a sequence *s* is *maximal* if *s* in not contained in any other sequence. Let the database *D* be a set of sequences and ordered by increasing recording time. Each sequence records each user's traversal path in the walk through system. The *support* for a sequence is defined as the fraction of *D* that "contains" this sequence. A *sequential pattern p* is a sequence whose *support* is equal to or more than the user-defined threshold. *Sequential patter*

mining is the process of extracting certain sequential patterns whose support exceeds a predefined minimal support threshold. Given a database D of client transactions, the problem of mining sequential patterns is to find the maximal sequences among all sequences that have a certain user-specified minimum support. Each maximal sequence represents a *sequential pattern*.

Finally, we will define our problem in two phases. Phase I: given a sequence database $D = \{s_1, s_2, ..., s_n\}$, we design a efficient mining algorithms to obtain our sequential patterns P; phase II: In order to reduce the disk access time, we distribute P into a set of clusters, such that minimize inter-cluster similarity and maximize intra-cluster similarity.

4. Pattern-Oriented Mining and Clustering Algorithms

In this section, we will explain our sequential pattern mining method, called ***V**iew-based **S**equence **P**attern **M**ining **(VSPM)***. Since our input data are different from those of traditional data mining algorithms [1]. We make several major modifications about the idea of pattern-growth method. Its general idea is to use frequent items to recursively project sequence databases into a set of smaller projected databases and grow subsequence fragments in each projected database. This process partitions both the data and set of frequent sequential patterns to be tested, and confines each test being conducted to the corresponding smaller projected database.

4.1. View-based Sequential Pattern Mining (VSPM) Algorithm

Now, we will explain our mining algorithms. The main ideas come from both *bounded-projection* and *pattern appending* mechanisms. The *bounded-projection* mechanism has one special characteristic, i.e., it always projects the remaining sequence recursively after a new sequential pattern found. They will not mine the objects across the different prefix views. As a result, we would mine the trimmed database recursively. The *pattern appending* mechanism uses the concept of *prefix property*. When we want to find a new sequential pattern in our database, we use the sequential pattern found in previous round as prefix, and append a new object as the new candidate pattern for verification. If the candidate pattern satisfied the minimum support, we regard it as a new sequential pattern and create a bounded projection of it recursively. In order to explore the interesting relationships among these objects, we propose two different kinds of appending methods − called *Intra-View-Appending* method and *Inter-View-Appending* method. The *Intra-View-Appending* method is used to *append a new object in the same view*, and the *Inter-View-Appending* method is used to *append a new object in the next view*. Demonstration example will be given later. The following is the pseudo codes of view sequence mining algorithm.

View-based Sequential Pattern Mining (VSPM) Algorithm

// D is the database. P is the set of frequent patterns, and is set to empty initially.

Input: D and P.
Output: P
Begin
1. Find length-1 frequent sequential patterns.
2. **While** (any projected sub-database exits) **do**
3. **Begin**
4. Project corresponding sub-sequences into sub-databases under the intra-view appending and inter-view appending.
5. Mine each sub-database corresponding to each projected sub-sequence.
6. Find all frequent sequential patterns by applying step 4 and step 5 on the sub-databases recursively.
7. **End**; // while
8. return P;
9. **End;** // procedure *VSPM* ends

Example 1 (*VSPM*). Given the traversal data base S and *min_support* = 3, we demonstrate the complete steps as follows.
Path1: <(1, 2)(3, 4)(5, 6)>.
Path2: <(1, 2)(3, 4)(5)>.
Path3: <(1, 2)(3)(4, 5)>.

Step1. *Find frequent patterns* with length-1. //in the form of "*item: support*"
First, we will have the following data: 1:3, 2:3, 3:3, 4:3, 5:3, 6:1. Therefore, we have *length-1* frequent sequential patterns: <1>, <2>, <3>, <4>, and <5>. Finally, we will have 5 projection-based sub-databases <1>_DB, <2>_DB, <3>_DB, <4>_DB and <5>_DB, respectively.

Step2. Take the projection-based sub-database, <1>_DB, for example. First, since item 2 and item 1 are *in same view*, the *intra-view appending* works. After the projection, we will get the sub-database <(1,2)>_DB. And the original database is shrunk to the following database.
P1: <(3,4)(5,6)>; P2: <(3,4)(5)>; P3: <(3)(4,5)>.
In this step, pattern <(1,2)> becomes a frequent sequent pattern since its support satisfies the minimum support. Next, item 3 is projected for the candidate.

Step3: the remaining steps are the same as the above. The final mining result is depicted in Figure 1. In Figure 1, the patterns which contain item 6 are circled. They show that the differences between projected-based mining and non-projected-based mining. In other words, without projecting mechanism, we have to expand *eight* sub-databases for candidates (i.e., *two* "stop" without circled plus *six* "stop" with circled). Compared to this case, with projecting mechanism, we only expand *two* sub-databases for candidates (i.e., "stop" without circled).

4.2. Disk Organization by Clustering Sequential Patterns

Clustering is a good candidate for inferring object correlations in storage systems. As the previous sections mentioned, object correlations can be exploited to improve storage system performance. First, correlations can be used to

direct prefetching. For example, if a strong correlation exists between objects *a* and *b*, these two objects can be fetched together from disks whenever on of them is accessed. The disk read-ahead optimization is an example of exploiting the simple data correlations by prefetching subsequent disk blocks ahead of time. Several studies [10,15] have shown that using these correlations can significantly improve the storage system performance. Our results in Section 6.2.2 demonstrate that prefetching based on object correlations can improve the performance much better than that of non-correlation layout in all cases.

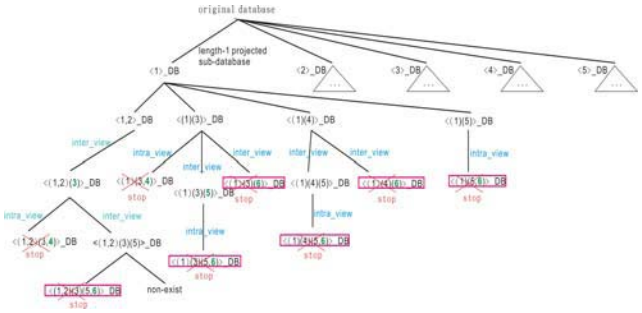


Fig. 1 Demonstration of our *VSPM* for generating projected-based sub-databases and sequential patterns.

A storage system can also lay out data is disks according to object correlations. For example, a object can be collocated with its correlated objects so that they can be fetched together using just one disk access. This optimization can reduce the number of disk seeks and rotations, which dominate the average disk access latency. With correlation-directed disk layouts, the system only needs to pay a one-time seek and rotational delay to get multiple objects that are likely to be accessed soon. Previous studies [8,9] have shown promising results in allocating correlated file blocks on the same track to avoid track-switching costs.

The main idea of our clustering approach is that of defining a new notion of cluster *centroid*, which represents the common properties of cluster elements. Similarity inside a cluster is hence measures by using the cluster representative. The cluster representative becomes a natural tool for finding an explanation of the cluster population. Our definition of cluster centroid is based on a data representation model which simplifies the ones used in pattern clustering. In fact, we use compact representation of Boolean vector *v* that states only presence and absence of items, while traditional pattern clustering methods require to store the frequencies of items. In this paper, we show that using our concept of cluster centroid associated with *Jaccard distance* [20,13] we obtain results having a quality comparable with other approaches used in this task, but we have better performances in terms of execution time. Moreover, cluster representatives provide an immediate explanation of cluster features.

4.3 Distance Measure

In the simplified hypothesis that frequent patterns do not contain frequencies, but behave simple as Boolean vectors (like a value 1 corresponds to the presence and a value 0 corresponds to the absence), and a more intuitive but equivalent way of defining the *Jaccard distance function* can be provided. This measure captures our idea of similarity between items, which are directly proportional to the number of common values, and inversely proportional to the number of different values for the same item.

Definition 1: ***Intra-distance measure (Co-occurrence)***
Let P_1 and P_2 be two sequential patterns. We can represent $D(P_1, P_2)$ as the normalized difference between the cardinality of their union and the cardinality of their intersection:

$$D(P_1, P_2) = 1 - \frac{|P_1 \cap P_2|}{|P_1 \cup P_2|} \quad \text{(Eq. 1)}$$

4.4. Cluster representative and pattern clustering algorithm

Intuitively, a cluster representative for virtual environment data should model the content of a cluster, in terms of the objects that are most likely to appear in a pattern belonging to the cluster. A problem with the traditional distance measures is that the computation of a cluster representative is computationally expensive. As a consequence, most approaches [13] approximate the cluster representative with the Euclidean representative. However, those approaches may suffer the following drawbacks:

- Huge cluster representatives cause poor performances, mainly because as soon as the clusters are populated, the cluster representatives are likely to because extremely huge.
- For different kinds of patterns, it seems to be difficult to find the proper cluster representatives.

In order to overcome such problems, we can compute an approximation that resembles the cluster representatives associated to Euclidean and mismatch-count distances. Union and intersection seem good candidates to start with. Since our clustering operations are based on set operations, we ignore the order of frequent patterns.

To avoid these undesired situations, we supply three tables. The first table is *FreqTable*. It records the frequency of any two patterns co-existing in the database *D*. The second table is *DistTable*. It records the distance between any two patterns. The last table is ***C****luster*. It records how many clusters are generated. The following is our clustering algorithm.

Pattern Clustering Algorithm
// *P* is the set of frequent patterns. *T* is the set of clusters, and is set to empty initially.
Input: *P* and *T*.
Output: T
Begin
1. *FreqTable*={ft_{ij}| the frequency of $pattern_i$ and $pattern_j$ co-existing in the database *D*};
2. *DistTable*={dt_{ij}| the distance between of $pattern_i$ and $pattern_j$ in the database *D*};
3. C_1={C_i| At the beginning each pattern to be a single cluster }
4. // Set up the Extra-Similarity Table for evaluation
5. M_1= Intra-Similar (C_1, ∅);

6. $k = 1$;
7. **while** $|C_k| > n$ do **Begin**
8. C_{k+1} = PatternCluster (C_k, M_k, *FreqTable, DistTable*);
9. M_{k+1} = Intra-Similar (C_{k+1}, M_k);
10. $k = k + 1$;
11. **End;**
12. return C_k ;
13. **End;**

5. System Architecture and Performance Evaluation

We implemented the data mining algorithms and prefetching mechanisms to show the effectiveness of the proposed methods. A traversal path database was recorded each user's traversal path and used for mining target. The simulation model we used and the experimental results are provided in Section 5.1 and Section 5.2, respectively.

5.1 Test data and Simulation Model

We use the virtual power plant model from http://www.cs.unc.edu/~walk/ created by Walkthrough Laboratory of Department of Computer Science of University of North Carolina at Chapel Hill. The Power Plant Model is a complete model of an actual coal fired power plant. The model consists of 12,748,510 triangles. Its size is 128 MBytes. Our traversal database keeps track of the traversal of the power plant by many anonymous, randomly users. For each user, the data records list all the areas of the power plant that user visited in a one week timeframe. Each path consists of 30 ~ 40 views. Each view consists of 20~30 objects on average. The number of objects is 11, 949, where each object is a some meaningful combination of triangles of power plant and it is considered as a data item.

5.2 Experimental Results and Performance Study

In this section, the effectiveness of the proposed clustering algorithm is investigated. All algorithms were implemented in Java. The experiments were run on a PC with a AMD Athlon 1800+ and 512 megabytes main memory, running Microsoft Windows 2000 server. Our main performance metric is the *average latency*. We also measured the *client cache hit ratio*. A decrease in the average latency is an indication of how useful the proposed methods are. The average latency can decrease as a result of both increases cache hit ratio via prefetching methods and better data organization in the disk. An increase in the cache hit ratio will also decrease the number of requests sent to server and, thus, lead to both saving of the scare memory source of the server and reduction in the server load.

5.2.1 Experimental Results on Mining Unit

In this subsection, we report our experimental results on the *VSPM* algorithm. Since *GSP* and *SPADE* are the two most important sequential pattern mining algorithms, we conduct an extensive performance study to compare *VSPM* with them. To evaluate the effectiveness and efficiency of the VSPM algorithm, we performed an extensive performance study of *GSP*, *SPADE*, *FreeSpan*, and *PrefixSpan*, on real data sets, with various kinds of sizes and data distribution. Besides, these four algorithms, *GSP*, *SPADE*, *FreeSpan*, and *PrefixSpan* were implemented in Java.

5.2.2 Virtual environment traces

The performance of our traversal data base is reported as follows. First, we follow the procedure described in [4] to set up out data set parameters. The meanings of all parameters are listed in Table 1. Figure 2 shows the performance comparison among the five algorithms for our virtual environment data set. From Figure 2, we can see that *VSPM* is as efficient as *PrefixSpan* does, but it is much more efficient than *SPADE*, *FreeSpan*, and *GSP*.

Table 1. Parameters for our traversal data set

Symbol	Meaning
$\|D\|$	Number of data sequences (i.e., size of database)
$\|C\|$	Average number of transactions per data sequence
$\|T\|$	Average number of items per transaction
$\|S\|$	Average length of maximal possible frequent sequences
$\|I\|$	Average size of itemsets in maximal possible frequent sequences
$\|N_S\|$	Number of maximal potentially frequent sequences
$\|N_I\|$	Number of maximal potentially frequent itemets
$\|N\|$	Number of items

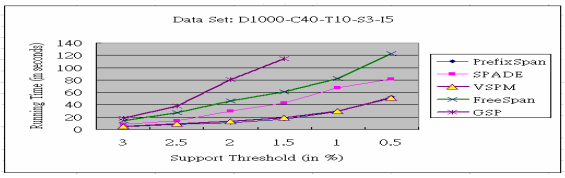


Fig.2. Execution time with respect to various support thresholds using our real data set.

5.2.3 Experimental results on clustering unit

For quality measure of clustering result, we adopted the cluster cohesion and the inter-clustering similarity. All are defined as follows.

Definition 2: *View-radius*

The v*iew radius* is defined as the radius of the visible circle in the virtual environments. As the radius increases, the more objects are observed. In other words, it controls how many objects are observed at the same time in one view.

In the meanwhile, we select the different views-radius for comparison. Figure 3 and 4 show the results. Algorithm with clustering outperforms other algorithms without clustering. Since the clustering mechanism can accurately support prefetching objects for future usage. Not only the access time is cut down but also the I/O efficiency is improved.

6. Conclusions and Future Work

In this paper, we have designed a frequent projection-based sequential pattern mining algorithm to find correlations among objects. Using the VE traces, our experiments show that *VSPM* is an efficient algorithm. Besides, we have evaluated correlation-directed prefetching and data layout. Our experimental results have shown that correlation-directed prefetching and data layout can improve I/O average response time by 35.6% to 1.249 compared to no-prefetching, and 33.3% to 2.625 compared to the number

of retrieved files. Finally, we have also designed two criteria to verify the validity of clustering method.

Our study has several limitations. One important limitation is that our disk layout was not especially designed for the extra long frequent sequential patterns. This direction will enhance the system performance.

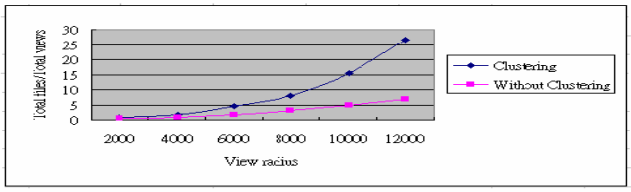


Fig. 3. Comparison of different algorithms on the number of files retrieved under the same view.

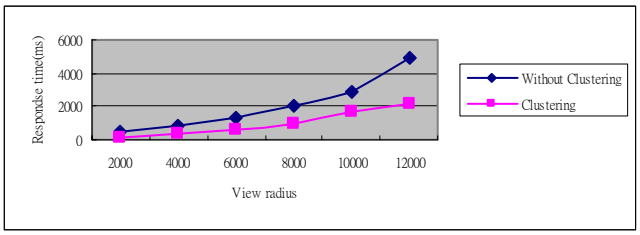


Fig. 4. Comparison of different algorithms on system response time under the same view_radius.

References

[1] S.S. Hung, T.C. Kuo, and D.S.M. Liu, "PrefixUnion: mining traversal patterns efficiently in virtual environments", *International Conference on Computational Science (ICCS-2005)*, LNCS vol. 3516, May 2005, pp. 830-834.

[2] M. S. Chen, J S. Park, and P. S. Yu, "Efficient data mining for path traversal patterns", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 10, Issue 2, 1998. pp. 209-221.

[3] R. Srikant and R. Agrawal, "Mining sequential patterns: generalizations and performance improvements", *Proceeding Fifth International Conference Extending Database Technology (EDBT'96)*, Mar, 1996, pp. 3-17.

[4] J. Pei et al, "PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth", *Proceedings of the International Conference on Data Engineering (ICDE)*, 2001, pp. 3-14.

[5] M, Zaki, "SPADE: An Efficient Algorithm for Mining Frequent Sequences", *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, Volume 25 Issue 2, Montreal, Quebec, Canada, June 1996, pp. 103-114.

[6] R. Srikant and R. Agrawal, "Mining quantitative association rules in large relational tables", *Proceedings* of the 1996 ACM *SIGMOD International Conference on Management of Data*, Volume 25 Issue 2, Montreal, Quebec, Canada, 1996, pp. 1-12.

[7] R. Agrawal and R. Srikant, "Mining sequential patterns", *11th International Conference on Data Engineering*, Volume 25, Issue 2, Montreal, Quebec, Canada, 1995, pp. 1-12.

[8] M. Sivathanu, V.Prabhakaran, F. Popovici, T.E. Denehy, A.C. Arpaci-Dusseau, and R.H. Arpaci-Dusseau, "Semantically-smart disk systems," *Proceedings of the Second USENIX Conference on File and Storage Technologies*, 2003.

[9] J. Choi, S. H. Noh, S. L. Min. and Y. Cho, "Towards applications/files-level characterization of block reference: a case for fine-grained buffer management," *Proceedings of the 2000 ACM SIGMETRICS International Conference on Measurement and Modeling of Computes*, 2000.

[10] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu, "FreeSpan: frequent pattern-projected sequential pattern mining," *Proceeding 2000 ACM SIGKDD International Conference Knowledge Discovery in Database (KDD'00),* August 2000, pp. 355-359.

[11] J. H. Clark, "Hierarchical geometric models for visible surface algorithms," *Communication of the ACM*, Vol. 19, No. 10, October, 1976, pp. 547-554.

[12] J.M. Airey, J.H. Rohlf, and J.F.P. Brooks, "Towards image realism with interactive updates in complex virtual building environments," *1990 ACM Symposium on Interactive 3D Graphics*, 24(2), Mar, 1990, pp. 41-50.

[13] L. Kaufman and Peter J. Rousseuw, *Finding Groups in Data*, Wiley, New York, 1990.

[14] L. Arge, K. Hinrichs, J. Vahrenhold, and J. Vitter, "Efficient bulk operations on dynamic R-tress," *Proceeding Workship on Algorithm Engineering*, 1999, pp. 104-128.

[15] Z. Wu and R. Leahy, "An optimal graph theoretic approach to data clustering: theory and its application to image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence* , Vol. 15, No. 13, 1993, pp. 1101-1113.

[16] S. Goil, H. Nagesh, and A. Choudhary, "MAFIA: efficient and scalable subspace clustering for very large data sets," Technical Report CPDC-TR-9906-010, Northwestern University, 2145 Sheridan Roas, Evanston IL 60208, June, 1999.

[17] A. Joshi and R. Krishnapuram, "On mining web access logs," *Proceeding SIGKDD Workship Research Issues on data mining and Knowledge Discovery(DMKD)*, 2000.

[18] Zhu, Y, "Uniform remeshing with an adaptive domain: a new scheme for view-dependent level-of-detail rendering of meshes," IEEE Transactions on Visualization and Computer Graphics, Vol. 11, Issue 3, May-June, 2005, pp. 306-316.

[19] E. Ohbuchi, "A Real-Time refraction renderer for volume objects using a polygon-rendering scheme," Proceeding of Computer Graphics, 2003, pp. 190-195.

[20] P. Jaccard, "The distribution of the Flora of the Alpine Zone," *New Phytologist*, 1912, pp. 37-50.

An Architecture based on multi-agent system and data mining for recommending research papers and researchers

Sílvio César Cazella[1,2], Luis Otávio Campos Alvares[2]
[1] Centro de Ciências Exatas e Tecnológicas, Universidade do Vale do Rio dos Sinos,
Av. Unisinos 950, CEP 93.022-000, São Leopoldo, RS, Brazil
[2] Instituto de Informática - Universidade Federal do Rio Grande do Sul
Av. Bento Gonçalves, 9500, Porto Alegre, Brazil
{cazella, alvares}@inf.ufrgs.br

Abstract *Looking for a useful material on the Internet sometimes is a very difficult task due to the great quantity of information available on it. Finding the right material and maybe the right person on the Internet could be a worthwhile result. In this paper, an architecture based on multi-agent system and data mining for recommending research papers and researchers is presented. Through this system we intend to create virtual web communities to facilitate academic people sharing common interests. We address the problem of creating virtual academic web communities for researchers in Brazil, taking into consideration information from a hybrid recommender system and the Brazilian e-government's system named CV-Lattes.*

Keywords: data mining, web-based tools, recommender system, multi-agent system.

1. Introduction

In academic life, sharing of experience is a basic principle, because of the need to acquire a huge quantity of information in a limited period of time. When a student begins working on a research project, as either an undergraduate or graduate student, he needs to find relevant information for this work. One important question is: how can he best find the relevant information? We assert that a recommender system can provide a solution for a student because it identifies relevant items and offer it to him. This approach is totally different from information retrieval because the effort to find information does not concentrate on a query provided by a user. This kind of system, based on some criteria, finds information that might be useful to a user, and then recommends it. It is also a useful technology in creating virtual web communities, because it permits the identification of similarities between users (like-minded) [9][12]. But a student would be able to obtain much more information about specific items if he was able to contact the relevant people from his area of interest, and exchange ideas. Here, we present a system, which recommends items and identify a concise community to include a student. In this way, he is able to take advantage of the community's experience, knowledge, and expertise about his subject area of interest. In this paper, we address the problem of creating virtual academic web communities using a recommender system, multi-agent system, data mining and the Brazilian e-government's[1] system named Curriculum Vitae Lattes.

1.1 Contributions of this paper

The contribution of this paper is to provide a complete architecture based on recommender system, multi-agent system, and data mining to provide personalized recommendations and identify virtual academic web communities. We also present a detailed description of the agents included in the architecture.

1.2 Related Work

A recommender system [1] presents a new approach to aiding the user in finding relevant information. Almost all recommender systems do the same thing: they identify items to offer to the user based on some criteria.

Some techniques, such as collaborative filtering [3][13] and content-based filtering [3][8], are applicable in recommender systems. Another technique is known as the hybrid technique, which applies collaborative filtering and content-based filtering, together [3].

Some recommender systems can be listed as example: REFERRAL WEB [8], which identifies, automatically, social networks (such as a community). In this system, the construction of the net is based on citations. RINGO [13]

1 In this paper we assume e-government to be an emergent concept. It refers to systems, which provide free information through an electronic resource. The resource is provided from a public government agency to citizens.

was developed to recommend music. This system explores similarities among likes from many different users in order to recommend items, but does not present information about the relevance of people who evaluate the items. GROUPLENS [9] recommends news to users by applying collaborative filtering. In this process, the identification of like-minded users is very important. FAB [3] applies a hybrid approach; it works by applying collaborative filtering beyond the content-based filtering to identify like-minded people, and then recommend items. SOAP [14] is a multi-agent system that recommends web sites. User, search, and recommender agents communicate to achieve recommendation for multiple users. It is possible to find information about many recommender systems on Adomavicius [1] and Schafer [12] work.

1.3 Outline of Paper

The remainder of the paper is organized as follows: Section 2 gives an overview of the virtual web communities' concept. Section 3 introduces the Recommenders' Rank metric and CV-Lattes system. Section 4 presents the architecture of W-RECMAS and describes the agents. Finally, Section 5 summarizes the research, and presents some directions for future work.

2. Virtual Web Communities

This section describes the concept of virtual web communities adopted in this work. The term community usually refers to a location where people with common interests gather to share experiences, ask questions, and collaborate [4] (e.g., groups.yahoo.com, Friendster.com, Friendzy.com, Tribe.net and orkut.com). A community is a useful place for people to learn from each other's experiences and knowledge, and share successful and unsuccessful cases.

The first forms of virtual communities were created based on users who were able to meet and discuss over distances. According to Rheingold[11], a virtual community represents the emergence of socially motivated communities of interest on the Internet. The authors describe virtual communities as "social aggregations that emerge from the net when enough people carry on those public discussions long enough, to form webs of personal relationship in cyberspace".

A community provides an excellent source of knowledge and information and, most important, helps to avoid duplication of current and past efforts of members. For example, we can imagine a case in which a student (e.g., a graduate student), beginning some research, may waste a considerable amount of time searching for information that others had previously discovered in their work. However, if the student were enrolled in a community, this would provide them with some shortcuts.

In order to create a community, we need to address some challenges [4]: a) members should be directed to like-minded people and relevant information; b) members should have access to the required information without feeling overloaded; c) members should be informed about other like-minded members; d) members should be able to disseminate information to the appropriate people.

We assert that by applying data mining, recommender systems, a multi-agent system, and a metric named Recommenders' Rank (RR) to deal with the issue of authority (see Section 3), we can supply enough resources to deal with all these challenges.

3. Recommenders' Rank and CV- Lattes System

The user's profile is a fundamental element in the recommendation process [10]. To define a user's profile in a recommender system, we need information from some sources. W-REMAS (Web-RECommender system based on Multi-Agent System for academic paper recommendation) system applies information from two sources: an electronic form and CV-Lattes system[2].

The first time a user access the W-RECMAS, he must register, choosing a username and a password. After that, he is required to complete an e-form to provide information such as full name, e-mail address, university name, home page address, academic level, areas of interest, and level of knowledge in a specific area (beginner, basic or deep). Once we have the user's full name, a Crawler Agent (see Section 4.1) will obtain more information about this user from the Internet, accessing a Brazilian e-government system. This agent retrieves information from a system named CV-Lattes, which was developed by CNPq[3], Ministry of Science and Technology, and other organizations in Brazil.

In this system's database, it is possible to find information about all the communities involved in research (e.g., graduate students, teachers, and professors). From this database, we use both qualitative data, i.e., areas of interest, and some quantitative data, such as production metrics (e.g., number of publications in journals, papers published in conferences, books and chapters published, students supervised, and so on). All this information is used to create the final user's profile and calculate a metric named "Recommender's Rank" (RR) to represent the authority of a user to evaluate items in a specific area of knowledge [7]. The reason for this metric is the necessity of providing to the target user some further information to explain the relevance

[2] http://lattes.cnpq.br/

3 CNPq (The National Council for Scientific and Technological Development) in Brazil is a foundation linked to the Ministry of Science and Technology (MCT), to support Brazilian research.

of a recommendation. In these work we assert that experts have more conditions to analyze items than not expert people (in some specific domains of knowledge [4][7]). For example, a RR equal to zero means that a user does not have much experience with a research area (we assume this user does not have so much authority to evaluate items for this area); five means that the user has a good deal of experience in a research area; and ten means that this user has in-depth knowledge about a research area (we assume that this user's opinion is very relevant because of his authority in the area). Each user will have an RR_{area} for each area of interest (each area is related to a set of papers about computer science in W-RECMAS). There is also an RR_{total}, which is an average among all RR_{areas} (in which the user presents opinion relevance) of a specific user. More information about this metric can be found in [5][7].

4. W-RECMAS' Architecture

An overview of W-RECMAS' architecture is shown in Figure 1. The system functionalities are supported by five types of software agents: Crawler Agent, Analyst Agent, Personal Agent, Recommender Agent and Community Agent. In this work, an agent is an autonomous and independent entity, which has a specific goal and the ability to do a specific task in a computational environment. These agents have the following properties [15]: autonomy; social ability; continuity; communicability and rationality.

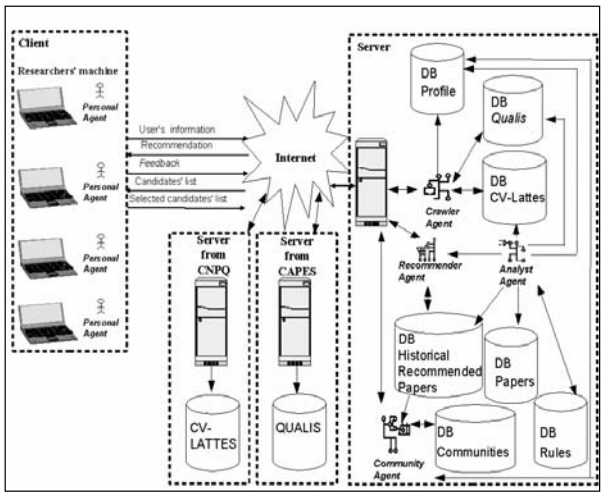


Fig. 1. Overview of W-RECMAS' architecture

According to figure 1, when a user access the system and gives his complete name and CV-Lattes Key (a personal code), this information will enable a Crawler Agent to search for more information about the user in the CV-Lattes system. Using the information provided by the user and by the Crawler Agent, the Analyst Agent calculates the Recommender's Rank (RR) by area of interest for this user, and a complete profile is created. The Analyst Agent tries to predict an item (in this case an item is an academic paper) for the users based on their profile, papers content, historical consumption, and ratings, according with a specific filtering technique. Once we have a recommendation a Recommender Agent delivers it to a Personal Agent (this agent will be alive in the user's machine observing the user and waiting for Recommender Agent contact). The Personal Agent reminds the user to send a recommendation feedback to the system, so it will improve future recommendations. A Community Agent verifies the user's information to find potential members for a community. These agents use ACL specifications from FIPA[4] (Foundation for Intelligent Physical Agents) to exchange information. Compared to the systems in Section 2, the system W-RECMAS presents some advantages such as: a) it takes into account the authority of people who evaluate the items [7]; b) the user's profile is not a static object, meaning it can evolve due to the system capability to predict a new user's areas of interests automatically, based on the opinion of relevant users (user with high authority level) [5]; c) it can identify and propose communities without direct user intervention, it can be done just through the use of the system [6]. The following subsections present a description of the agents and their functionalities.

4.1 Crawler Agent

The agent's goal is to supply information about users to other agents and to monitor the user's CV-Lattes. Once the agent receives a user's full name and CV-Lattes Key from an e-form, it verifies whether this really is a new user, submits this information to the CV-Lattes system, and then recovers the user's entire curriculum vitae. This agent sends a message to the Analyst Agent, in the case of a new CV-Lattes being successfully downloaded to Raw CV-Lattes database (the word Raw here is used to define a content not analyzed yet). Figure 2 shows a view of the Crawler Agent's internal architecture and interfaces. The Crawler Agent monitors the CV-Lattes system periodically to verify any possible updates to a user's CV-Lattes. If an update has occurred, the agent recovers the user's CV-Lattes and sends a message to the Analyst Agent, which updates the user's profile.

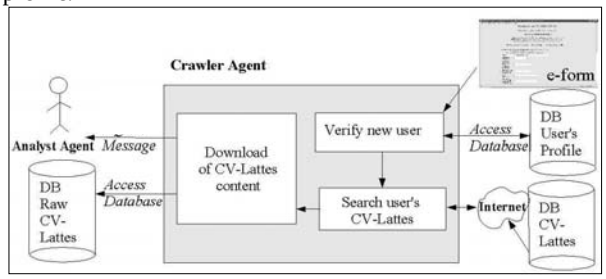


Fig. 2. Internal Architecture of Crawler Agent

[4] http://www.fipa.org

4.2 Analyst Agent

The Analyst Agent is the most complex agent in the architecture because it has a mix of responsibilities:

a) Parse the user's CV-Lattes: When an Analyst Agent receives a message from the Crawler Agent requesting analysis of a new user's CV-Lattes, the Analyst Agent begins parsing the CV-Lattes' content, wraps it in a different object, and records it in the user's profile database.

b) Calculate the Recommender's Rank metric: The Analyst Agent calculates the relevance of a user's opinion (authority) for the user's chosen areas of interest. For example, suppose that we have a user who is a Ph.D. and the individual's area of interest is "Data Mining"; he has seven publications in journals, thirty-two other types of papers and eight chapters published in books; twenty-three students supervised; participation in twenty-one examining committees; ten students currently being advised. His RR_{area} for this area of interest could be 7.9, meaning that, if this user helps in any recommendation, the user's opinion relevance will be equal to 7.9 (on a scale from 0 to 10), a high relevance.

c) Find the best match between the user's profile and available papers: In order to accomplish this, the Analyst Agent needs to decide which filtering technique is best for each user at a specific moment. The system is hybrid, i.e., it can use collaborative or content-based filtering. In the first type, we have a domain of items to be recommended and the user can give some opinions about the items (usually ratings). This kind of technique proposes some personalization by exploiting similarities and dissimilarities among users' preferences. The system compares the users' ratings and finds "similar" (like-minded) users. Based on criteria and similarities, it will then recommend items in the future. In the other hand, Content-based filtering attempts to analyze the content of an item and not the ratings.

d) Predict new user's areas of interest based on a collaborative approach: Here, the Analyst Agent applies the association rule [2] technique to discover new knowledge to improve the list of user's interests, based on others' opinions (word of mouth [13]). The main idea of this approach is to avoid the over-specialization problem [3]. First of all we extract and transform user's areas of interest and create a matrix. In this matrix each column represents an area of interest and each line represents a user (UxA) and a value equal one (1) is attributed if the user is interested in a specific area or zero (0) if not [5][6]. This content is submitted to the mining algorithm. We use rules to describe the new knowledge discovered because it is an intuitive way to represent the knowledge.

For example, applying a specific support and confidence threshold the rule R1 could be find – R1:{Data Mining} → {Machine Learning}. This rule suggests that most researchers in our database who are interested in papers related to the "Data Mining" area also tend to be interested in papers related to the "Machine Learning" area. If this rule is considered useful, then it can be applied as new knowledge to predict a new user's interest. In this work the original Apriori [2] algorithm is used.

In order to confirm if the rule has value, beyond confidence and support, we are applying an average among the values of the RR_{total} of users who match totally with the new rule. If the result is greater than or equal to a threshold, we consider this rule applicable for the others. Once the rule is new, and the relevance is greater than or equal to a threshold, the Analyst Agent will find the users who match with the left hand side of the rule, and match with the opposite of the right hand side of the rule, to start the recommendations according to this new rule of interest. If the user indicates interest in this new area (according to the user's feedback), the user's profile will be updated. Figure 3 presents a view of the Analyst Agent's internal architecture and its interfaces.

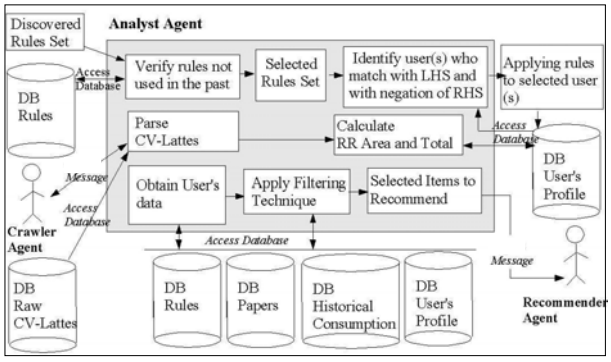


Fig. 3. Internal Architecture of Analyst Agent

4.3 Recommender Agent

This agent was designed to communicate the list of recommendations to a Personal Agent, and it also prepares an explanation about the recommendations' relevance to the Personal Agent owner. For example, the user may receive the following message – "Paper: Predicting new user's interest based on data mining and user's relevance opinion, is a good choice for you because, based on your profile, it matches with your interests, and beyond that, a user from your group with Recommender's Rank equal to 9.7, strongly recommends this paper".

The Recommender Agent receives a list of recommendations to a specific user from the Analyst Agent. From this list, it selects three items based on the Recommender's Rank. The rest of the items not already recommended will be saved in a database of Historical Consumption, with a flag with value

equal to FR (future recommendation). Other possible flag values are RF (recommendation with feedback) and RE (recommendation without feedback). The Recommender Agent receives the user's feedback from the Personal Agent (it could be explicit or implicit feedback) and attributes this to the respective recommendation provided for this user. Figure 4 presents a view of the Recommender Agent's internal architecture and its interfaces.

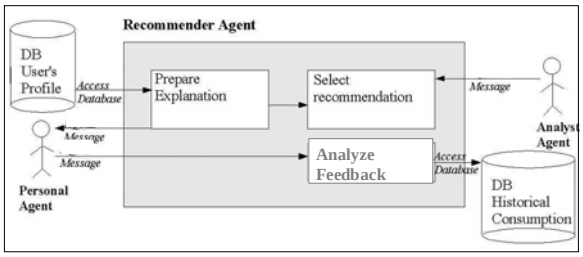


Fig. 4. Internal Architecture of Recommender Agent

4.4 Personal Agent

This agent is located in the user's machine and its core functionalities are to present the paper recommendation(s) to the user (using push technology [12]), and return feedback about the recommended paper to the system (it can be a user's feedback or an agent's feedback).

The recommendation feedback is essential for a recommender system to work. The Personal Agent takes the user's implicit and explicit feedback and sends it to the system, where it will be used in the collaborative-filtering technique to find like-minded users. If the user gives explicit feedback about the recommendation (using a Likert scale from 1 to 5), the agent sends this rating to the system. However, when the user does not give feedback concerning the recommended paper, the agent needs to apply some rules to predict possible feedback. Figure 5 presents a view of the Recommender Agent's internal architecture and its interfaces.

This agent interacts with the Recommender Agent, from which it receives a list with all selected papers to be recommended to the user and the reason for the recommendation (explanation).

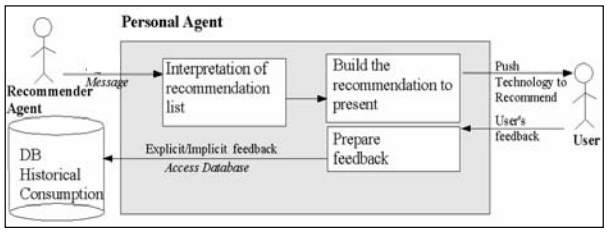


Fig. 5. Internal Architecture of Personal Agent

4.5 Community Agent

Another concern in our work is related to the existence of users that other users would like to get in contact. Here we don't work with explicit trust relation but implicit trust relation. It means that our system is able to identify this kind of relation and then tries to put these people in contact.

Aiming to establish the virtual communities the Community Agent was designed; it identifies potential (candidates) users for the community, and attempts to put the users in contact, thereby creating an on-line community. In order to create a community, the agent address the challenges already cited in section 2: a) members should be directed to like-minded people and relevant information; b) members should have access to information they require, without feeling overloaded; c) members should be informed about other like-minded members; d) members should be able to disseminate information to the appropriate people. In the next paragraph we define some sets and explain the complete rule applied by the agent to define candidates for a community.

Let $U = \{u_0,\ldots,u_{n-1}\}$ be a set of n users, let $AI=\{a_0,\ldots,a_{m-1}\}$ be a set of m areas of possible interest. Let C be a community where $C \subseteq U$. The agent consider users as candidates for a new community if the following rule is satisfied:

Iff $((related\_area(u_n,AI) \cap related\_area(U_{n+1}, AI)) <> \varnothing)$ *and*
$like\_minded(U_n,U_{n+1}) = true$ *and*
$trustworthy(U_n, U_{n+1}) = true$
then candidates

Applying this rule we assume users as candidates for a new community if and only if: a) the users have at least one area of interest in common; b) they are like-minded (like-minded people are those people who like things in the same way, it means they show the same likes and to determine this value we apply the well know nearest neighborhood algorithm [9][13]; and c) one user trust in other user.

In this work we assume that if a user receive a recommendation from a group of people (like-minded) and the target user gave a positive feedback for at least three recommendations, there is a trustworthy relationship. In order to put the users of the potential community in contact, the agent cannot simply send the data from each user to the others, because the users' data is private information. For this reason, the Community Agent starts a communication with the users' Personal Agents. This communication is applied to verify if one user wants to get in contact with others. For example, $User_1$'s Personal Agent receives as information a list of users whose could participate in the possible community, and each user is designated by the RR_{area} for each common area of interest and RR_{total} and the academic level. The Personal Agent presents the list of possible users to be included in the community, and requests to his owner to select which users from the list he would like to get in

contact with, to exchange ideas. At the same time the Personal Agents from the others members of the same list, perform the same task. Once the Community Agent has received answers from all users' Personal Agents about the possible community, it analyzes the answers.

For example, $User_1$ will get in contact with $User_2$, if and only if $User_1$ shows explicit interest in contact with $User_2$, and $User_2$ shows the same interest. Otherwise, they will not get in contact. After the verification and a merge among all answers have been done, the Community Agent provides the contact e-mail to the users to enable them to get in contact. A user can be included in many different communities. Figure 6 presents a view of the Community Agent's internal architecture and its interfaces.

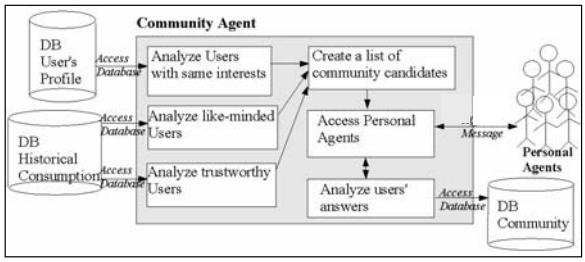


Fig. 6. Internal Architecture of Community Agent

5. Conclusions and Future Work

In this paper we presented a complete definition of an architecture based on multi-agent system and data mining to recommend research papers and researchers. The main idea in this work is to create conditions to people easily receive recommendation of relevant material and get in contact with relevant people to exchange ideas. The three principal contributions presented in this paper are: (1) a detailed architecture to identify and recommend academic communities and papers; (2) support to work with multi-agent system and data mining; (3) how to deal with the relevance of users' opinion (authority).

Our prototype was built using technologies as Java (java.sun.com), PHP (www.php.net), Appache web server (httpd.apache.org), Perl (www.perl.org), Postgress database and some scripts. Currently, our system is being placed on line, to be used in a controlled experiment, with researchers.

Our future work includes studies about: a) how to deal with the user's reputation inside a specific community – in the present work the RR metric is only impacted by the CV-Lattes updates, but it could be impacted by the others opinion about recommendations (user's reputation); b) how to deal with the knowledge generated by the data mining task to update correctly the user's profiles.

Acknowledgements

Our sincere appreciation to all who have helped with this paper revision. This work was developed with partial funding from the Brazilian agency CAPES under grant BEX1357/03-4.

References

[1]Adomavicius, G.; Tuzhilin, A. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Transactions on Knowledge and Data Engineering, Vol. 17, n. 6, USA, (2005), pp.734-749.

[2]Agrawal, R. et al. Fast algorithms for mining association rules in large databases. In: International Conference on Very Large Databases. Santiago, Chile, (1994) , pp. 478-499.

[3]Balabanovic, M.; Shoham, Y. Fab: Content-Based, Collaborative Recommendation. Communications of the ACM, Vol. 40, n.3, New York, USA, (1997), pp. 66-72.

[4]Case, S. et al. Enhancing E-Communities with agent-based systems. IEEE Computer Society Press ,Vol. 34, n.7, Los Alamitos, CA, USA, (2001), pp. 64-69.

[5]Cazella, S. C.; Alvares, L. O. C. Combining Data Mining Technique and Users' Relevance Opinion to Build an Efficient Recommender System. Revista da Tecnologia da Informação, UCB, Vol. 4, n. 2, Brazil, (2005), pp. 09-20.

[6]Cazella, S. C.; Alvares, L. O. C. Creating Academic Web Communities: A Recommender System to Aid Brazilian Researchers to Find Information and Contact Relevant People. In: Proceedings of the 6th Annual Global Information Technology Management World Conference, Alaska, USA, (2005), pp.45-48.

[7]Cazella, S. C.; Alvares, L. O. C. Modeling user's opinion relevance to recommending research papers. In: Tenth International Conference. Springer's LNAI, 2005. L. Ardissono, P. Brna, and A. Mitrovic (Eds.): UM 2005, LNAI 3538, Scotland, (2005), pp. 337 – 341.

[8]Kautz, H. et al. Referral Web: combining social networks and Collaborative filtering. Communications of the ACM, Vol. 40, n.3, New York, USA, (1997), pp. 63-65.

[9]Konstan, J.A. et al. Grouplens: Applying Collaborative Filtering to Usenet News. Communications of the ACM, Vol. 40, n.3, New York, USA, (1997), pp. 77-87.

[10]Pazzani, M. A. Framework for collaborative, content-based and demographic filtering. Artificial Intelligence Review, Vol. 13, Issue 5. Kluwer Academic Publishers, Hingham, MA, USA, (1999), pp. 393-408.

[11]Rheingold, H. The virtual community: homesteading on the electronic frontier. MIT Press, Massachusetts, USA, (2000).

[12]Schafer, J.B. et al. E-Commerce recommendation applications. Data Mining and Knowledge Discovery, Vol. 5, Issue 1-2. Kluwer Academic Publishers, Hingham, MA, USA, (2001), pp. 115-153.

[13]Shardanand, U.; Maes, P. Social information filtering: Algorithms for automating "word of mouth". In: Proceedings of Human Factors in Computing Systems, Denver, Colorado, USA, (1995), pp. 210-217.

[14]Voss, A., Kreifelts, T. SOAP: Social Agents Providing People with Useful Information. In: Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work. Phoenix, Arizona, USA, (1997), pp. 291-298.

[15]Weiss, G. (ed.):Multi-agent Systems: a modern approach to distributed artificial intelligence. Cambridge, MA: MIT Press, (1999).

Salient Phrases-based Clustering and Ranking in Chinese Bulletin Board System

Xiaoyuan Wu, Shen Huang, Yong Yu
Dept. of Computer Science and Engineering Shanghai JiaoTong University, Shanghai, China
{wuxy, huangshen, yyu}@sjtu.edu.cn

Abstract. Nowadays, Chinese Bulletin Board System (BBS) has been expanded at an incredible speed. Many Chinese people have great excitement for participating in the activities in BBS, like Question/Answer and opinion discussion. However, many poor-quality messages flood BBS portals, and users will feel horrible when they want to find valuable information from thousands of messages. In this paper, we propose a Salient Phrases-based Clustering and Ranking (*SPCR*) system to facilitate users to quickly understand main topics of a collection of messages. Our method tries to automatically extract salient phrases from corpus to form a hierarchical summarization and group relevant threads[1] into clusters. Thus, users could recognize the main topics of a corpus in a glance and find their interested information quickly. Our method's feasibility and effectiveness are verified by several experiments on a Chinese BBS dataset.

1. Introduction

Bulletin Board System (BBS) is a kind of web virtual space where people can discuss anything freely. Especially in China, people always have great excitement for participating in the activities in BBS. According to the statistics published by China Internet Network Information Center in 2004, the scale of virtual community like BBS in China has expanded to 27.6% of the whole Chinese Web[2].

Every BBS portal has many different topic-based boards, for example, "Java", "Artificial Intelligence", "News", "Virus", etc. However, there are still too many messages in one board. Thus, users may feel difficulty to find valuable information from such a large corpus.

In this paper, we propose a novel method to help users get quick understanding of the main topics and sub-topics of a large amount of messages. We extract salient phrases to form a hierarchical summarization and cluster relevant threads into clusters. By our system, users will find their interested information quickly, without reading messages one by one. First, we extract parent salient phrases from the whole corpus. Each salient phrase could represent a topic, For instance, we extract "防火墙"(firewall), "诺顿" (Norton), etc from a dataset about "computer virus". Several properties of phrases and regression models are utilized to determine whether a phrase is salient or not. Second, in order to get more specific concepts and form a hierarchical summarization for quick browsing, we continue to extract child salient phrases of each parent salient phrase. For example, we extract "system patch" and "windows", etc as the child salient phrases of "system". Child phrases are extracted by a mixture model of subsuming probability and co-occurrence. Third, corresponding threads are labelled with relevant salient phrases and grouped into several clusters. In each cluster, the threads are ranked by their relevance to the salient phrases. We call the whole approach Salient Phrases-based Clustering and Ranking (*SPCR*). Finally, we provide users a visual interface to find the right messages quickly. In the left side of Figure 1, both parent and child salient phrases are listed. After users choosing a salient phrase they are interested in, the corresponding cluster of threads will be shown in the right panel. Thus, users could recognize the main topics in a glance and find the relevant threads quickly.

Figure 1. Visual interface of *SPCR*.

The rest of the paper is organized as follows: Some related work is introduced in Section 2. In section 3, we

[1] The root of the thread is the first message posted by someone to seek an answer or to initialize a discussion regarding a specific topic. Then the thread expands as other people reply to this message and continue the discussion.
[2] http://tech.sina.com.cn/focus/04_net_rep/index.shtml

explain our methods of salient term extraction and thread ranking. Section 4 describes the experimental results of our methods. Finally, we conclude our paper in Section 5.

2. Related Work

BBS and newsgroups gradually play a significant role in the Web, but there were only some limited amount of academic research in last few years. [1][3] proposed methods to assess authors of the messages by their historical activities in Newsgroup. Xi et al. [16] presented an approach to learn effective ranking function in newsgroup search.

The problem of keyword extraction (salient phrase) has been investigated in a number of studies, e.g. [11][12][14][15]. Most of them focused on extracting keywords from a single document, while our task is to extract keywords from a corpus.

Some topic finding studies (e.g. [13][8]) are related to our method. Sanderson and Croft [13] built concept hierarchies by finding pairs of concepts (x, y) in which x subsumes y. Lawrie and Croft [8] used the Dominating Set Problem for graphs to choose topic phrases by considering their relation to the rest of the vocabulary used in the document set. Our method further calculate several important properties to identify salient phrases

Some previous studies about clustering search results have the similar purpose with our method. In order to facilitate users' quick browsing and understanding, some recent work in [4][5][7], also presented methods to organize on-the-fly the search results into a hierarchy of labelled folds. The work in [17] extracted salient words from web search results, and then clustered results using these salient words. Our method differs from those previous studies. We utilize more properties to calculate salient term scores. Furthermore, we propose an effective method to rank threads of each cluster to get the most relevant information.

3. The Algorithms of *SPCR*

The goal of our proposed method is to help users to get information efficiently and conveniently. Our technique is mainly composed of four steps:

- Parent salient phrase extraction and ranking,
- Child salient phrase extraction and ranking,
- Grouping synonyms,
- Thread clustering and ranking.

In this section, we will discuss these steps in detail.

3.1. Parent Salient Phrase Extraction

The purpose of this step is to identify salient phrases from an initial set of messages. First, we list some particular properties of phrases in Chinese BBS messages. In addition, we briefly introduce how to use regression models to calculate the salient score of each phrase.

- **Property Extraction**

We list the four properties calculated during messages pre-processing. These properties are supposed to be relative to the salient scores of phrases.

TFIDF

Intuitively, more frequent phrases are more likely to be better candidates of salient phrases. The motivation for usage of an IDF factor is that phrases which appear in many messages are not useful for representing a sub-topic.

$$TFIDF = f(t) \cdot \log \frac{N}{|D(t)|} \quad (1)$$

where $f(t)$ denotes the frequency of phrase t and $D(t)$ represents the set of messages that contain phrase t. Besides, we denote N as the total number of messages.

Phrase Frequency in Message Title

When users post a message to BBS, they usually describe their questions or opinions briefly in titles. Thus, we calculate the phrase frequency in title (denoted by *TITLE*).

Average of Phrase's First Occurrence in Message

First occurrence is calculated as the number of phrases that precede the phrase's first appearance, divided by the number of phrases in a message. The result is a number between 0 and 1 that represents how much of the message precedes the phrase's first appearance. Salient phrases are assumed to be appeared in the front of messages. We use *ATFO* to denote this property.

$$ATFO = 1 - \frac{1}{|D(t)|} \cdot \sum_{d \in D(t)} \frac{p(td)}{\#d} \quad (2)$$

where $p(td)$ denotes the number of phrases that precede the phrase t's first appearance in message d and $\#d$ represents the number of phrases in message d. Phrases with higher *ATFO* value are preferred.

Message Depth in a Thread

It is a common sense that there are many topic drift phenomena in BBS portals. As the message number in a thread becomes larger, the topic is more likely to drift away from the original one. It is assumed that phrases appearing in first several messages of a thread should be less influenced by noisy data. The average value of message depth a term occurs in can be calculated by,

$$DEPTH_t = \frac{1}{f(t)} \sum_{d \in D(t)} [Depth(d) \cdot f(td)] \quad (3)$$

where $f(td)$ denotes the frequency of phrase t in message d. If a message d is the root of a thread, then $Depth(d) = 1$.

- **Learning to Rank Salient Phrases**

Given the above four properties, we could use a single formula to combine them and calculate a single salience score for each term. Thus, we utilize training data to learn a regression model. In this paper, we use linear regression and support vector regression to complete this task. These two models are both widely used in regression.

Linear Regression: The relationship between two variables is modeled by fitting a linear equation to observed data. The linear regression model postulates that:

$$y = b_0 + \sum_{j=1}^{p} b_j x_j + e \quad (4)$$

In our case, independent variable x = (*TFIDF*, *TITLE*, *ATFO*, *DEPTH*), and dependent y can be any real-valued score. We use y to rank salient terms in a descending order, thus the salient terms could be ranked by their importance.

Support Vector Regression: In support vector regression, the input x is first mapped onto a high dimensional feature space using some nonlinear mapping, and then a linear model is constructed in this feature space. Support vector regression uses a new type of loss function called ε-insensitive loss function:

$$L_\varepsilon(y, f(x,\omega)) = \begin{cases} 0 & \text{if} |y - f(x,\omega)| \le \varepsilon \\ |y - f(x,\omega)| - \varepsilon & \text{otherwise} \end{cases} \quad (5)$$

Support vector regression tries to minimize $\|\omega\|^2$. This can be described by introducing (non-negative) slack variables ξ_i, ξ_i^*, i=1,..., n, to measure the deviation of training samples outside ε-insensitive zone. Thus support vector regression is formalized as minimization of the following functional:

$$\min \frac{1}{2} \| \omega \|^2 + C \sum_{i=1}^{n} \xi_i + C \sum_{i=1}^{n} \xi_i^* \quad (6)$$

$$\begin{aligned} & y_i - f(x_i, \omega) \le \varepsilon + \xi_i^* \\ & f(x_i - \omega) - y_i \le \varepsilon + \xi_i \\ & \xi_i^*, \xi_i \ge 0, i = 1, ..., n \end{aligned} \quad (7)$$

3.2. Finding Child Salient Phrases

We hope to find child salient phrases which can facilitate users to understand more specific sub-topics. In order to extract child salient phrases for each parent phrase, we introduce a method presented in [13]. It is defined as follows, for two phrase, x and y, x is said to subsume y if the following conditions hold, $P(x|y) \geqslant 0.8$, $P(y|x) < 1$. In other words, x subsumes y if the documents which y occurs in are a subset of the documents which x occurs in. In the hierarchy of phrases, x is the parent of y.

We calculate $P(x|y)$ to get top n phrases with high probabilities as the child salient phrases of x. However, the result is not satisfied. The phrases ranked high by this method are truly subsumed by parent phrases, but some of those phrases are non-informative to users. For example, some rarely appeared phrases are subsumed by parent phrases, but they can not represent sub-topics. Therefore, even if the value of $P(x|y)$ equals 1, some phrases are still not good candidates for child salient phrase.

To solve this problem, we extract child salient phrases from the neighbour phrases of parent salient phrases. For example, "configure" often occurs nearby "firewall", thus "firewall configure" seems to be a good child phrase and represent an aspect of "firewall". One advantage that we restrict the child salient phrase candidates in the range of neighbour phrases is to reduce noise in results. The simple heuristic seems to work well in practice. Therefore we use following two properties together to determine the score of each candidate phrase.

1. $P(x|y)$, the probability of x in the condition of y occurs
2. Frequency of phrase x and y's co-occurrence in a small size window.

Thus, we use a linear combination equation to determine the score of each child phrase. The value of $P(x|y)$ and CoOccurrence are both normalized.

$$ChildScore = \lambda \times P(x|y) + (1-\lambda) \times CoOccurrence \quad (8)$$

where y is a phrase in neighbourhood of parent phrase x. If we hope to find more levels of salient phrases, we can run this method recursively.

3.3. Grouping Synonyms

It is common that people use different words to describe the same concept. In our Chinese BBS dataset, we found three basic types of synonyms. First, many users often use abbreviation like "卡巴" for "卡巴斯基" (Kaspersky). Second, miss spelling, like "特洛伊"(Trojan) and "特洛依". Finally, some regular synonyms, such as "配置" and "设置" which both mean "configure". To our best knowledge, no similar tools like WordNet [2] are available to identify Chinese synonyms.

To solve this problem, we use Levenshtein Distance (LD) algorithm [9] and mutual information to group Chinese synonyms.

Levenshtein Distance

Levenshtein Distance is a measure of the similarity between two strings. The distance is the number of deletions, insertions, or substitutions required to transform into. For example, if S_i is "特洛伊" and S_j is "特洛依", then $LD(S_i, S_j) = 1$, because one substitution (change "伊" to "依") is needed. The smaller the Levenshtein Distance is, the more similar the strings are. This can be applied to the identification of Chinese synonyms.

Mutual Information

The Mutual Information of two random variables is a quantity that measures the independence of the two variables, as in equation (9),

$$I(S_i, S_j) = \log \frac{P(S_i, S_j)}{P(S_j)P(S_j)} \quad (9)$$

$P(S_i, S_j)$ is the joint probability of the term S_i and S_j. Its maximally likelihood estimator is n_{ij}/N, where n_{ij} is the number of messages involving both S_i and S_j, and N is the number of total messages. $P(S_i)$ and $P(S_j)$ are probabilities

of the terms S_i and S_j, which can be estimated as n_i/N and n_j/N respectively. If S_i and S_j are independent, $I(S_i,S_j)$ is zero. However, if S_i and S_j often co-occur in the same messages, $I(S_i,S_j)$ will turn out to be high.

Finally, the above two measures are combined together to judge whether two clusters should be merged.

$$SM_{ij} = \beta \cdot \frac{Len_i + Len_j}{2LD_{ij}} + (1-\beta) \cdot I(S_i, S_j) \quad (10)$$

where Len_i and Len_j are the length of S_i, S_j. According to our previous work in [10], β is set to 0.6. In addition, by trial and error, S_i and S_j are considered as synonyms if $SM_{ij} \geqslant 3$

3.4. Clustering and Ranking Threads

After extracting salient phrases, we let messages be automatically clustered according to the salient phrases. Since there are many Q/A, and opinion discussions existed in newsgroups, we assume that context in threads will help users to get more information. Thus we put a thread into a cluster when any message of this thread belongs to that cluster. In the visual interface in Figure 1, when a user selects a salient phrase, corresponding threads in this cluster are shown. Users could read the whole threads containing the salient phrase rather than several separated messages. In addition, we need to rank threads in clusters, and show the threads that are most relevant to the salient phrase. Two factors are considered here to determine relevant scores of threads.

Phrase Frequency in Thread

Obviously, if the salient phrase appears more frequent in one thread, the thread is more likely to be relevant.

$$P(t \mid TD) = \frac{tf(t,TD)}{TL} \quad (11)$$

where $tf(t, TD)$ is the raw phrase frequency of phrase t in thread TD and TL is the total number of phrases in thread TD.

Distribution of Salient Phrase in Thread

As we mentioned above, a thread is grouped into a cluster when any message of this thread contains corresponding salient phrases. Thus, maybe only one or two messages in a thread are relevant to the salient phrase while the others talk about something else. Intuitively, a thread should be judged more relevant, if the salient phrases appear in more messages of this thread. We use entropy to measure the distribution of salient phrases in thread. The thread with high entropy is preferred here.

$$Distribution = -\sum_{i=1}^{n} \frac{TF_i}{TF} \times \log \frac{TF_i}{TF} \quad (12)$$

where TF_i represents phrase frequency in message i and TF is the total phrase frequency in a thread. Besides, 0log0=0 is also defined.

4. Experiments and Results

In this section, we will introduce several experiments to prove the effectiveness of the proposed methods. First, we describe the experiment setup and evaluation of parent salient phrases extraction. Second, child salient phrases extraction is closely examined. Last, the work of thread ranking in each cluster is evaluated.

4.1. Parent Salient Phrase Extraction

All the experiments are based on 6,458 messages crawled from the websites in Table 1. The main topic of this dataset is about "computer virus".

Table 1. Websites of our messages resource

.com.cn	pconline.com.cn, zol.com.cn, zdnet.com.cn, enet.com.cn
.com	forum.ikaka.com, chinadforce.com, qq.com, yesky.com
.net	pchome.net, langfang.net

- **Evaluation Measure**

We use precision at top N results to measure the performance:

$$P@N = \frac{|C \cap R|}{R} \quad (13)$$

where R is the set of top N salient phrases extracted by our method, and C is the set of manually tagged correct salient phrases. For parent salient phrase, we use $P@10$, $P@20$, and $P@30$ for evaluation.

- **Training Data Collection**

After all the messages are pre-processed, we get a list of candidate salient phrases which are all nouns or nouns phrases by a Chinese POS tagger [6]. We first simply rank phrases by their phrase frequency, and top 200 phrases are used for labelling, since the number of phrases of the whole corpus is too large and phrases appearing rarely are hardly representative for topics. Four graduate students are invited to label these phrases. We assign 1 to y values for phrases that are considered as salient, and assign 0 to y values for others. These y values together with phrase properties are used in regression.

- **Experimental Result**

We first use each single property described in Section 3.1 to rank phrases. The average precision at top 10, 20 and 30 are shown in Figure 2.

From Figure 2, we could observe that *TFIDF* and Phrase frequencies in title (*TITLE*) are much better indictors for salient score than other properties. We prove our hypothesis that in BBS users usually put the most important information in titles in order to attract others' attention. *ATFO* does not work well in our dataset, since our messages not only contain pure Question\Answer and opinion discussion messages, but also include some long advertisements and long documents transcribed from

other websites. These two are common phenomena in Chinese BBS portals. Besides, depth of messages in threads (*DEPTH*) is also not a good indicator in this dataset. The reason might be that our "computer virus" dataset is technique oriented, and there are less topic drift phenomena existed. Maybe, in some boards like "news" and "entertainment", depth will be an important property.

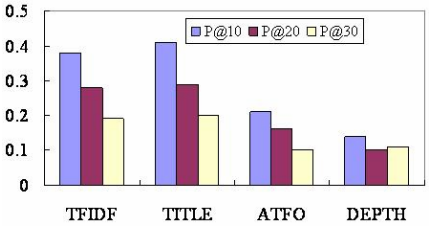


Figure 2. Performance of each single property.

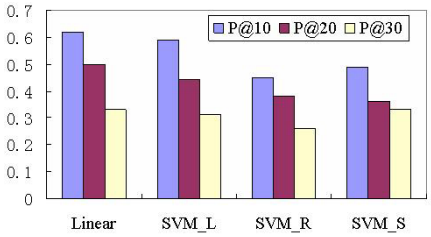


Figure 3. Performance of each regression model.

In order to do regression analysis, we partition the dataset into four parts and use four-fold cross validation to evaluate the average performance of salient term extraction. For support vector regression, three kernel functions are used here: linear kernel (denoted by SVM_L), RBF kernel (denoted by SVM_R) and sigmoid tanh kernel (denoted by SVM_S). The comparison of these regression models is shown in Figure 3. The precision achieved by linear regression and SVM_L are almost same and both gain significant improvements than each single property in Figure 2. We now give the single equation that combines all the properties together,

$$y = 0.072 + 0.720 \times TFIDF + 0.834 \times TITLE + 0.125 \times ATFO + 0.022 \times DEPTH$$

The coefficients are obtained by linear regression training. Each property has been normalized by divided by their maximal value.

4.2. Child Salient Phrase Extraction

In this part, child salient phrase extraction method will be evaluated. Evaluating the concept hierarchies is really a big challenge. Because of the lager number of total child phrase candidates, it is very difficult to label the ground truth data. Here, we utilize a method to directly evaluate the relationship between parent phrases and child phrases, which was introduced in [13]. Five of the organizing relations in [13] were presented here.

1. Child is an aspect of parent, e.g. "端口"(port) is an aspect of "防火墙"(firewall).
2. Child is a type of parent, e.g. "特洛伊"(Trojan) is a type of "木马"(horse).
3. Child is as same as parent.
4. Child is opposite of parent.
5. Do not know or they have some other relations.

The first two relation types indicate that a child is more specific than its parent, namely this parent-child pair is meaningful. Most of the users focus on the top ranked salient phrases, so top 10 parents each with its top 10 child salient phrases are judged according to the five types of relations above.

- **Experimental Result**

We use P($x|y$) and co-occurrence frequencies (denoted by *P* and *C* in Table 2) together to calculate the scores of candidate child salient phrases. We set the default neighbour size as 10, namely we choose five words before and five words after parent words to form a pool of child word candidates. In addition, we set λ as 0.7 in equation (8) for the experiment using some heuristics.

As it can be seen in Table 2, when we combine two properties together, we get 72% (51% + 21%) of the parent-child pairs have the "aspect" or "type" relationships, which outperforms that using the two properties separately.

Table 2. Comparison of different extraction methods

	aspect	type	same	Opposite	don't know
P	23%	25%	7%	1%	44%
C	38%	15%	9%	0%	38%
P&C	51%	21%	7%	1%	20%

We found that in most cases, two levels' hierarchy is enough for users to obtain information, and our methods also could be applied recursively to get more levels of a hierarchy.

4.3. Ranking Threads in Clusters

- **Experiment Setup**

Because of the large number of phrases and threads, we only select the most important phrases and threads for labelling. Top 10 salient phrases and their corresponding threads are labelled by our evaluators. We use P@5, P@10 and P@15 as the basic measures.

- **Experimental Result**

We also partition the data into four parts and use four-fold cross validation to evaluate the average performance. Figure 4 shows the thread ranking precision of top 10 parent phrases. The Y-axis indicates precision of corresponding thread ranking, and the X-axis indicates the top 10 parent salient phrases.

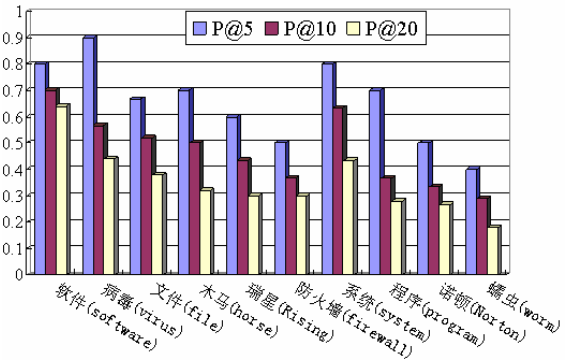


Figure 4. Precision of thread ranking in top 10 parent phrases.

It can be seen that our thread ranking algorithm is very effective. By this method, users could get the most relevant threads of each cluster. From the Figure 4, we still found some clusters do not achieve high precision, such as the cluster labelled by "蠕虫"(worm), since advertisements and poor-quality messages flood BBS. Thus, in the future we will try to find some methods to filter out advertisements and improve the precision of thread ranking.

5. Conclusions

In this paper, we extract salient phrases in Chinese BBS messages as a hierarchical summarization to facilitate users' quick browsing and understanding. In this paper, we proposed novel methods to extract parent and child salient phrases. Moreover, threads are clustered, and the relevant threads could be found easily by our ranking mechanism. Besides, a visual interface is provided for users to find their required information quickly, rather than read messages one by one. The methods can also be applicable to other datasets by a few adjustments, such as some English forums and newsgroups.

References

[1] Fiore, A., LeeTiernan, S., and Smith, M. Observed Behavior and Perceived Value of Authors in Usenet Newsgroups: Bridging the Gap. CHI 2002, April 20-25, 2002, Minneapolis, Minnesota, USA.

[2] Fellbaum, C. WordNet: on Electronic lexical Database, MIT Press.

[3] Fernanda B. Viégas and Marc Smith. Newsgroup crowds and AuthorLines: Visualizing the activity of individuals in conversational cyberspaces.

[4] Ferragina, P., and Gulli, A. A personalized search engine based on web-snippet hierarchical clustering. In www14, 2005.

[5] Ferragina, P., and Gulli, A. The Anatomy of a Hierarchical Clustering Engine for Web-page, News and Book Snippets. Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM'04)

[6] http://www.nlp.org.cn

[7] Kummamuru, K., Lotlikar, R., Roy, S., Singal, K., and Krishnapuram, R. A Hierarchical Monothetic Document Custering Agorithm for Summarization and Browsing Search Results. In Proceedings of 13th International World Wide Web Conference, (2004)

[8] Lawrie D. and Croft W. B. Finding Topic Words for Hierarchical Summarization. In Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'01)

[9] Levenshtein, V.I. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. Cybernetics and Control Theory 10 (1966) 707-710.

[10] Liu, W., Xue, G.R., Huang, S. and Yu, Y. Interactive Chinese Search Results Clustering for Personalization. The 6th International Conference on Web-Age Information Management (WAIM2005), Hangzhou, China, October 11-13, 2005.

[11] Matsuo, Y., and Ishizuka, M. Keyword Extraction from a Single Document using Word Co-occurrence Statistical Information. International Journal on Artificial Intelligence Tools.

[12] Turney, P.D. Coherent key phrase extraction via Web mining, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03).

[13] Sanderson, M., and Croft, W.B. Deriving concept hierarchies from text. In Proceedings of the 22nd annual international ACM SIGIR conference on Research and Development in Information Retrieval, pages 206–213, 1999.

[14] Tonella, P., Ricca, F., Pianta, E., and Girardi, C. Using Keyword Extraction for Web Site Clustering. In 5th International Workshop on Web Site Evolution

[15] Witten, I., Paynter, G., Frank, E., Gutwin, C. and NevillManning, C. KEA: Practical Automatic Key phrase Extraction. In the Proceedings of ACM Digital Libraries Conference, pp. 254-255, 1999.

[16] Xi, W.S., Lind, J., and Brill, E. Learning Effective Ranking Functions for Newsgroup Search. SIGIR'04, July 25–29, 2004, Sheffield, South Yorkshire, UK.

[17] Zeng, H.J., He, Q.C., Chen, Z., Ma, W.Y., and Ma, J.W. Learning to Cluster Web Search Results. SIGIR'04, July 25–29, 2004, Sheffield, South Yorkshire, UK

GeoARM: an Interoperable Framework to Improve Geographic Data Preprocessing and Spatial Association Rule Mining

Vania Bogorny, Paulo Martins Engel, Luis Otavio Alvares

Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Av. Bento Gonçalves, 9500 – Campus do Vale – Bloco IV
Bairro Agronomia – Porto Alegre – RS – Brazil
{vbogorny, engel, alvares} @ inf.ufrgs.Br

Abstract *– Geographic data preprocessing is the most expensive and effort consuming step in the knowledge discovery process, but has received little attention in the literature. For the data mining step, especially for association rule mining, many different algorithms have been proposed. Their main drawback, however, is the huge amount of generated rules, most of which are well known patterns. This paper presents an interoperable framework to reduce both the number of spatial joins in geographic data preprocessing and the number of spatial association rules. Experiments showed a considerable time reduction in geographic data preprocessing and association rule mining, with a very significant reduction of the total number of rules.*

Keywords. Spatial data mining, spatial association rules, framework, knowledge base, geographic data preprocessing

1. INTRODUCTION

Geographic data are real world entities, also called spatial features, which have a location on Earth's surface [1]. Spatial features (e.g. Brazil, Spain) belong to a feature type (e.g. country), and have both non-spatial attributes (e.g. name, population) and spatial attributes (geographic coordinates *x,y)*.

Beyond the spatial attributes, there are implicit spatial relationships, which are intrinsic to geographic data, but usually not explicitly stored in geographic databases (GDB). Because of spatial relationships, real world entities can affect the behavior of other features in the neighborhood, and due this reason they must be extracted for data mining and knowledge discovery [2]. Spatial relationships are the main characteristic which differs knowledge discovery in geographic databases from knowledge discovery in classical databases (KDD).

Figure 1 shows an example of implicit spatial relationships where gas stations and industrial residues repositories may be *close, far,* or maybe *intersect* water bodies. Considering that water analyses showed high chemical pollution, the extraction of spatial relationships among water resources, gas stations, and industrial residues repositories is of fundamental importance for knowledge discovery.

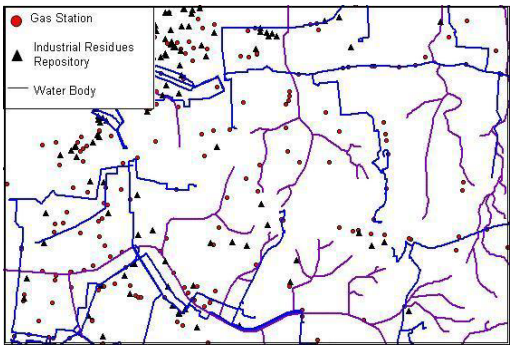


Figure 1 – Examples of implicit spatial relationships

Although spatial relationships are the main characteristic to be considered in data mining, not all relationships hold interesting information. Many relationships are well known geographic domain associations that may hind the discovery process and produce a large number of patterns without novel and useful knowledge.

Figure 2 shows an example of well known geographic domain dependences between gas stations and streets. Notice that there is an explicit pattern: all gas stations intersect streets. Such relationships produce patterns with 100% confidence in the discovery process.

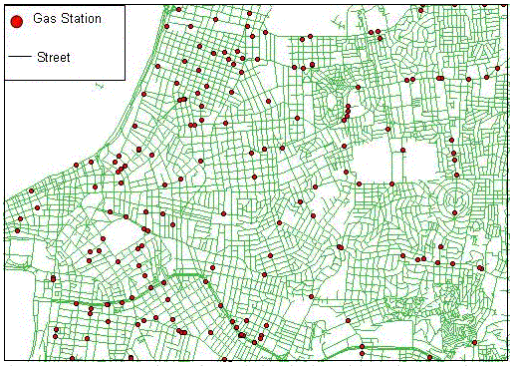


Figure 2 – Examples of spatial relationships that produce well known geographic domain patterns

In spatial association rule mining, which is a data mining technique that has been extensively used to extract knowledge from GDB, well known associations (dependences) generate two main problems:

a) *Data preprocessing:* a large computational time is required to preprocess GDB to extract spatial relationships. The spatial join (cartesian product) operation, required to extract spatial predicates, is the most expensive operation in databases and the big processing bottleneck of spatial data analysis and knowledge discovery;
b) *Association rule mining:* a large number of spatial association rules without novel, useful, and interesting knowledge is generated.

Although users may be interested in high confidence rules, not all strong rules necessarily hold considerable information. Moreover, the mixed presentation of hundreds or thousands of interesting and uninteresting rules can discourage users from interpreting them all in order to find 'patterns' of novel and unexpected knowledge.

1.1 The Problem of Mining Association Rules with Well Known Dependences

We illustrate the problem of mining spatial association rules with well known geographic associations through a small experiment. Every row of the dataset mined was a district of the city of Porto Alegre – Brazil (target feature type), and the columns were two non-spatial attributes (hepatitis rate and sanitary condition) and three relevant spatial feature types with a spatial relationship with district (water bodies, slums, and treated water network). We wanted to investigate associations to high hepatitis incidence. District and treated water network have a well known geographic dependence because every water network belongs to one or more districts and all districts have water networks. We mined spatial association rules with three different values of minimum support, and analyzed every generated rule. The summary of the results is shown in Table 1.

Table 1. Spatial association rule reduction by eliminating one dependence

Min. Sup.	Mining With Dependences			Removing dependence in data pre-processing
	Total Numb. rules	Rules with the dependence	Rules without the dependence	
0.1	203	163	40	40
0.15	91	78	13	13
0.2	32	31	1	1

A total of 203 rules was generated for minimum support 0.1. Among the 203 rules, 163 had the dependence, and only 40 were generated without the dependence. In this case the user would have to analyze 203 rules, among which only 40 would be the most interesting, because 163 had the well known dependence.

Notice that even increasing minimum support to 0.2, from a total number of 32 rules, 31 were still generated with the dependence. This shows that minimum support does not warrant the elimination of rules with well known associations, and increasing minimum support may eliminate interesting rules.

By removing the dependence in data preprocessing (column on the right in Table 1), association rule mining algorithms will generate only rules without the dependence.

Existing algorithms for mining spatial association rules [3][4][5][6] are not intelligent enough to decide which spatial relationships are relevant to the discovery process and which are well known geographic domain associations. Geographic domain associations represent spatial integrity constraints that are explicitly represented in geographic database schemas and geo-ontologies. However, such knowledge repositories have not been used to improve the KDD process.

In this paper we present an intelligent framework to preprocess geographic databases and eliminate well known dependences for mining spatial association rules.

1.2 Related Works and Contribution

Existing approaches for mining spatial association rules do neither make use of prior knowledge to specify which spatial relationships should be computed in data preprocessing nor to reduce the number of well known patterns. Koperski [3] presented an approach for mining spatial association rules which is a top-down, progressive refinement method. In a first step spatial approximations are calculated, and in a second step, more precise spatial relationships are computed to the result of the first step. Minimun support is used in data preprocessing to extract frequent spatial relationships. The method has been implemented in the module Geo-Associator of the GeoMiner [7] system. A similar method has been proposed by [5] for geographic objects with broad boundaries.

Appice [4] proposed an upgrade of Geo-Associator to first-order logic, and all spatial features and spatial relationships are extracted from spatial databases and represented on a deductive relational database. This process is computationally expensive and non-trivial with real data.

Mennis [6] applied Apriori[8] to geographic data to extract spatial association rules. Data preprocessing is performed with the operations provided by GIS, but no prior knowledge is used to either improve data preprocessing or prune well known association rules.

In these approaches rules are filtered by thresholds of minimum support and minimum confidence, which do not warrant the elimination of well known geographic dependences. The main difference from these approaches and our framework is that we automatically preprocess geographic databases using prior knowledge to reduce both the number of spatial joins and the number of rules.

The main contribution of this paper is the reduction of well

known geographic patterns and the reduction of the computational time to preprocess geographic databases and the extraction of association rules.

The proposed framework was implemented in Weka [9], which we extended to support dynamic geographic data preprocessing, discretization, and transformation for mining spatial association rules at different granularity levels. Weka is a well established free and open source toolkit with friendly and graphical user interface which covers the whole KDD process. Other systems such as ARES or GeoMiner do not provide the same advantages, or are no longer available outside academic institutions. Moreover, any association rule mining algorithm may be applied using our framework, since the well known associations are pruned in data preprocessing steps.

1.3 Scope and Outline

The proposed framework is and extension of [10] to improve the KDD process using prior knowledge. The focus in this paper is to show how prior knowledge can be used to improve geographic data preprocessing and spatial association rule mining, and not how to represent geographic knowledge into a knowledge base.

The remainder of the paper is organized as follows: Section 2 introduces geographic domain associations and shows how they are represented in geographic database schemas and geo-ontologies. Section 3 presents the framework for enhancing geographic data preprocessing and spatial association rule mining. Section 4 shows experiments with real databases, while Section 5 concludes the paper and suggests some directions of future work.

2. GEOGRAPHIC DOMAIN ASSOCIATIONS

Well known geographic associations are mandatory spatial relationships, usually represented as spatial integrity constraints [11] in geographic database schemas [12][13] and geo-ontologies [14]. They can also be provided by the user, and in this case, a larger set of associations can be specified; not only associations explicitly represented in the schema, but many other geographic domain associations which produce well known patterns.

In geographic database schemas, well known geographic associations are normally represented by *one-one* and *one-many* cardinality constraints [13] that must hold in order to warrant the consistency of the data. Figure 3 shows an example of a geographic database schema, represented in a UML [15] class diagram. Explicit mandatory *one-one* and *one-many* relationships between gas stations and streets, county and streets, water resources and counties, as well as islands and water resources will produce well known rules because they always hold if the database is consistent. Notice that gas stations and water resources or islands and gas stations do not have any *explicit* mandatory relationship represented in the schema, so their *implicit* relationships may be interesting for knowledge discovery.

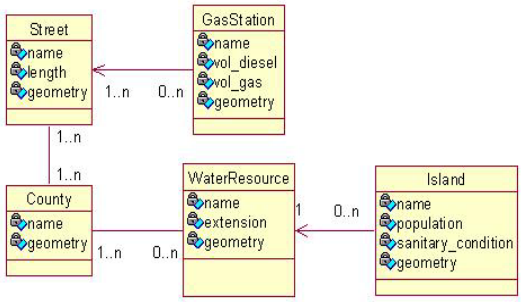


Figure 3 - Part of a conceptual geographic database schema

In the logical level, *one-one* and *one-many* relationships normally result in foreign-keys in relational geographic databases, and in pointers to classes in object-oriented geographic databases [13]. So they can be automatically retrieved with processes of reverse engineering [16] if the schema is not available.

Many different approaches to extract associations from relational databases using reverse engineering are available in the literature. For data mining and knowledge discovery in non-geographic databases reverse engineering has been used to understand the data model in legacy systems [17], or to automatically extract SQL queries [18], but not as prior knowledge to reduce well known patterns.

Ontologies are also rich knowledge repositories that have been used recently in many and different fields in Computer Science. Although research is not so far yet in ontologies for geographic data, some geo-ontologies have been emerging recently. Besides defining a geo-ontology for administrative data for the country of Portugal, Chaves [19] defined a geo-ontology meta-model, named GKB (Geographic Knowledge Base). In [14] we extended this approach to support spatial integrity constraints.

A mandatory *one-one* relationship between island and water resource, for example, may be specified in a geo-ontology with cardinality constraints, as shown bellow.

```
<owl:Class rdf:ID="Island">
  <rdfs:SubClassOf rdf:resource="#SpatialFeatureType"/>
  <rdfs:subClassOf>
   <owl:Restriction>
    <owl:minCardinality
     rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
     1</owl:minCardinality>
    <owl:allValuesFrom rdf:resource="#WaterResource"/>
    <owl:OnProperty>
     <owl:ObjectProperty rdf:about="#Within"/>
    </owl:OnProperty>
   </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Geographic associations produce well known patterns independently of the type of topological (e.g. *touches, contains, crosses*, etc) or distance relationship. For example, if there is a gas station, then there must be a street, independently if the topological relationship is *touches, crosses, within* or *intersects*. A large number of non-novel

rules is generated for each different topological relationship occurring more often than a specified minimum support (e.g. *contains (GasStation)→* ***crosses****(Street); contains (GasStation)→* ***touches****(Street)).* Due this reason, if a pair of geographic feature types is specified in the knowledge base as a pair with a well known dependence, then no spatial relationship among the pair needs to be computed.

3. THE *GEOARM* FRAMEWORK

In order to optimize GDB preprocessing and spatial association rule mining, Figure 4 shows an interoperable framework with support to the whole discovery process using the association rule mining technique. It is composed of three abstraction levels, as described in [10]: data mining, data preparation, and data repository.

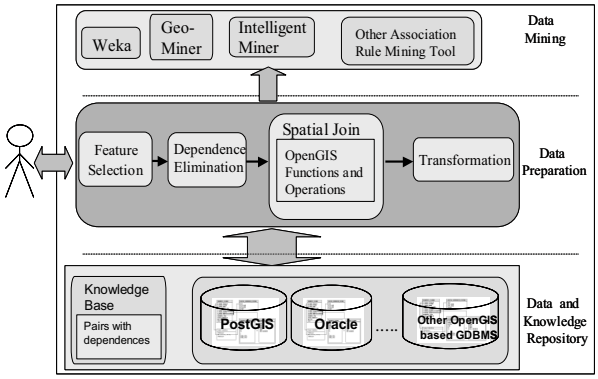


Figure 4 – *GeoARM* (Geographic Association Rule Miner)

At the bottom are the geographic data repositories, stored in GDBMS (geographic database management systems), constructed under OGC specifications. There is also a knowledge repository which stores well known geographic associations, extracted from database schemas, geo-ontologies, or provided by the user. At the top are the data mining toolkits or algorithms for mining association rules. In the center is the spatial data preparation level which covers the *gap* between data mining tools and geographic databases. In this level the data repositories are accessed through JDBC connections and data are retrieved, preprocessed, and transformed into the single table format, according to the user specifications.

There are four main modules to implement the tasks of geographic data preparation for association rule mining: *feature selection, dependence elimination, spatial join,* and *transformation,* which are described in the sequence.

3.1 Feature Selection and Dependence Elimination

The *Feature Selection* module retrieves all relevant information from the database, including the target feature type (TFT), the target feature non-spatial attributes and the set of relevant feature types (RFT) that may have some influence on the TFT. The feature types are retrieved through the OpenGIS database schema, stored in the table GEOMETRY_COLUMNS.

The *Dependence Elimination* module verifies all associations between the target feature type and all relevant feature types. It searches the knowledge base and if the TFT has a dependence with a RFT, then the RFT is eliminated from the set *S* of relevant feature types. Notice that for each relevant feature type removed from the set *S*, no spatial join is required to extract spatial relationships. By consequence, no spatial association rule will be generated with this relevant feature type.

3.2 Spatial Join

The *Spatial Join* module computes and materializes the user-specified spatial relationships between the TFT and the RFT, retrieved by the *Feature Selection* module and filtered by *Dependence Elimination* module. Three types of spatial relationships are materialized:

a) *Topological*: computes the detailed topological relationships (e.g. touches, contains);
b) *Intersection*: extracts more general topological relationships, i.e., if the TFT intersects or not the RFT;
c) *Distance*: extracts *close* and *far* distance relations. *Close* is dominant over *far* in the feature type granularity level due the fact that close objects are more co-related than objects that are far. For instance, if an instance of the TFT is close to some instances of a RFT and far from others, *close* is materialized and *far* is unconsidered.

Spatial joins to extract spatial predicates are performed on-the-fly with operations provided by the GDBMS, and only over the relevant feature types defined by the user. We follow the Open GIS Consortium (OGC) [1] specifications, what makes *GeoARM* interoperable with all GDBMS constructed under OGC specifications (e.g. Oracle, PostGIS, MySQL, etc). The OGC is a not-for-profit organization dedicated to provide interoperability for Geographic Information Systems. Besides a standard set of operations to compute spatial relations for SQL, implemented by most GDBMS, the OGC also defines a database schema for storage of spatial data with all database characteristics.

Before compute spatial joins, MBR (minimum boundary rectangle) is performed for accelerating the extraction of spatial relationships. The *Spatial Join* module output is stored into a temporary database table *T* with the following attributes: *TFT* instance identifier, the spatial relation between *TFT* and *RFT*, the *RFT* name, and the *RFT* identifier.

3.3 Transformation

The *Transformation* module transposes as well as discretisizes the *Spatial Join* module output (table *T*) into the single table representation, understandable by association rule mining algorithms. The single table *ST* is created with the TFT non-spatial attributes and all its

instances. Then for each instance in *ST*, a new attribute is created for every different RFT in *T,* which has a spatial relation with the target feature.

This module requires two user-specified parameters: relevant features granularity - defines the level of data representation for data mining, which can be on feature instance (e.g. factory A, factory B) or feature type (e.g. factory) level; - data mining algorithm - defines the exact output format which can lithely vary according to each rule mining algorithm.

4. EXPERIMENTS AND EVALUATION

In order to evaluate the interoperability of the framework, experiments were performed with real geographic databases stored under Oracle 10g and PostGIS. Census sectors, a database table with 2157 polygons and non-spatial attributes such as population, sanitary condition, etc, was defined as the TFT. Datasets with different relevant feature types (e.g. bus routes – 4062 multilines, slums – 513 polygons, water resources – 1030 multilines) were preprocessed and mined, using prior knowledge and without using prior knowledge. The process was performed with the extended Weka and the Apriori algorithm, using different values of minimum support.

4.1 Evaluation of the Method for Data Preprocessing

Considering the granularity level of feature type and two dependences among the target feature type and the relevant feature types, Figure 5 shows a graph with the computational time reduction using *GeoARM* for data preprocessing.

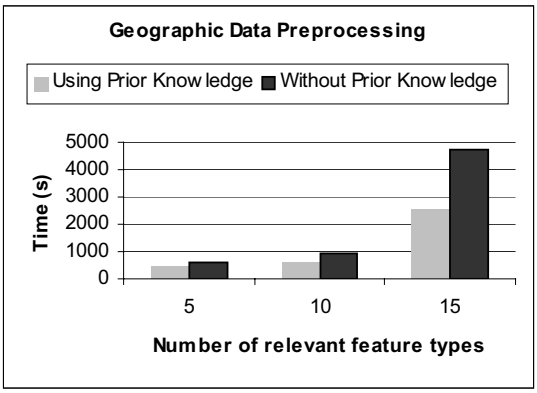


Figure 5 - Geographic data preprocessing

As can be observed in Figure 5, time reduction when preprocessing geographic databases using prior knowledge is significant even when only two well known dependences are eliminated. However, time decreases according to both number of instances and geometry type of the RFT being eliminated. For example, being the instances of the eliminated RFT 10.000 *polygons*, the time reduction would be much more significant than if the instances were, for example, 2.000 *points*.

In summary, it is difficult to precise the time reduction in data preprocessing, which varies according to the data. However, the elimination of relevant feature types avoids the spatial join operation, and this warrants the time reduction in data preprocessing.

4.2 Evaluating the Method for Mining Association Rules

Figure 6 describes one of the association rule mining experiments, with the elimination of 2 well known dependences. Notice that *GeoARM* eliminated around 68% of the number of rules even when only one dependence is removed.

When 2 dependences are removed, our framework eliminated 90% of the rules. While other methods prune rules using higher minimum support, our framework reduces rules independently of this threshold. Moreover, increasing minimum support can eliminate interesting rules, while our method warrants that only rules with well known dependences will be eliminated.

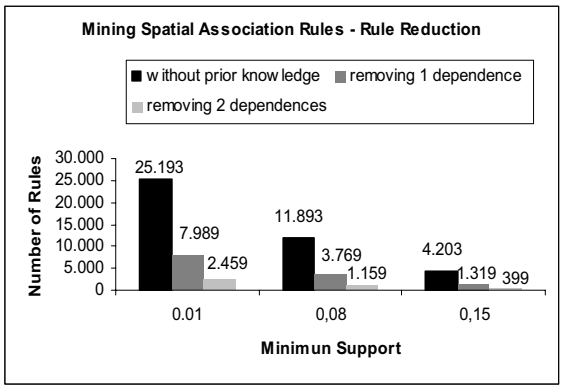


Figure 6 - Association rule reduction using Apriori

Figure 7 shows the time reduction in association rule mining when prior knowledge is used in data preprocessing. Notice that for any value of minimum support, the time reduction reaches 50% if one dependence is removed and more than 70% when two dependences are eliminated.

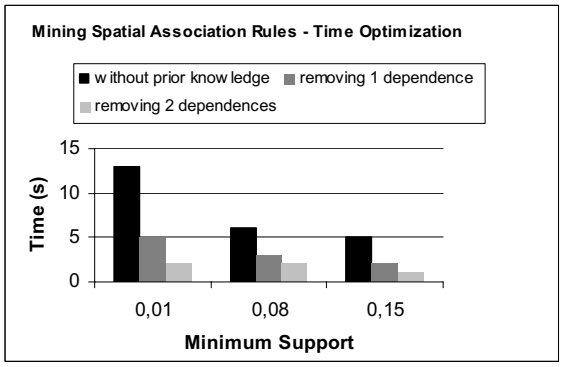


Figure 7- Computational time reduction using Apriori

5. CONCLUSIONS AND FUTURE WORK

In this paper we presented a framework for mining spatial association rules from geographic databases using prior knowledge in data preprocessing. The domain knowledge refers to mandatory geographic associations which are explicitly represented in geographic database schemas and geo-ontologies. We showed that explicit mandatory relationships produce irrelevant patterns, while the implicit spatial relationships may lead to more interesting rules.

Experiments showed that independent of the number of elements, one dependence is enough to prune a large number of rules, and the higher the number of the well known dependences, the larger is the rule reduction.

The main contribution of our approach is for the data mining user, which will analyze much less obvious rules. The method is effective independently of other thresholds, and it warrants that geographic domain associations between the target feature type and the relevant feature types will not appear among the set of rules.

The use of prior knowledge in geographic data preprocessing has three main advantages: spatial relationships between features with dependences are not computed; time reduction is very significant in data preprocessing and rule mining; and the most significant, the reduction of the number of rules.

One limitation of the framework, however, is that only well known associations between the target feature type and the relevant feature types are eliminated. As future work we will remove well known associations among relevant feature types, which can only be performed into the association rule mining algorithm.

ACKNOWLEDGMENT

Our thanks for both CAPES and CNPQ which partially provided the financial support for this research. To Procempa, for the geographic database, and for Instituto de Informática da UFRGS, for the registration fee.

REFERENCES

1. Open Gis Consortium. Topic 5, the *OpenGIS* abstract specification – *OpenGIS* features – Version 4 (1999). Available at: http://www.*OpenGIS*.org/techno/specs.htm. Accessed in August (2005).
2. Ester, M., Kriegel, H-P, Sander, J. (1997). Spatial Data Mining: A Database Approach. In Proceedings of the 5th International Symposium on Large Spatial Databases (SSD'07), Springer, Berlin, Germany, pp. 47-66.
3. Koperski, K., and Han, J. Discovery of spatial association rules in geographic information databases. In Proceedings of the SSD 4th international Symposium in large Spatial Databases, (SSD'95) Springer, Portland, Maine, USA, 1995, p.47-66.
4. Appice, A., et al. Discovery of spatial association rules in geo-referenced census data: A relational Mining Approach. Intelligent Data Analysis. Vol. 6, 2003.
5. Clementini, E., Di Felice, P., Koperski, K. Mining multiple-level spatial association rules for objects with a broad boundary. *Data & Knowledge Engineering*, 34(3):251-270 (2000)
6. Mennis, J., Liu, J. Mining association rules in spatio-temporal data: an analysis of urban socioeconomic and land cover change. Transactions in GIS 9 (1), 2005, 5-17.
7. Han, J., Koperski, K., Stefanovic, N. GeoMiner: A System Prototype for Spatial Data Mining, in SIGMOD 1997, J. Peckham, ed., Proceedings of the ACM-SIGMOD International Conference on Management of Data, *SIGMOD Record* **26**(2) (1997), 553–556.
8. Agrawal, R., Srikant, R. Fast algorithms for mining association rules. In Proceedings of the 20th Int'l Conference on Very Large Databases (VLDB'94). (September 1994), Santiago, Chile, 1994.
9. Witten, I., H., Frank, E. Data Mining: Practical Machine Learning Tools And Techniques With Java Implementations. Morgan Kaufmann Publishers, San Francisco, CA, 2000.
10. Bogorny, V., Engel, P. M., and Alvares, L.O. A reuse-based spatial data preparation framework for data mining. In Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering, (SEKE'05). Jul. 2005, Taipei, Taiwan, 2005, 649-652.
11. Servigne, S. et al. A Methodology for Spatial Consistency Improvement of Geographic Databases. *Geoinformatica*, Kluwer Academic Publishers, Hingham 4-1 (2000) 7-34.
12. Bogorny, V., Iochpe, C. Extending the OpenGIS Model to Support Topological Integrity Constraints. In *Proceedings of the Brazilian Symposium on Databases*, (SBBD'2001) COPPE/UFRJ, Rio de Janeiro, Brazil, 2001, 25-39 .
13. Shekhar, S., and Chawla, S. Spatial databases: a tour. Prentice Hall, Upper Saddle River, NJ, 2003.
14. Bogorny, V., Engel, P. M., Alvares, L.O. Towards the Reduction of Spatial Join for Knowledge Discovery in Geographic Databases using Geo-Ontologies and Spatial Integrity Constraints. In: Second Workshop on Knowledge Discovery and Ontologies (KDO'2005), in conjunction with the ECML/PKDD, Porto, Portugal, 2005, 51-58.
15. Booch, G., Rumbaugh, J. and Jacobson, I. The unified modeling language: user guide. Addison-Wesley, 1998.
16. Chifosky, E.J., Cross, J.H. Reverse Engineering and Design Recovery: a taxonomy. IEEE Software, Jan .1990.
17. MCKearney, S., Roberts, H. Reverse engineering databases for knowledge discovery. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96). Portland, Oregon, August 1996. 375-378.
18. Shoval, P., Shreiber, N. Database reverse engineering: from the relational to the binary relationship model. Data and Knowledge Engineering. 10: 293-315, 1993.
19. Chaves, M. S. et al. A Geographic Knowledge Base for SemanticWeb Applications. In: Brazilian Symposium on Databases (SBBD'2005), Uberlandia, Brazil (2005).

Classification by Multi-Perspective Representation Method

Jia Zeng* Reda Alhajj†

Abstract

Classification algorithms have been applied to various domains, such as biomedical data classification, pattern recognition, document classification, etc. In this paper, we present a multi-perspective representation (MPR) method, which is based on the idea of analyzing an object from different perspectives. We explore the application of the MPR method on two real-world data collections. The experimental results on both data sets demonstrate that the MPR method is well applicable and effective to the classification task.

1 Introduction

Classification is a supervised learning process. The task of a classifier is to predict a class label for any valid input object after analyzing a number of training samples. Many well developed automated learning algorithms, such as *support vector machine* (SVM), neural network methods and decision tree approaches, have been shown to be successful in the field of data classification.

The *multi-perspective representation* (MPR) method was originally proposed by Chen, Zeng and Tokuda for information retrieval purposes [2]. It assumes that a document can be observed from multiple perspectives to obtain its derivatives. By exploiting the derivatives as *intermediates*, the analysis of an original document can be indirectly constructed by integrating the results obtained from the analysis conducted on its derivatives. This approach has been shown to be effective in the field of information retrieval and it is also considered to be a generalized methodology that is expected to be well applicable to many other fields.

It has been noted that the classification problem has a very close association with information retrieval tasks [9]. This can be seen as follows. One basic strategy that a classifier uses to label a given data is to analyze the similarities between all of the existent classes and the given unlabeled query data, and then label the query data by the class with the biggest similarity value. Similarly, an information retrieval algorithm is designed to compare the similarities between a given query and all the existent information in storage, and returns to the user the information that is the most relevant to his/her query. Due to the analogy between classification and information retrieval, we believe that a successful information retrieval algorithm has the potential to perform well in a data classification task.

In this paper, we investigate the application of the MPR method in the field of classification. Experiments have been conducted using two benchmark data sets: *Leukemia cancer data collection* and *wine data collection*. The results on both collections have demonstrated the applicability and effectiveness of using the MPR method for data classification.

The remaining parts of this paper are organized as follows. In Section 2, we provide an overview to the classic classification algorithms. In Section 3, we present the proposed MPR method. A detailed description of the experiments is presented in Section 4. Concluding remarks and future work are described in Section 5.

2 Related Work

Classification is a well-researched area. A number of algorithms that have been proposed in the literature have been used to solve the classification problems in different applications. In this section, we present a brief overview of some of the representative classification approaches: the centroid-based method, the nearest neighbor, support vector machine, neural network and decision tree.

Among all the classification methods under discussion, the centroid-based method, the nearest neighbor method and the support vector machine share one common characteristic, i.e., they all try to find a mapping between the data to by analyzed and its geometric representation on a multi-dimensional vector space. More specifically, in these algorithms, the training set of data objects with a priori known class are processed so that a complete set of features that are used to describe these data are extracted. Each feature is assigned a dimension to form a multi-dimensional *feature*

*Jia Zeng is with the Department of Computer Science, University of Calgary, Calgary, Alberta, Canada, jzeng@cpsc.ucalgary.ca

†Reda Alhajj is with the Department of Computer Science, University of Calgary, Calgary, Alberta, Canada, alhajj@cpsc.ucalgary.ca; he is also affiliated with the Department of Computer Science, Global University, Beirut, Lebanon

space. All the data objects in the training set are mapped onto this feature space so that their vector representations can be obtained. Based upon these vector representations, the classifier can set up a model that summarizes the information conveyed by the training data. This phase is called the learning stage. In the testing phase, any given unlabeled data is processed to obtain its vector representation in the feature space. Then the classifier conducts analysis on the new feature vector using the model that has been pregenerated and labels the query data accordingly.

The difference between the three algorithms mentioned above is how the classifier sets up the model in its learning phase. The centroid-based method assumes that each class has an explicit profile, also called the *class centroid*. The class centroid can be calculated and used as a representative of all the data items belonging to this class. To find the class that is the most similar to the query data, the classifier only needs to compare the query data to the centroids of all the existent classes. Therefore, the centroid-based approaches are efficient in terms of its computational complexity, which is a linear function of the total number of classes. However, in order to achieve an effective classification result, the algorithm of computing the centroid should be analyzed so as to make sure that the centroid is a good summarization of the class which it represents.

In the nearest neighbor approach, all the data objects that exist in the training set are taken into consideration while the classifier tries to locate the most similar class for the query data. To be more specific, the geometric distances between the query vector and the vectors representing the data objects in the training set are calculated and the object whose vector representation is the closest to the query vector is selected as the query data's *nearest neighbor*. Then the classifier labels the query data by the class of its nearest neighbor. A variant of this algorithm, called *k-nearest neighbor* or kNN in short, selects a set of k nearest neighbors and assigns the class label to the query data based upon the most numerous class with the set. The nearest neighbor method has some strong consistency results, i.e., as the amount of data contained in the training set approaches infinity, nearest neighbor is guaranteed to yield an error rate no worse than twice the Bayes error rate (the minimum achievable error rate given the distribution of the data) [6]. However, unlike the centroid-based method, this approach will be more computationally intensive as the size of the training set grows. Many optimization methods have been proposed in the literature, which seek to reduce the number of distances actually computed.

To deal with the classification problem that involves two classes, the support vector machine algorithm creates a hyperplane that separates the data in the training set into two classes with the maximum-margin. The query data is then projected onto this hyperplane and its final position determines the class which it should be labeled by. According to [7], the use of the maximum-margin hyperplane provides a probabilistic error bound that is minimized when the margin is maximized. However, the utility of this theoretical analysis is sometimes questioned given the large slack associated with these bounds [7].

The neural network method, which tries to emulate biological neurons and their behaviors, is also popularly used to solve classification problems. Unlike the three classification approaches we discussed above, which are one-time learning algorithms in principle, the neural network actually has the ability to learn through experience. Its other features, such as being fault tolerant, scalable and flexible, also contribute in making this method advantageous. However, there are also some known problems regarding ththe neural network method. One major drawback is that it is difficult to determine when the learning phase has been achieved and therefore sometimes this model suffers from the problem of over-learning.

Decision tree is a predictive model that uses the form of tree structure. Its internal nodes are tests on the training data and its leaf nodes represent classes. The decision tree is a robust model, which performs well with large amounts of data in a short time. However, similar to the neural network, in this approach, it is also difficult to evaluate the degree of generalization; thus there is still the danger of over-fitting.

3 The Applied Classification Methodology

3.1 Overview of the MPR Method

As mentioned in [2], the inspiration of proposing the *multi-perspective representation* (MPR) approach comes from the observations of some daily life examples.

Throughout the humans' experience of recognizing objects, it has been noted that multi-perspective descriptions have always been beneficial. Take visual and acoustic recognition for example. It has been known that we need two eyes in order to obtain a depth perception of an object. As well, using both of our ears helps people to distinguish subtle differences among different music styles. These observations have led to the belief that perceiving objects from two or more perspectives is always an advantage.

The MPR method has applied the idea of analyzing one object from different perspectives into the areas of information science. It is assumed that each information object can be observed from different perspectives. Analysis can be conducted on each of the derivatives of the object to be investigated. These results are then combined together in order to reconstruct the analysis on the original information object.

3.2 Applying the MPR Method for Classification

In a typical classification data set, there are many different classes consisting of multiple different member items, and all the member items from the same class share some commonalities. We can interpret this by the MPR method in the following way. Each class contained in the data collection is an information object to be investigated, which we call the *original class*. All the data items that belong to the same class are considered to be the *elementary components*, which can be used to represent the attributes of the original class.

Based upon these elementary components, we can then obtain multiple derivatives, also called *perspective classes*, of the original class. These derived classes are analyzed by a classification algorithm to arrive at some intermediate results. Finally, a combination strategy is applied so that the analysis on the original class is restored.

3.2.1 Obtain Multi-Perspective Representations

There is no fixed solution for implementing the procedure of perceiving one class from different perspectives, i.e., to obtain its multiple derivatives. To facilitate the understanding of how this process can possibly be implemented, we offer some sample strategies that might be applicable to a number of cases.

The first strategy is called the *random scheme*. Suppose we have a data collection D that contains N_c classes: $\{C_1, C_2, \cdots, C_{N_c}\}$. For each class C_i, there exist n_i data items: $\{d_{i1}, d_{i2}, \cdots, d_{in_i}\}$. Let us denote the total number of data existent in the collection by N_d, which is equal to $\sum_{j=1}^{N_c} n_j$.

We can obtain multiple perspective representations for each original class in the following way. For each class C_i, where $1 \leq i \leq N_c$, we assign it an array of indices ranging from $[1, n_i]$, where data item d_{ij} in class C_i is correspondent to the j^{th} index in the array. This array is then shuffled and a random index array is obtained. According to the newly assigned indices, the data items in class C_i are evenly distributed into p disjoint sets, where p denotes the number of perspectives conducted on class C_i.

The second strategy is called the *sequential scheme*, where a concept of *overlapping*, denoted by o, is introduced. This scheme takes all the data items contained in class C_i and distributes them to p different perspective classes in such a way that for every $o + p$ data items in C_i, o items are shared by all the p perspective classes that are associated with C_i.

Upon completion of applying either of the two schemes discussed above, we can obtain a new representation of the original data collection D. This new data collection, denoted by D', consists of p different sub-collections: $\{P_1, P_2, \cdots, P_p\}$, which we call the *perspective collections*. Each perspective collection is a set of N_c perspective classes that are associated with the original classes $C_1, C_2, \cdots, C_{N_c}$, respectively.

3.2.2 Similarity Evaluation

Having obtained multiple perspective collections for the original collection, we can then temporarily treat each perspective collection P_k ($1 \leq k \leq p$) as an original data collection. For a given unlabeled query data q, we can conduct a similarity evaluation on data q and the perspective classes that are contained in the perspective collection P_k. The perspective class that demonstrates the biggest degree of similarity to q is used to label q.

In principle, any reasonable classification method can be used to conduct such a similarity evaluation on the perspective collections. In this paper, we have focused on applying two vector-space model based approaches: the *standard term vector method* and the *latent semantic indexing* (LSI) method.

The basic idea of vector-space model is to represent each object as a vector. Suppose we have a classification collection consisting of data items that are associated with m features. For each data item d_{ij}, we assign it a *feature vector* of m dimension, denoted by v_{ij}. For a class C_i that contains data items $\{d_{i1}, d_{i2}, \cdots, d_{in_i}\}$, we can calculate the *centroid vector* of C_i, denoted by s_i, as follows.

$$s_i = \frac{\sum_{j=1}^{n_i} v_{ij}}{n_i} \tag{1}$$

To estimate the similarity between a class C_i and a given unlabeled data q, the standard term vector method employs a similarity measure to evaluate the similarity between the centroid vector of C_i and the vector of q, denoted by v_q. One popular metric is called *cosine similarity* and it can be evaluated by the following equation:

$$\cos(s_i, v_q) = \frac{s_i \cdot v_q}{\|s_i\| \|v_q\|} \tag{2}$$

The LSI method was developed based upon the standard term vector method [3]. The major difference is that LSI applies a dimensional reduction scheme, called *singular value decomposition* (SVD), and consequently, all the features vectors are projected onto a reduced space, called the *LSI space*. Therefore, each data item is represented by the projection of its feature vector onto the LSI space.

After applying a similarity evaluation on each perspective collection P_i, we can obtain N_c values that estimate the similarities between q and the N_c perspective classes contained in P_i. Therefore, by labeling q with the perspective class that demonstrates the biggest similarity value, an intermediate classification result can be obtained by the analysis on P_i. As easily can be seen, both the standard term

vector method and the LSI method fall into the category of centroid-based approaches.

3.2.3 Combination Strategy

In order to effectively reconstruct the analysis for each original class through the intermediates' contribution, we have to employ a combination strategy. There are many possible ways to implement this scheme. In this paper, we present a very simple method, called the *voting strategy*.

In order to apply the voting strategy, we need to specify the number of perspectives, p, to be an odd number that is bigger than or equal to N_c. Denote $\{l_1, l_2, \cdots, l_p\}$ to be the array of labels, where l_i represents the class label resulted from the similarity evaluation on the i^{th} perspective collection and q, i.e., which class should the data q be labeled by. Then we can conduct a majority vote on the elements in the label array and apply the "winner-takes-all" strategy, i.e., the class label that wins the vote is considered to be the final decision made by the classifier.

It is worth mentioning that, even though p is an odd number, it is still possible that the voting process results in a tie. Under these circumstances, we suggest looking into the degrees of similarity between q and the classes that win the most votes. Finally, label q by the class that has the biggest similarity value.

4 Experiments

4.1 Leukemia Cancer Data

4.1.1 Problem Description

The challenge of cancer treatment has been to target specific therapies to pathogenetically distinct tumor types, to maximize efficacy and minimize toxicity [4]. Therefore, to improve the accuracy and efficiency of cancer classification has become a central topic to advance the cancer treatment.

The leukemia cancer data set provided by Golub *et al.* [4] is available in public domain[1]. This collection is used for testing the performance of a classifier that intends to distinguish the *acute lymphoblastic leukemia* (ALL) from the *acute myeloid leukemia* (AML). The entire collection contains two data sets: a training set consisting of 38 samples and a testing set consisting of 34 samples.

In the training set, there exist 27 ALL instances and 11 AML instances. In the testing set, there exist 20 ALL instances and 14 AML instances. Each sample is considered to be associated with thousands of genes. For each gene, a quantitative expression level is obtained. Therefore, every sample in the data set is represented by a vector of genes, which makes the use of vector-space-model based approach very appropriate.

4.1.2 Implementation Procedure

In pre-processing the data set, some python[2] scripts are used to remove the unnecessary tags that come with the original collection.

The main method is implemented in GNU Octave[3]. Octave is a high-level language primarily designed for numerical computations. Due to the fact that the vector-space model based approaches mainly represent objects by vectors, it is ideal to choose a language like Octave to achieve an efficient implementation.

To apply the MPR method in this collection, we perceive the original ALL class and the original AML class in the original training set from p different perspectives. Therefore, p different perspective training sets are obtained, each of which contains a perspective ALL class and a perspective AML class.

Since the original training set only contains 38 instances in total, we believe that compared to the random scheme, the sequential scheme would be a more appropriate choice because it suggests that overlapping content exist in all the perspective training sets and consequently the size of each perspective training set would not be too small to adequately represent the attributes of the original training set.

After conducting the sequential scheme, the samples in the original training set are distributed to p perspective training sets, where for every $o + p$ samples in the original training set, o samples are shared by all the p perspective training sets.

Regarding to the similarity evaluation scheme, we choose to apply the LSI method on this collection. The reason is that like many other gene-expression profiling data, the leukemia collection is represented by high-dimensional gene vectors. As indicated by Golub *et al.*'s work [4], among the thousands of genes that are used to express the cancer data, only a small amount of them are strongly correlated with the class distinction. In other words, the majority of the genes are not highly relevant and thus may be considered as noise for this case. One important feature of the LSI method is that by projecting high-dimensional vectors onto smaller semantic space, there is a good chance that the noise information in the data collection can be removed and the latent semantic features of the data set can be obtained.

To obtain the similarity evaluation, we conduct an LSI analysis to evaluate the similarity between any given query data q in the testing set and each of the perspective training sets. Subsequently, a set of p classification results will be

[1] http://www.broad.mit.edu/cancer/

[2] Python is an interpreted, interactive, object-oriented programming language [5]

[3] Available for free download under the GNU General Public License: http://www.octave.org

obtained from the analysis on all the perspective training sets.

Finally, we apply the voting strategy to combine the intermediate classification results in order to arrive at one single classification result for data q.

4.1.3 Results and Evaluation

In order to evaluate the performance of the MPR classifier on the leukemia collection, we apply the measure of *classification accuracy*, which is the ratio of the number of correctly classified data in the testing set over the total number of data in the testing set. As well, due to the fact that the testing set used in this experiment contains more ALL instances than AML instances, it might result in a bias if we only use classification accuracy as the evaluation metric. Therefore, we also employ the measure of *Matthew's correlation coefficient* (MCC) to evaluate the classification performance.

Table 1. Performance of the classifier, $p = 3$

o	0	1	2	3	4
Accuracy	91.18%	91.18%	91.18%	94.12%	94.12%
MCC	0.8266	0.8266	0.8266	0.8827	0.8827
o	5	6	7	8	9
Accuracy	91.18%	94.12%	94.12%	91.18%	94.12%
MCC	0.8213	0.8827	0.8827	0.8266	0.8827
o	10				
Accuracy	91.18%				
MCC	0.8174				

Table 2. Performance of the classifier, $p = 5$

o	0	10	20
Accuracy	82.35%	94.12%	94.12%
MCC	0.6630	0.8786	0.8827

We have conducted two series of experiments. In the first set of experiments, we assign p to be 3. The overlapping parameter o is set to be ranging from 0 to 10. Table 1 demonstrates the performance of the classifier using the MPR scheme on the LSI method. In the second series of experiments, we let $p = 5$ and three different o values are used: 0, 10, 20. The classification performance is shown in Table 2.

From the above results, we can see that, the classifier based upon the MPR method can achieve as high as 94.1% accuracy. As well, it can be noted that, in applying the sequential scheme, the overlapping content plays a major role in improving the classifier's performance. However, the classification performance is not always enhanced as the value of o increases. In other words, the influence of the parameter o has yet to be determined.

Golub *et al.* [4] have conducted experiments on the same set of leukemia data collection we used in our experiments. They have analyzed the genes that are related to the sample data and sorted them in the order of their degree of correlation to ALL-AML class distinction. Then a fixed subset of genes that are highly correlated to the class distinction is used in predicting the class for a new sample. Their classifier has been used to label the class for each of the data in the testing set. It has been reported that strong predictions are made for 29 samples out of 34 samples. Based on the fact that the classifier using our method correctly predicts 32 samples in the testing set, we can claim that the MPR method using LSI method outperforms the method proposed by Golub *et al.* on this classification task.

4.2 Wine Data

4.2.1 Problem Description

Like the leukemia collection, the wine data set is also a description of a real-world problem. This collection is also made available in the public domain [8].

The data contained in this data set are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars [8]. This analysis determines the quantities of 13 constituents found in each of the three types of wines [8].

The original collection contains 178 samples in total. In particular, the numbers of samples that belong to class 1, class 2 and class 3 are: 59, 71 and 48, respectively.

4.2.2 Implementation Procedure

The wine data set does not separate the data into the training set and the testing set. We have applied a pre-processing to partition the entire set into two disjoint subsets: 90% data are used for training and the other 10% are used for testing. We maintain the original class distribution when carrying out the partition process.

To apply the MPR method, we explore the application of sequential scheme which separates the original training set into p disjoint perspective training sets, where $p = 3$ and $p = 5$ are used for this experiment.

In evaluating the similarity between a given data sample q from the testing set and the perspective training sets, we employ the standard term vector method. There are two reasons why we choose this method. On one hand, the data items in the wine collection are only associated with 13 features. Correspondingly, the vectors that are used to represent these data have only 13 dimensions, which is a fairly small number. Therefore, we believe that the standard term vector approach, which does not apply a dimensional reduction scheme, can perform adequately on this collection. On the other hand, we think it would be interesting to explore

the effectiveness of the proposed MPR method on more than one approach.

After conducting the similarity evaluation and obtaining the intermediate classification results from the analysis on the data q and the p perspective collections, we then apply the voting strategy on these intermediate results to arrive at one single result for the classifier.

Table 3. 90% Data for Training, $p = 3$

o	0	10	20	30	40
Accuracy	100%	94.44%	94.44%	88.89%	94.44%

Table 4. 90% Data for Training, $p = 5$

o	0	10	20	30	40
Accuracy	94.44%	94.44%	94.44%	88.89%	88.89%

Table 5. 80% Data for Training, $p = 3$

o	0	10	20	30	40
Accuracy	97.22%	97.22%	97.22%	97.22%	97.22%

Table 6. 80% Data for Training, $p = 5$

o	0	10	20	30	40
Accuracy	97.22%	97.22%	97.22%	97.22%	97.22%

Table 7. 70% Data for Training, $p = 3$

o	0	10	20	30	40
Accuracy	96.23%	96.23%	96.23%	96.23%	96.23%

Table 8. 70% Data for Training, $p = 5$

o	0	10	20	30	40
Accuracy	96.23%	96.23%	94.34%	94.34%	96.23%

4.2.3 Results and Evaluation

The evaluation criteria we employed in this experiment is the classification accuracy[4]. The experimental results can be seen from Table 3 to Table 8.

In the literature, Aeberhard *et al.* have also used the wine collection to conduct comparison of the performance of some statistical classification methods, such as *quadratic discriminant analysis* (QDA), *linear discriminant analysis* (LDA), *regularized discriminant analysis* (RDA) and 1NN [1]. The best result is achieved by using the RDA method, which yields a 100% classification accuracy. The performance of the other three classifiers in terms of the classification accuracy are stated as follows: QDA 99.4%, LDA 98.9% and 1NN 96.1%.

[4]Since there are three classes in this collection, the MCC metric is no longer applicable.

5 Conclusions

In this paper, we have investigated the application of the multi-perspective representation method in the field of data classification. The experimental results on two real-world benchmark collections have shown that the MPR method, as a general methodology, has a very good applicability to the classification tasks. The classifiers using this approach on either the standard term vector method or the latent semantic indexing method have demonstrated very good classification performance.

References

[1] S. Aeberhard, D. Coomans, and O. de Vel. Comparative analysis of statistical pattern recognition methods in high dimensional settings. *Pattern Recognition*, 27(8):1065–1077, 1994.

[2] L. Chen, J. Zeng, and N. Tokuda. A "stereo" document representation for textual information retrieval. *Journal of the American Society for Information Science and Technology*, 2005. Accepted for publication.

[3] S. Deerwester, S. Dumais, T. Landauer, G. Furnas, and R. Harshman. Indexing by latent semantic indexing. *Journal of the Society for Information Science*, 41:391–407, 1990.

[4] T. Golub, D. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. Mesirov, H. Coller, M. Loh, J. Downling, M. Caligiuri, C. Bloomfield, and E. Lander. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science*, 286:531–537, 1999.

[5] http://www.python.org/doc/Summary.html (Retrieval Time: Dec 2005).

[6] http://en.wikipedia.org/wiki/Nearest_neighbor_%28 pattern_recognition%29 (Retrieval Time: Dec 2005).

[7] http://en.wikipedia.org/wiki/Support_vector_machine (Retrieval Time: Dec 2005).

[8] http://sci2s.ugr.es/keel-dataset/problemas/clasificacion/wine.php (Retrieval Time: Dec 2005).

[9] J. Zeng. Information retrieval by multi-perspective representation. Master's thesis, University of Northern British Columbia, 2005.

An Architecture for Personal Cognitive Assistance

David Garlan, Bradley Schmerl
School of Computer Science, Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA, 15213
USA
+1 412 268 5057
{garlan,schmerl}@cs.cmu.edu

Abstract. Current desktop environments provide weak support for carrying out complex user-oriented tasks. Although individual applications are becoming increasingly sophisticated and feature-rich, users must map their high-level goals to the low-level operational vocabulary of applications, and deal with a myriad of routine tasks (such as keeping up with email, keeping calendars and web sites up-to-date, etc.). An alternative vision is that of a personal cognitive assistant. Like a good secretary, such an assistant would help users accomplish their high-level goals, coordinating the use of multiple applications, automatically handling routine tasks, and, most importantly, adapting to the individual needs of a user over time. In this paper we describe the architecture and its implementation for a personal cognitive assistant called RADAR. Key features include (a) extensibility through the use of a plug-in agent architecture (b) transparent integration with legacy applications and data of today's desktop environments, and (c) extensive use of learning so that the environment adapts to the individual user over time.

Keywords
Personal cognitive assistant, agent, software architecture

1 INTRODUCTION

Computers are playing an increasingly indispensable role in complex day-to-day activities of many people. Email, calendaring systems, daily planners, web sites, and the like are now an essential component of most people's lives.

Unfortunately today's desktop environments provide weak support for carrying out complex user-oriented tasks, or even dealing with the myriad details of handling everyday computer-assisted information, communication, and planning tasks. Although individual applications are becoming increasingly sophisticated, users must map their high-level goals to the low-level vocabulary of specific applications and services, and deal with a barrage of routine tasks, such as keeping up with their email, and keeping their calendars web sites up-to-date.

An alternative vision is that of a *personal cognitive assistant* (PCA). Like a good secretary, a PCA would help users accomplish their high-level goals, coordinating the use of multiple applications, automatically handling routine tasks, and, most importantly, adapting to the individual needs of a user over time. A PCA would also be able to work cooperatively with the user, automating tasks where appropriate, and staying out of the way where not.

However, realizing such a vision raises a number of hard software engineering challenges. First, to be useful and economical, a PCA should dovetail with existing (and future) applications, file systems, and user processes. Even if one could afford to reengineer all existing desktop applications (which we can't), most users would not be inclined to learn to use an entirely new set of applications, regardless of the benefits provided. Second, a PCA should be extensible. That is, it should be possible to incrementally add new capabilities for personal assistance over time, possibly taking advantage of third-party components to increase the range of support. Third, it should be adaptive. Over time the capabilities of the environment should automatically adapt to the needs and preferences of a user, without a lot of specific user guidance and oversight.

In this paper we describe an architecture and its implementation for a PCA, called RADAR, that tackles these challenges head-on. Building on top of existing agent-oriented and distributed systems architectures, RADAR provides a pluggable framework for integrating "specialists" that collectively augment a user's ability to handle complex tasks. Such specialists complement the capabilities of existing desktop environments, applications, and file systems, automating routine (but often complex) tasks programmatically. New specialists can be added or removed at any time. Moreover, learning is a core capability: over time specialists adapt to the needs and preferences of users.

While RADAR is the product of a large number of cooperating researchers, developed over the past three years at Carnegie Mellon, in this paper we focus specifically on the design of its architecture and the ways in which that architecture supports key engineering properties of compositionality, extensibility, and integration with existing applications and services. We also describe the current implementation and outline recent empirical results of RADAR's

effectiveness in supporting a class of crisis management tasks.

2 Related Work

One important branch of related research is traditional approaches to artificial intelligence, which attempt to automate human-oriented activities such as medical diagnosis, hardware configuration, chess, and robotics. In most of these systems the goal is to have the AI system *replace* the human, and many of these systems have focused on very specific task domains (like chess or medicine). In contrast our work on a personal cognitive assistant attempts to *augment* human capability, and to do this for rather mundane (although often voluminous and complex) tasks like prioritizing email, or helping to manage one's calendar. Additionally, most AI systems have not investigated the engineering issues of developing a component-based approach, or integrating AI capability with legacy systems.

More closely related are other approaches to assisting users with tasks in familiar desktop environments. The Calo project, for example, has been investigating similar approaches [3]. Like RADAR, Calo provides an integration framework for learning-based task-specific components. RADAR differs from Calo in two respects. First, RADAR attempts to co-exist with off-the-shelf applications and data, such as Outlook, while Calo has taken the approach of reengineering standard desktop applications to work smoothly with it's task support. The advantage of RADAR is the ability to plug its capability into any desktop environment; the advantage of Calo is that reimplementation of standard applications provides better opportunities for close collaboration between them and the cognitive assistant.

Other work that attempts to help users with ordinary tasks comes out of the ubiquitous computing [1][16][19] While these efforts attempt to dovetail with existing infrastructure and applications, their primary focus is on the use of heterogeneous and pervasive devices to help users accomplish tasks more effectively.

Another closely related area is that of Agent-oriented Architectures [7][9][11][12][13][14]. Over the past decade there has been considerable interest in multi-agent systems, and middleware to support them. In particular, a number of architectural frameworks have been proposed, including AAS [6], Zeus [5], FIPA[10]. As described later, we build on top of agent-oriented architectures (and, in particular, FIPA), specializing the general notions of agents and coordination with the specific architectural structures that characterize the RADAR architecture.

3 Architectural Requirements for a PCA

The vision of a PCA is that of a smart assistant, that in some sense "understands" the user, helping out where needed and effective, but staying out of the way otherwise. Inherent in this view is the idea that a PCA should complement what a user normally does, and how a user normally does it. Although over time a user might adapt his behavior to rely more heavily on the PCA as he gains trust in it, the user should not be forced to do this.

Consider the following scenario. A busy user has loaded a PCA onto the desktop. At first the user notices little change to his normal way of working. However, exploring the PCA console, he discovers that he can activate a calendar assistant. After activating it, the user is prompted to identify some general preferences for things like what calendaring application he wants to use, what times to keep free on the schedule, cancellation policies, and the like. Since the user is wary of turning over control to any automated calendaring assistant, he decides to be conservative requesting that the assistant should schedule meetings only during the hours of 10-12 on weekdays, always confirming schedule changes before committing them, and it should never cancel or reschedule an existing meeting. As time progresses he notices that the calendaring assistant has been able to correctly identify email messages that relate to scheduling requests, and to suggest reasonable scheduling actions. Based on positive experience, he decides to let the assistant do it automatically. Over time, he discovers that the assistant can do more and more: it learns his desires for canceling meetings (e.g., preferring to move subordinates' meetings before those with his boss); it learns that when the user goes on vacation or business travel, email should be sent to people with whom he has regular meetings to let them know, etc. Quite happy with this capability he continues to let it do more, confident that it is learning how he would like it to be done, and asking for permission before attempting anything radically new.

From an engineering perspective this vision implies three essential requirements for a PCA:

1. **Compatibility:** The services and assistance provided by a PCA should co-exist with the capabilities of current legacy applications and services. The user should not have to abandon old ways of doing business, or learn to use new applications with different interfaces. While additional capability provided by a PCA will necessarily require some additional forms of user interaction, these should supplement, not replace, existing forms of interaction. This implies compatibility not only with applications, but also with information sources as well. For example, email messages are often an important stimulus and information source for an assistant (for example, signifying the need to start a new task). Understanding email messages, written in a natural language, and stored in standard email repositories (e.g., Imap), is essential.

2. **Extensibility:** It should be possible to incrementally augment the capabilities of the PCA. For example, if some new form of task assistance becomes available, it should be easily pluggable into the existing system, adding new capability without disrupting the old, and dovetailing with existing assistance provided by the PCA. One can even imagine a marketplace for personal task assistance in which different forms of the same kind of assistance might be purchased at different price-quality points
3. **Adaptability:** The system should conform to the user, learning new opportunities for assisting the user, and inferring appropriate behavior based on how users carry out their tasks. Learning should apply to a wide variety things, including prioritization of tasks (e.g., helping a user focus on the important things), policies for interaction with others (e.g., deciding who should have access to certain kinds of information), clusterings of related activities (e.g., noticing that if action A is performed action B is usually also performed), interpretation of natural language (e.g., recognizing idioms that relate to task achievement), and many others.

In addition to these requirements, there are a number of other more-standard systems-oriented engineering qualities, such as robustness, availability, security, and performance. Indeed, the services provided by the PCA should have comparable quality attributes to today's mail systems, which tend to be available in a global setting, highly robust, secure, and reasonably efficient.

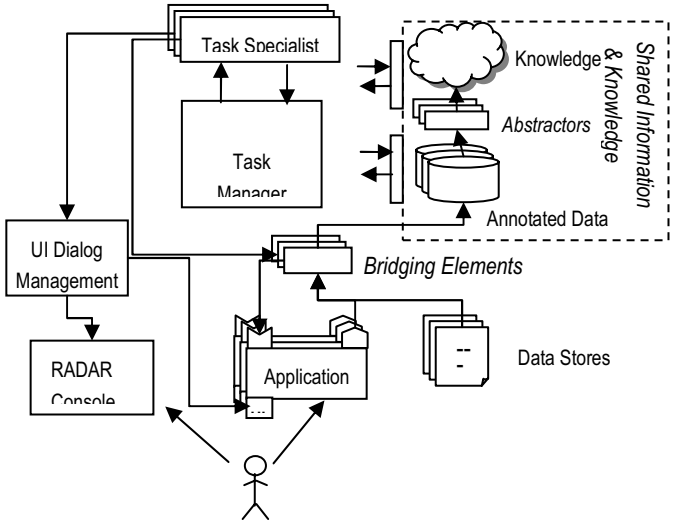


Figure 1. An Individual's Radar Architecture

4 THE RADAR ARCHITECTURE

To achieve these goals RADAR has adopted a layered runtime architecture, pictured in Figure 1, which depicts the architecture from the point of view of a single individual working in a personal RADAR space. We first give a high level overview, and then look in more detail at specific technical issues.

4.1 Overview

At the bottom layer are legacy applications, services, and data stores. Applications include things like email readers, web browsers, calendar managers, and the like. Data stores include documents stored in local and remote file systems, repositories of email, calendar information, contact lists, etc. Users interact with these in normal ways. Application APIs are used or written to allow RADAR to integrate with legacy application. For example, (**M**) interfaces inform RADAR of events that happen in the system (e.g., the user moves an appointment), that might trigger new tasks or learning by RADAR; Control (**C**) interfaces allow RADAR to make changes to the desktop space (e.g., to schedule a new appointment); User Interface embellishments (**UI**) allow RADAR to present information to the user in a manner the user is familiar with (for example, to display RADAR-proposed alternatives for a meeting on a user's calendar). In addition we add a RADAR Console, which provides a user with direct access to RADAR and its capabilities.

On top of these RADAR adds a layer of task assistance. This can be divided into four parts:

Task specialists: A task specialist (or just *specialist*) is a component that attempts to provide assistance for a particular kind of task, such as schedule management, web site updating, and routine email handling. The number and kind of specialists can vary from user to user, and over time for a single user as new specialists are added or removed from that user's RADAR space. Each specialist contains knowledge about how to conduct a particular task, and each contains a learning component that allows each specialist to adapt to the user with respect to preferences, preferred methods of doing the task, etc. It stores this learned knowledge in the shared knowledge base.

Task management: To coordinate the work of the specialists and to provide overall tracking and control of tasks is a task manager. The task manager comprises a number of logical services, including task dispatch (interacting with specialists to assign new tasks), task tracking (keeping track of high-level state of tasks – see below), task query (retrieving all tasks that match certain selection criteria), and task prioritization (keeping track of the relative priority of tasks).

Shared information and knowledge: To be effective, specialists and task management services must manipulate data in richer forms than is conventionally stored in today's desktop environments. For example, intelligent email assistance requires that key features of email messages are identified and classified. Similarly, calendaring information may need to be structured in higher-level ways than is natively stored by a calendar system. In addition, there is the need to represent knowledge representing high level

entities and relationships in the user's world. For example, social nets that determine what relationships a user has to other people, are stored in a knowledge base, and used by specialists and task management to determine security procedures (e.g., who are my friends), policies for actions (e.g. don't cancel a meeting scheduled by my boss), and general knowledge about the environment (e.g., what rooms are physically close to my office).

Bridging elements: to get information from the desktop level into RADAR space, requires certain bridging elements. There are several kinds of these. One kind transfers information from desktop space into RADAR space. These include categorizers and extractors that understand natural language to label and categorize the information from the desktop space. A second kind of bridging element takes information directly from legacy applications, through the **M** interface in Figure 1. These allow RADAR specialists to monitor activities performed directly by legacy applications, and to control those applications programmatically. (We discuss the differences between categorizers and extractors below.) The difference between such bridging elements and specialists is that, although they may both have knowledge specific to particular tasks, bridging elements are responsible for transforming native representations of data (such textual email) into task-oriented information (such as the existence of a new task), while specialists have knowledge of how to assist the user in *carrying out* the tasks.

4.2 Technical Details

To illustrate how information from the desktop space flows through RADAR, consider the arrival at John's desktop of an email from fred@a.com containing the text "I would like to organize a meeting with you and Melinda next Tuesday."

1. Categorizers and extractors take this information and annotate it with structural information such as the positions of names (Melinda), dates (Tuesday next), and that the message is to do with organizing a meeting. A new task for organizing a meeting is also constructed by the extractor and sent to the task manager for dispatch. The task is stored as annotated data, with a reference to the original message.
2. Abstractors take the structured information and annotate it with knowledge. For example, it notes that Melinda is John's boss, and that John prefers meetings on Tuesday to be in the morning. The knowledge is placed in a knowledge base, which can be queried by other components.
3. The Task Management component notices that a new task to organize a meeting has been proposed, by triggers in the task database. It locates a task specialist that is responsible for managing John's calendar, and assigns the task to it.
4. The Task Specialist attempts to find suitable slots on John's calendar for the meeting to take place. This might involve confirmation with John, which will be done through UI dialog management, and by placing the new meeting on John's calendar (through the control (**C**) interface of his calendaring application in Figure 1).

One important requirement of the flow of information through RADAR is the need to manage interaction with a user of RADAR. If a part of RADAR wishes to communicate the user, it should only do so at appropriate times. For example, in step 4 above, a calendar management specialist might want to confirm an appointment. If it immediately interrupts the user to request this, it might interrupt the user who was working on another task, causing him to lose context. For real world use, this will most likely make Fred less efficient because he is constantly being interrupted. Thus, all Radar-initiated interaction with the user is mediated through the UI dialog management component, which manages when and how a user should be interrupted. The UI Dialog Manager learns when and whether to interrupt the user [2], based on knowledge of the user's focus and interruption policies. The UI dialog might present this information via the RADAR console, by RADAR-specific UI embellishments in legacy applications (the **UI** interface in Figure 1), or other interfaces using techniques similar to those described in [8].

A central notion of RADAR is the idea of a *task.* A task is a unit of work that the user cares about that can be automated (or partially automated) by RADAR. The unit of work could be assigned to a single task specialist, or it may involve the coordination (through a task planner) of multiple task specialists. Such a planner would itself be implemented as a specialist.[1]

A key component in managing tasks in RADAR is the Task Management facility, which is responsible for the following task-related duties:

1. *Task Dispatch and Specialist Registry*. The Task Manager acts as a directory facility for matching particular types of tasks to specialists that can be used to automate them. The task manager is then responsible for assigning tasks to specialists, and also indicates to specialists when to suspend or stop particular tasks (for example, at the user's behest, or because the task is no longer valid, or another task is more important). In addition to dispatching tasks, this component is also responsible for detecting the liveness and availability of particular specialists.

[1] The current implementation contains only a rudimentary planner. See also Section 7 on future work.

2. *Information privacy and access control.* In many instances, users of RADAR will want to restrict information that is made available to others. For example, a user may not want to make details of their schedule available to others, and may not want RADAR to automatically schedule meetings if they are requested from certain people. While the knowledge particular to this lives in the shared knowledge base, the Task Management facility is responsible for ensuring that the user's preferences are met.

3. *Inter-Radar Communication.* To ensure privacy and access control, the task manager mediates RADAR's communication with other users. This gives RADAR the opportunity to also determine how best to communicate with a particular person. For example, if that person has their own instance of RADAR, then this component can contact their RADAR; if the person doesn't use RADAR, it chooses alternative ways to communicate (e.g., email, IM, cellphone text message).

4.3 Satisfying the Requirements

This architectural design, addresses the three critical requirements for a PCA outlined in Section 3. First, *compatibility* is supported through the layered architecture, which augments existing applications and data without replacing them. While applications must be modified in small ways to provide monitoring and control capabilities from the RADAR layer, and have certain user interface enhancements, by and large they remain unchanged.

Second, *extensibility* is supported through a component-oriented architecture in which task assistance is provided by modules (specialists) that can be incrementally added to or removed from the RADAR ensemble, simply by registering or deregistering them.

Third, adaptability is supported in several ways. The RADAR console allows a user to specify policies directly. In addition each specialist and the task manager provides its own learning capabilities, as outlined above, which coupled with a shared knowledge base, and reusable mechanisms for learning (e.g., extractors and abstractors) allow RADAR to adapt over time to a user's needs.

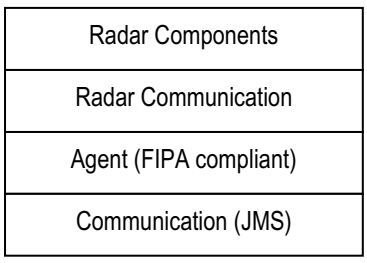


Figure 2. RADAR Layered Implementation Architecture.

5 IMPLEMENTATION

RADAR is designed to run as a server-oriented system in which the main capabilities are provided in stable environments that communicate with a user's personal desktop or mobile platform. As such, RADAR task management and assistance operates much like email servers, communicating with mail clients, but accessing mail stored in a stable way on externally-maintained and robust servers. This design helps provide the needed availability required to support a continuous, globally accessible service.[2]

The implementation of RADAR is based on a layered use of existing technology, illustrated in Figure 2. At the lowest implementation layer are standard middleware services for distributed systems. Specifically, we used Java Messaging Services (JMS), which provide a network-wide service for sending messages between components. The interface to this layer provides an API that hides details of the middleware, supporting basic communication mechanisms for remote method invocation and publish-subscribe.

At the next higher level is an agent-oriented architecture, which provides a virtual agent layer. The agent layer provides a FIPA-compliant API that defines the types of messages that can be used to exchange information between components, and specifies the building blocks on which more sophisticated communication protocols are built.

The RADAR communication layer specializes more general agent-oriented paradigms, defining specific protocols for communication between specialists and the task management services, interaction with the knowledge base, registration and invocation of the bridging elements, and the RADAR console for interacting with the user. This layer defines the rights and responsibilities for specialists, bridging elements, shared data and knowledge through a set of interface specifications.

Building on top this architectural infrastructure, RADAR V1.0 includes the following components and capabilities:

- Extractors and categorizers that understand general language terms such as places and names, but also task-specific information such as scheduling constraint requests.
- Specialists that assist the user with:
  - Managing a company website, by correcting errors in people's information based on emails, and publishing the updates to a website ;

[2] Although targeted for server-oriented deployment, RADAR also permits client-oriented configurations in which more of the functions run on the client side. (Indeed, our initial implementation used this configuration).

- Managing a schedule, which includes scheduling appointments and finding spaces where meetings can take place,;
- Preparing work summaries, or briefings, that can be sent to superiors, by learning which emails and tasks are more important and helping the user to summarize this information;

- Integration with Microsoft Outlook, for organizing users' email and as a user interface for controlling some aspects of Radar. For this, Outlook's COM interface was used to provide the **UI**, **C**, and **M** interfaces, providing natural extension points from which to integrate Outlook with RADAR. While the interface to each legacy application will differ, our experience in another project [17] suggests that wrapping applications to provide the necessary interfaces is possible, and is becoming increasingly easy with modern applications. We are, however, limited to facilities provided by the interfaces of applications.

6 Evaluation

While designed to promote the requirements outlined in Section 3, a critical question is how well RADAR performs in a live setting, and how effective is learning in automating everyday tasks. To investigate these questions, the RADAR team carried out extensive experimental evaluation. The details of the evaluation are reported by others in [18]; here, we give a summary.

A controlled crisis scenario was constructed: a week before a conference is due to start, a building that was to be used to host the conference becomes unavailable. Subjects in the experiment were asked to reschedule the conference sessions in alternative rooms, manage the constraints on speakers who have already booked travel assuming the previous scheduled, and brief the program committee on progress, and stay current with arriving email. The crisis is exacerbated by the fact that the primary conference organizer is unavailable to help, although he used RADAR to help organize the conference initially.

Two instantiations of RADAR were used in the experiment:

1. *Without any information learned about the conference.* This tested the effect of RADAR without it having prior specialization to crisis situation. It does not know, for example, whether a particular message concerns a meeting. There were 31 subjects in this group.

2. *With preloaded knowledge* learned as if RADAR had been used by the conference organizer to organize the conference initially. Extractors and categorizers had been trained so that they could recognize task-related email. This group contained 47 subjects.

Test subjects engaged with the conference planning crisis scenario during two sessions of 90 minutes. In this test, learning was shown to have statistically significant positive influences on several system-wide performance metrics.

7 Conclusion and Future Work

Realizing the vision of a fully-featured PCA is a formidable task that will take significant advances in research and engineering to achieve and demonstrate. In this paper we describe first steps toward realizing that vision. The key to this is the design of a pluggable architecture that permits extensibility and adaptability, while remaining compatible with existing desktop services and applications. Our implementation of RADAR v1.0 and its performance on tests are encouraging: it demonstrates that an integrated task management system can be implemented and be effective even in handling highly-stressful situations and with complex tasks.

However, considerable work remains to be done to fully realize the potential of a PCA. First, is the discovery of new forms of learning that can help the user. With respect to the crucial capability for learning to provide better task management, for example, we are now exploring the possibility of learning such things as how to order tasks according to their importance. In particular, we think it should be possible to take into consideration such things as the type of task, the history about how quickly similar tasks have been completed, and who originated the task, to predict the importance of task when it enters the system.

Second, is representation and assistance with complex tasks. In many cases such tasks will require planning as well as learning. This requires research on combining learning and planning in complex tasks, as well as implementation mechanisms to make such capabilities available as common services to RADAR specialists.

Third, is the need to provide a user with greater transparency into the workings of RADAR. While RADAR can provide demonstrable value-added to users, at present it is unable to explain its actions in a way that allows the user to understand exactly what RADAR has learned and why it believes what it does. Partly this is due to the nature of statistical machine learning, but it also related to the enhancement of specialists so that as a part of their normal functionality they can explain their task understanding and actions in user-oriented terms. Fourth is the effective interaction between multiple RADAR spaces. At present RADAR uses standard modes of communication (such as email) to communicate between users. But it should be possible to do much better when two users both have a RADAR working in their environment.

ACKNOWLEDGEMENTS

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010. Any opinions, findings and con-

clusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA or the Department of Interior-National Business Center (DOI-NBC).

REFERENCES

[1] First International Workshop on Computer Support for Human Tasks and Activties, Co-located with Pervasive 2004, Vienna, April 2004.

[2] D. Avrahami and S. Hudson, QnA: Augmenting an instant messaging client to balance user responsiveness and performance. *Proceeings of the ACM Conference on Computer Supported Cooperative Work (CSCW), pp. 515-518, Jan. 2004.*

[3] Berry, P., Myers, K., Uribe, T., and Yorke-Smith, N. Task Management under Change and Uncertainty. Constraint Solving Experience with the CALO Project. Proc. CP'05 Workshop on Constraint Solving under Change and Uncertainty. Spain, 2005.

[4] F.Bellifemine, F.Bergenti, A.Poggi, G.Rimassa, P.Turci, Middleware and Programming Support for Agent Systems. Proc. 3rd International Symposium "From Agent Theory to Agent Implementation", Austria, 2002.

[5] J.C.Collins, D.T.Ndumu, H.S.Nwana, L.C.Lee. The ZEUS agent building toolkit. *BT Technology Journal* **16**(3), 1998.

[6] P. R. Cohen, A. Cheyer, M. Wang, S. C. Baeg, OAA: An Open Agent Architecture, AAAI Spring Symposium, 1994.

[7] S. Cranefield, M. Purvis, An agent-based architecture for software tool coordination, in *the proceedings of the workshop on theoretical and practical foundations of intelligent agents*, Springer, 1996.

[8] A. Faulring and B. Myers, Enabling rich human-agent interactions for a calendar scheduling agent. *Proceedings of the Conference on Human Factors in Computing Systems Extended Abstracts (CHI)*, Portland, Oregon, May 2005.

[9] T. Finin, J. Weber, G. Wiederhold, et al., Specification of the KQML Agent-Communication Language, 1993.

[10] The Foundations for Intelligent Physical Agents (FIPA). http://www.fipa.org.

[11] S. Franklin, A. Graesser, Is it an Agent or just a Program? A Taxonomy for Autonomous Agents, in: *Proceedings of the Third International Workshop on Agents Theories, Architectures, and Languages*, Springer-Verlag, 1996.

[12] M. R. Genesereth, S. P. Ketchpel, Software Agents, Communications of the ACM, Vol. 37, No. 7, July 1994.

[13] B. Hayes-Roth, K. Pfleger, P. Lalanda, P. Morignot, M. Balabanovic, A domain-specific Software Architecture for adaptive intelligent systems, IEEE Transactions on Software Engineering, April 1995.

[14] T. Khedro, M. Genesereth, The federation architecture for interoperable agent-based concurrent engineering systems. In *International Journal on Concurrent Engineering, Research and Applications*, Vol. 2, pages 125-131, 1994.

[15] Y. Shoham, Agent-oriented programming, Artificial Intelligence, Vol. 60, No. 1, pages 51-92, 1993.

[16] J.P. Sousa, Scaling Task Management in Space and Time: Reducing User Overhead in Ubiquitous-Computing Environments. Ph.D. Thesis, Carnegie Mellon University School of Computer Science Technical Report CMU-CS-05-123, 2005.

[17] J.P. Sousa, V. Poladian, D. Garlan, and B. Schmerl. Capitalizing on Awareness of User Tasks for Guiding Self-Adaptation. *Proc. the 1st International Workshop on Adaptive and Self-managing Enterprise Applications at CAISE'05.* Portugal, 2005.

[18] Steinfeld, A., Bennett, R., Cunningham, K., Lahut, M., Quinones, P.-A., Wexler, D., Siewiorek, D., Cohen, P., Fitzgerald, J., Hansson, O., Hayes, J., Pool. M, and Drummond, M. The RADAR Test Methodology: Evaluating a Multi-Task ML System with Humans in the Loop. Carnegie Mellon University School of Computer Science Technical Report CMU-CS-06-124, CMU-HCII-06-102, May, 2006.

[19] Want, R.; Pering, T.; Danneels, G.; Kumar, M; Sundar, M.; and Light, J., "*The Personal Server: changing the way we think about ubiquitous computing*", Proc. of Ubicomp 2002: 4th International Conference on Ubiquitous Computing, Springer LNCS 2498, Goteborg, Sweden, 2002.

Updating styles challenge the updating needs within component-based software architectures

Mourad Oussalah, Dalila Tamzalit, Olivier Le Goaer* Abdelhak-Djamel Seriai**

*LINA - CNRS FRE 2729, 2 rue de la Houssinière, Nantes, F-44000 France
{mourad.oussalah,dalila.tamzalit,olivier.le-goaer}@univ-nantes.fr
**G.I.P - Ecole des Mines de Douai, Douai, F-59500 France
seriai@ensm-douai.fr

Abstract

The modification of the elements of a component-based software architecture is necessary to meet new needs and new situations usually unpredictable. All these modifications are summarized as updates, and cover the usual notions of evolution, adaptation, customization or reconfiguration. There are several solutions to update architectures. A solution often requires a strong expertise and is stylized because it privileges a strategy rather than another. Since there are recurring problems and recurring solutions, we propose in this paper the updating styles as pseudo-formal entities, to specify the stylized expertises, to use and re-use them. Later, we aim to build an environment based on a library of updating styles, to assist architects in modifying their component-based architectures.

1 Introduction

Component-based approach is regarded as a paradigm which fills the lacks of the object paradigm [11]. Component-based proposals thus appeared to answer requirements engineering, specifications, coding and systems modeling. However, like any software, component-based architectures have to be necessarily updated in order to satisfy various new requirements, often unpredictable. By update we summarize all modifications made to a software system, at the various stages of software lifecycle, especially at the maintenance stage. It includes common notions like *evolution, adaptation, customization or reconfiguration*. In this paper, we deal with the management of the update needs within component-based architectures. Indeed, there are recurring update needs and their recurring dedicated solutions, nevertheless the choice of the most suitable solution and its implementation require a strong expertise. To our knowledge, this update expertise (or know-how) within component-based architectures was not captured, represented and re-used like was design expertise by means of the architectural patterns and styles [5, 6, 10, 7]. We propose in this paper a new concept, the *updating style*, that we define as the combination of a proven competence and a stylized solution to an update need. A proven competence ensures that the update needs will be correctly filled and a stylized solution means that some concerns will be privileged rather than others.

In this paper, we initially expose the reports which justified this new approach. Then, we present the context of our work: the component-based architectures and the update of this type of architectures. Next, we present the updatings styles definition and concepts to lead on a three-level model before we conclude and present our prospects for this work.

2 Motivations and objectives

The problematic we are interested in results from several observations of software updating:

Updating is a complex work: in increasing complex architectures solutions are seldom commonplace: it is necessary to be able to identify what tasks must be carried out and in what order. Moreover, they must respect both integrity and quality constraints. Without dedicated tools it is the architect's expertise which determines the success or the failure of updates.

Updating is a recurring work: some update concerns are shared by a great number of architects: interoperability, reconfiguration, upgrade, adjustment to the available resources, etc. Others only affect an architect for his particular project. In all cases, the same tasks are regularly encountered and necessary.

Updating is a stylized work: consciously or unconsciously, the architect chooses a solution rather than another to solve its update problem. These different way of update converge toward the same objective but may privilege different concerns as quality, simplicity, speed or costs. Indeed, there are often several possible strategies to achieve a goal: they are stylistic properties of the update.

Updating is a reusable work: there are recurring update needs and there are recurring proven and stylized solutions to answer them. Expertises must be captured and represented to be re-used by architects, by the same way that elements of designs are re-used. That is a means of reducing the risks, the efforts and, well on, the costs.

We are convince that the capitalization of stylized update expertises is significant, for use and re-use purposes. Besides, we believe in the distribution of predefined update expertises, like are COTS[1] components for architectural builds. More generally we think that COTS products and predefined update expertises can be regarded as the two orthogonal dimensions which have to be combined to allow the real "programing in the large", and nowadays the "programing in the world". For this purpose, we propose the *updating style* (U.S) as an encapsulation of a stylized update expertise. From now on, our work consists in defining, specifying, formalizing, and modeling the updating styles and their relationships.

3 Component-based architectures: the updating challenge

We present in this section the fundamental concepts and language of component-based architectures (CBA) and the hypothesis we have done to update such architectures.

3.1 Architectural concepts and description langage

To update a component-based architecture consists into update its constituent elements: the architectural elements. It is commonly admitted by the research community working on architectural description languages that there are at least four fundamental architectural elements [9]: Component, Connector, Interface, Configuration. For a clear definition of these elements, refer to [1].
Architectural description languages (ADL) emerge as notation support for the architectures models [8]. There is a large variety of ADLs emerging from various academic and industrial groups. The coverage of the ADLs is broad, each one has its characteristics and aims at precise concerns of architectural design[9] and there is no ADL which is best appropriate for all the possible purposes, but they agree nevertheless on the common minimal set of architectural elements we have mentioned. Unfortunately, ADLs partially deals with update needs. Their update mechanisms are limited and update strategies are lacking. Nevertheless, ADLs have sometimes extensions to be able to describe the dynamics of architectures, like Dynamic ACME[4] for instance, but the update expertise still remains a difficult work which falls to the architect.

[1] commercial off-the-shelf

3.2 Reification of architectural elements

We consider that any element which can be modified must be reified, to be able to manage its update. For example, it is the case of the four aforesaid fundamental architectural elements. Hence, they are considered as first-class entities. In order to ensure this reification, we consider that a component-based software architecture must be described on three abstraction levels (Fig 1):

- M2: is the level where all the architectural elements are specified. It constitutes the highest abstraction level called *Meta-Architecture level*.
- M1: is the *Architecture level* and allows the description of an architecture by using one or more architectural elements of the Meta-Architecture level.
- M0: lastly, the *Application level* is an instantiation of the Architecture level. It represents an application at the run-time.

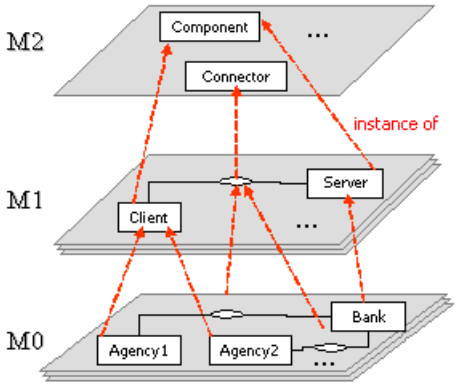


Figure 1. Abstraction levels within component-based architectures

Update needs and problems exist on each abstraction level even if the majority of the research works mainly focus

on the M1 and M0 levels. With our model, at each abstraction level, the entities potentially affected by updates can be managed and our updating styles will require this significant property. Last, notice that the management of the inter-level impacts is also necessary to preserve the global architecture coherence but is out of scope of this paper.

4 The emergence of the updating styles

In order to make implicit architectural knowledge explicit for re-use, we propose the updating style (U.S) as a pseudo-formal entity, able to apprehend in a complete way updating problems while being sufficiently formal to be able to be exploited by various tools. Thus, we switch from an only "contemplative" use of process patterns [3] to a more "productive" use of updating styles.

4.1 Updating styles definition

Alexander [2] shows that although every building is unique, each may be created by following a collection of general patterns. Indeed, although every updating is unique, each may be created by following a collection of general updating styles. In other words, an updating style is a quite general and stylized solution to a common update problem or issue, from which a specific solution may be derived. Thus, we define an updating style as follows:
– *An updating style characterizes a family of updating processes sharing a recognizable characteristic, all conforming to a particular manner to update architectural elements. It captures a proven and stylized expertise which meets a recurring update need within component-based architectures, according to particular concerns, and consists in a declarative description of the update stages to be followed, that must any person using the style conforms to.* –

4.2 Updating styles specification

An U.S offers a service to update a component-based architecture. The context of its use is described on a semi-formal way within the *header* part. If the U.S has a way to realize its update service then it has a competence, described in a formal way within the *competence* part. These two parts form the specification of U.S (Fig 2).

The header part contains meta-data such as the name and the goal, but also architectural data such as the required inputs and expected outputs and the constraints to be respected from the beginning to the end of the update procedure. These informations make possible to classify, store and retrieve the U.S in an (semi)-automated way. For instance, thanks to the architectural data, the matching of an

Figure 2. The two specification parts of an updating style

U.S and a given architecture quickly indicate us if the U.S is suitable or not. The meta-data can indicate if the style is the most suitable.

The competence part is optional. If it is specified, the U.S is *concrete*, else the U.S is *abstract*. The competence part contains a declarative rules-based expertise. Rules will be recursively matched with a given architecture to produce a specific update process that the architect will have to follow. A rule can call another U.S or else call an update task that can be sometimes associated to an ad-hoc implementation unit. This property make possible to execute the U.S using the best current update techniques available on the considered architectural level (M2, M1 or M0).

4.3 U.S relationships: specialization & composition

U.S can be linked with two kinds of relationships: the specialization (inheritance) and the composition.

Specialization: the inheritance mechanism make possible to specialize exiting U.S to build new ones, for more precise update situations. The inheritance mechanism affects the header and competence parts and allows to redefine and extend the encapsuled data. The resulting inheritance tree addresses a given business domain (Fig 3). We currently privilege the top-down approach i.e the classification of each new U.S starting from the root of the specialization tree, but we will soon interest in the bottom-up approach i.e the factorization of existing U.S to create new categories.

Composition: we do not consider all updating styles as monolithic blocks. Indeed, a key idea of our proposal is to assembly U.S to build increasingly complex updates, thus maximizing the re-use concept. We believe in a full-style solution: a set of minimal U.S - the primitive ones - make possible to build new ones by their composition and so on. It is a hierarchical model to build complex update expertises.

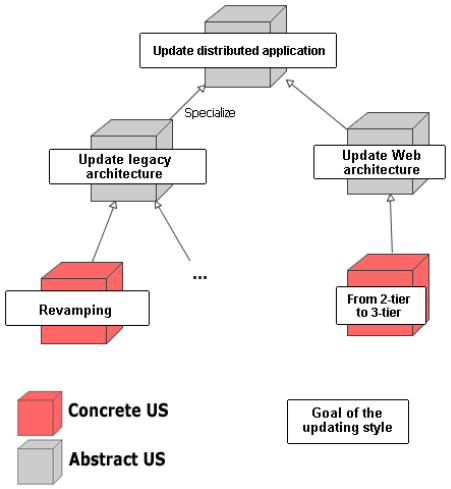


Figure 3. Example: specialization tree to update distributed applications

4.4 A three-level U.S model

Like for CBA (section 3.2), we have defined a three abstraction levels model for the reification of our updating styles. Due to place restriction we cannot detail this model in this paper but you can find the survey of these levels below:

- M2: *Meta level*, one finds there the U.S fundamental concepts like the header and competence parts.
- M1: *Style level*, contains U.S descriptions, using the concepts of the meta level.
- M0: *Process level*, the U.S are instancied as a specific sequence of update tasks. Moreover, some tasks are linked to dedicated implementation units and hence can be executed.

We currently discuss the representation's formalism for each level of this model. Nevertheless we have noticed that the reification of the U.S within the component-based paradigm make our model reflective. Indeed, in this case, the U.S could be modified (updated) by other U.S, as well as any other component-based architecture.

5 Conclusion and perspectives

Encapsulate stylized update expertise is a reuse-driven approach necessary to deal with one of the most critical aspect of the software engineering, where the work from scratch is seldom recommended. We have defined in this paper the updating style as a pseudo-formal first-class entity which is more productive than a process pattern, intended for people in charge of developing and maintaining CBAs. We believe in the use of predefined software products and also in predefined software processes like updating styles, to support a real large-scale software industry. Thanks to their standardized description, the U.S can be exploited by tools to be transformed, checked, requested and still more. Besides, we have planned to build an environment based on a library of updating styles, named USE (Updating Style Environment), to actively integrate the U.S during the architectural lifecycle. The USE's engine will be a transformation and query processor coupled with a navigation language to browse the inheritance and composition trees, in order to classify, store, retrieve and execute U.S according to the architect's needs. A main objective of USE is to enable non-skilled architects to update their architecture accurately and skilled architects to accelerate these updates. In both cases, update will be made easier, with a lower risk and cost.

References

[1] T. K. Adel Smeda and M. Oussalah. Academic component models. In *Component engineering : Concepts, techniques and tools*, pages 45–86. Vuibert Informatique, 2005.

[2] C. Alexander. *The Timeless Way of Building.* Oxford University Press, 1979.

[3] Ambler. *Process Patterns – Building Large-Scale Systems Using Object Technology*. Cambridge University Press/ SIGS Books, 1998.

[4] D.Wile. Using dynamic acme. *In Proceedings of the Working Conference on Complex and Dynamic Systems Architecture*, Australia, December 2001.

[5] R. J. J. V. Erich GAMMA, Richard HELM. *Design Patterns : Elements of Reusable Object-Oriented Softwares*. Addison Wesley, 1995.

[6] H. R. P. S. F. Buschmann, R. Meunier and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley and Sons, 1996.

[7] D. Garlan. What is style? In *Proceedings of the Dagstuhl Workshop on Software Architecture*, Saarbruecken, Germany, February 1995.

[8] P. Kogut and P. Clements. Features of architecture description languages. *Proceedings of the Software technologie Conference, Salt Lake City*, April 1995.

[9] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description langages. *IEEE Transactions on Software Engineering, Vol. 26*, 2000.

[10] R. T. Monroe and D. Garlan. Style based reuse for software architecture. *Proceedings of the 1996 International Conference on Software Reuse*, April 1996.

[11] C. Szyperski. *Component Software, Beyond Object-Oriented Programming (Second Edition)*. Addison-Wesley, ACM Press, 2002.

Verifying a Software Architecture Reconstruction Framework with a Case Study

Seonah Lee and Sungwon Kang
SOftware Technology Institute
Information and Communications University
517-100, Dogok, Kangnam, Seoul, Korea 135-120
{salee, kangsw}@icu.ac.kr

Abstract

In this paper, we propose a framework for reconstructing software architecture from a software system. The framework consists of the steps to reconstruct software architecture for modifying source code. The framework can be used for developers to analyze source code and reconstruct software architecture for modifying the code.

In addition, we perform a case study by analyzing Reflexion Model, an open source code, to verify that the framework can be effectively applied to software architecture reconstruction. For that, we evaluate several tools in order to select those that support our reconstruction. Then we verify the framework and report several issues discovered with the case study. By applying the issues encountered in the case study to the framework, we have made our framework a more practical one that developers can easily understand and adopt.

1. Introduction

Software Architecture Reconstruction has become the main stream of the reverse engineering research area [1, 5]. Architecture reconstruction is a reverse engineering activity that aims at recovering lost design decisions. Robust and clear software architecture is often the key determinant of the success or failure of many software projects [5]. Software architecture reconstruction is especially important in such cases. Some researchers have developed tools for software architecture reconstruction [5, 7]. Other researchers, realizing that software architecture reconstruction is not as simple as traditional reverse engineering of source code, have suggested their frameworks for software architecture reconstruction [4, 7].

However, there is no systematic and concrete guideline for reconstructing software architecture in order to modify source code. Specially, open source systems often have no accurate architecture documentation, and no architects to interview. Also, analysts may have no domain knowledge. Furthermore, the reconstruction of software architecture is not only to generate simple architecture views [2], but also to discover the design decisions, and to provide information for redeveloping the software. In order to understand the software architecture, the developers should know the domain model and the technology of the software, as well as its quality attributes and functional requirements. Hence we propose a software architecture reconstruction framework for modifying source code, and verify the framework by analyzing a software system according to the prescribed steps.

The remainder of the paper is organized as follows: Section 2 reviews the previous studies. Section 3 defines the steps to reconstruct software architecture for modifying open source. Section 4.1 evaluates tools that support the software architecture reconstruction steps. Sections 4.2 and 4.3 conduct a case study of Reflexion Model, an open source program. Section 5 reports the verification result from the case study. Finally, Section 6 summarizes the contributions of this paper and describes future research directions.

2. Related Works

In this section, we consider the previous research on software architecture reconstruction. Section 2.1 surveys the existing frameworks, which define the steps for reconstructing software architecture. Section 2.2 examines several case studies, which offer lessons learned from the analysis of software systems

2.1 Frameworks for Software Architecture Reconstruction

Two research groups have proposed interesting frameworks for software architecture reconstruction. One group, Nokia, suggested Symphony, the reference framework to compare all activities related to software

architecture reconstruction. The other group, SEI, suggested QADSAR, the framework defining the steps to reconstruct architectural view from source code.

Symphony [4, 5] outlines a view-driven software architecture reconstruction. Symphony consists of two phases. The first phase analyzes the problem for which architecture is needed and defines viewpoints and their mapping from source code. The second step extracts and analyzes information, applies mapping and creates views. Symphony provides a reference framework to find and demarcate research problems in software architecture reconstruction. However, it does not provide a practical guideline to reconstruct software architecture for specific situations [3].

The QADSAR approach [6, 7] has five steps: scope identification, source model extraction, source model abstraction, element and property instantiation, and quality attribute evaluation. QARSAR is valuable because it is a systematic way to introduce a quality attribute driven perspective to software architecture reconstruction. However, QADSAR does not reflect industrial experience in which developers analyze existing source code similar to their target system before developing a new one. Thus, case studies that reconstruct software architecture [8, 9] should reflect a framework that reconstructs software architecture so that the framework could be applied more practically.

2.2 Case Studies of Software Architecture Reconstruction

Two case studies are particularly relevant to our research. The first one is a case study of Linux kernel [8], which distinguishes architecture in the developer's mind from architecture built-into a system and shows the differences between them. The second one is a case study of Apache [9], which shows how students who have no domain knowledge learned Apache software architecture in a course.

Bowman, Holt and Brewster [8] performed a case study on extracting architectural documentation from Linux kernel. Bowman et al.'s approach is as follows. First, examine the existing documentation. Second, group source files into subsystems, based on directory structure, naming conventions, and code comments. Third, extract relations between the source files. Fourth, determine relations among subsystems. Finally, use the relations to form software architecture of the system. Since these steps are useful to recover design decisions for modifying a software system without an architect to interview, we incorporated these steps into our framework for reconstructing software architecture.

Gröne, Knöpfel and Kugel [9] summarized the results of one-semester course of source code analysis on the structure of Apache 1.3. The students, who had no domain knowledge about the system, followed the steps below and presented the architecture successfully at the end of the course. The first step is to define the purpose of the analysis and explain key concepts of the system. The second step is to gather domain knowledge, understand the system and model the conceptual architecture of the system. The third step is to understand the function, the configuration and handling of the software product. The final step is to understand the implementation of the software product. We used these steps to establish our own framework for people having no domain knowledge of modifying open source.

3. A Software Architecture Reconstruction Framework (SAROS)

In this section, we outline our framework of software architecture reconstruction for modifying open source (SAROS). We applied the following principles to our framework. First, the base framework is Symphony because Symphony embraces both the preparation and the execution of the reconstruction. Reconstruction Preparation is required because developers understand problems of existing software and a mechanism of the software in this phase. Second, the reconstruction execution of our steps is based on QADSAR because QADSAR has the concrete steps for building architecture views. The third principle is to reflect the activities of case studies, such as Gathering Domain Knowledge, and Understanding the Configuration. These activities define more effective steps for analyzing a software system without having to interview architects.

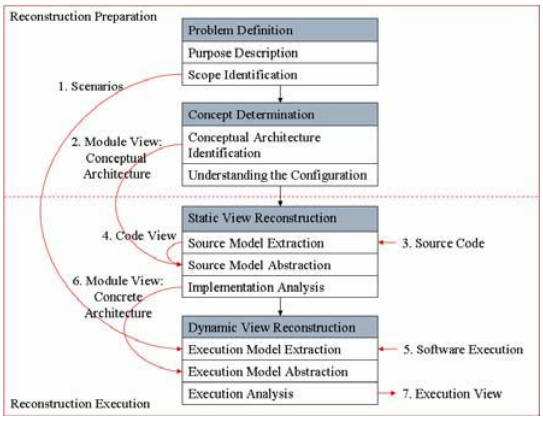


Figure 1. Our Software Architecture Reconstruction Framework for Modifying Open Source (SAROS)

Figure 1 depicts our Framework, SAROS. Steps for reconstructing software architecture are grouped under Reconstruction Preparation and Reconstruction Execution. Through the process, an analyst reconstructs software architectural views and determines what parts to modify and how to modify them. In the framework, an output of Reconstruction Preparation becomes an input of Reconstruction Execution. A scenario, describing a current status and situation to be improved in the Reconstruction Preparation phase, should be an input to the Reconstruction Execution phase, in order to determine which execution trace should be recorded in the Dynamic Reconstruction View task. Conceptual Architecture, defined in the Reconstruction Preparation phase, should be an input to the Reconstruction Execution phase, in order to customize a module view of an abstract source model in the Static View Reconstruction task. In Reconstruction Execution, Static View Reconstruction analyzes source code and generates a Code View and a Module View. The final result of Static View Reconstruction becomes an input of Dynamic View Reconstruction in order to aggregate detailed modules to abstract modules of Concrete Architecture. Finally, Reconstruction Execution produces the three views of Code View, Module View and Execution View [2, 10].

4. A Case Study: Reflexion Model Program

We performed a case study according to the framework in Section 3. The purpose of the case study is to verify if the framework can be applied to software architecture reconstruction. In addition, the case study provides guidelines for developers to understand and adopt our framework. We selected the Reflexion Model [13, 14, 15] as our target system because it was a moderate-size software and appropriate for a short-term case study. The software was implemented in Java 1.3 and Eclipse 2.1.1 on MS Windows. It consisted of 76 files and the total number of lines was 13,996 and the ratio of comments to code was 0.34.

4.1 Tool Selection for Our Case Study

We selected tools prior to conducting our case study. We chose the candidate tools by considering the environment of our target software. After reviewing candidate tools that could be applied to Java languages for static analysis, we selected Doxygen [16], Together [17], and Reflexion Model [14] for our case study. We excluded other tools because there were no free trial versions, a tool does not provide convenient graphic user interface, or because the selected tools provide more appropriate information.

After reviewing candidate tools that could be applied to Java languages for dynamic analysis, we selected only HPROF [17]. Dynamic analysis tools are very sensitive to their environment and more difficult to obtain than static analysis tools. Almost all advanced tools such as Jinsight [18], Shimba [19] and AVID [18] were unavailable to us. Other tools [20] generally do not support the platform and language of our case study, leaving us no alternative other than HPROF.

4.2 Reconstruction Preparation

The Reconstruction Preparation phase analyzes problems for which architecture is needed and defines the conceptual architecture of the system [4, 5, 8, 9]. Reconstruction Preparation has two tasks: Problem Definition and Concept Determination.

4.2.1 Problem Definition

The purpose of Problem Definition is to analyze and discuss the problem triggering the reconstruction with stakeholders. It consists of two steps, Purpose Description and Scope Identification.

① **Purpose Description:** In this step, an analyst studies a system to find a compelling reason to start a reconstruction. For that, we collected and read technical papers about Reflexion Model [14, 15] to learn how to operate the system. Then we prepared the environment and installed the software. Next we used the software to analyze a compiler translating TTCN code to C Code. Finally, in the experiment we discovered better ways to use the software [16].

② **Scope Identification:** In this step, the analyst identifies quality attribute scenarios which are grounds for finding the part to be reconstructed. For that, we prioritized the five improvement items and selected the first one: supporting hierarchical high-level modeling. Then we specified the use-case for supporting hierarchical high-level modeling in Figure 2.

- When defining high-level model, the user chooses a rectangle icon from the menu of the high level model. Then, the user draws a rectangle representing the upper modules embracing lower modules shaped in ellipses
- When defining map, the user adds the mapping relationships between the lower modules and the upper modules to the map.
- When computing map, the tool computes hierarchical mappings between several layers, and then presents the hierarchical model.

Figure 2. Scenarios

4.2.2 Concept Determination

The purpose of Concept Determination is to identify the architecture concept and a recovery strategy that are relevant to the problem. It consists of Conceptual Architecture Identification and Understanding the Configuration.

① **Conceptual Architecture Identification:** In this step, the analyst draws the conceptual architecture by studying technical papers. We drew the conceptual architecture (Figure 2) that represents the main functions of the Reflexion Program. The Extraction part parses the source code and extracts detailed source information from the code. The Modeling part supports user drawing the high level model, then converts the diagram into a text file. The Mapping part parses the text file that holds the information of the high level model and mapping rules. The Calculation part maps the relationships among source elements to the relationships among modules of high level model. The Presentation part shows the result of the calculation in the graphic form.

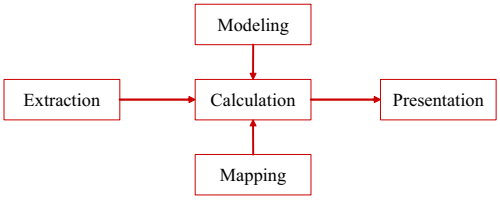


Figure 3. Conceptual Architecture of Reflexion Model

② **Understanding the Configuration:** In this step, we learned how to configure and administer the product and how to utilize the API. We had to study and experiment with the mechanism and graphic packages for Eclipse plug-in software before we started configuring. After that, we imported the source code as an Eclipse plug-in project into the Eclipse tool, and created the executable file of the Reflexion Model. For studying the API, we analyzed the main file and grasped the four classes connected to the user interface of the RM program. We caught the four classes as starting points of the analysis of the RM source code.

4.3 Reconstruction Execution

The Reconstruction Execution phase extracts and analyzes information, defines and applies mapping between source and conceptual model, and creates views. Reconstruction Execution has two tasks: Static View Reconstruction and Dynamic View Reconstruction.

4.3.1 Static View Reconstruction

The purpose of Static View Reconstruction is to analyze code and build the concrete architecture in a module view [2, 8, 9, 13, 14] from the source code. It consists of three steps: Static Model Extraction, Static Model Abstraction, and Implementation Analysis.

① **Source Model Extraction:** In this step, an analyst extracts source elements from available source code files. For that, we executed Doxygen that showed a directory structure, a list of files, classes, etc. Then, we edited the result and made Table 1, which includes the list of files. Next, we executed Together to grasp class diagrams. However, those tools did not extract call relationships among source elements, except for inheritance relationships. We noticed some tools extracting and abstracting the relationships at once. Thus we passed the extraction of concrete relationships from this extraction step to the next abstraction step and expected abstraction tools to help.

Table 1. Code Structure of Reflexion Model

/Doxygen/RM/jrmtool/sec/ca/ubc/cs				
jRMTool	compute	*.java (9)		
	eclipse	graph	spline	*.java (3)
			*.java (10)	
		gui	plugin	*.java(15)
			*.java (1)	
	map	*.java (1)		
	model	*.java (3)		
	struct	*.java (5)		
	util	*.java (10)		
sugiyama	algorithm	*.java (12)		
	model	*.java (4)		
	*.java (4)			

② **Source Model Abstraction:** In this step, the analyst should identify and apply aggregation strategies. We utilized the RM for showing the tentative concrete architecture that exhibits actual relationships among modules. We began with the directory structure, and modified mapping rules several times until we had captured the proper mapping between the conceptual architecture and source elements. Finally, we were able to create concrete architecture (Figure 4). A number on an arc between two modules represents the number of call relationships between those two modules. A difference between our conceptual architecture (Figure 3) and the concrete architecture (Figure 4) is that there are no definite call relationships between the Calculation module and the Extraction, Modeling, and Mapping modules in the concrete architecture; The symbols, ❶❷❸, denote no call relationships among the modules in Figure 4. Since our target source has graphic user interface and event handling mechanisms, the relationships between events are implicit.

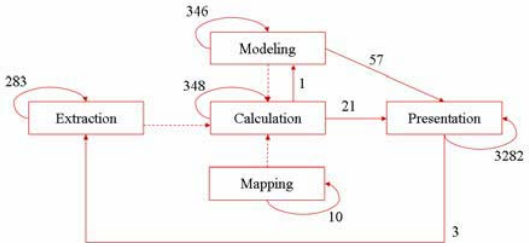


Figure 4. The Concrete Architecture of Reflexion Model

③ **Implementation Analysis:** In this step, the analyst elicits detailed information about the software product that is required for reconstruction. Our purpose for the modification was to enable the Reflexion Model to support hierarchical high-level modeling. For that, we had to identify parts of source code to be modified. Then we examined the source code and made a table of classes and functions which may contain important information.

4.3.2 Dynamic View Reconstruction

The purposes of Dynamic View Reconstruction are to analyze execution records of software and construct an abstract execution model in a C&C view [2, 11, 12, 13]. It consists of the three steps: Execution Model Extraction, Execution Model Abstraction, and Execution Analysis.

① **Execution Model Extraction:** In this step, the analyst extracts the real interaction of software elements from software execution. For that, we selected the third scenario described in Figure 2. Then we extracted a trace file from the execution of the function, "compute Reflexion Model," by using HPROF. As a result, we created a trace file with 150,796 lines. The trace data was not accurate in the call relationships between functions because HPROF just recorded execution order and did not mark when a function starts and finishes.

② **Execution Model Abstraction:** In this step, the analyst abstracts details of real interaction of software elements. We introduced the mapping rules of the "Source Model Abstraction." According to the mapping rules, we abstracted the trace data of the "Execution Model Extraction" to the upper level, as in Figure 6. A number on an arc between two modules represents the number of execution orders between two modules. Then we discovered that the modules that had showed no relations in the static analysis showed some relations in the dynamic analysis. We analyzed an execution order of Reflexion Model by importing and manipulating the trace data in MS Excel. As a result, we discovered that the implicit execution order of modules in a Static View became explicit, and we were able to see the flow from extraction to presentation.

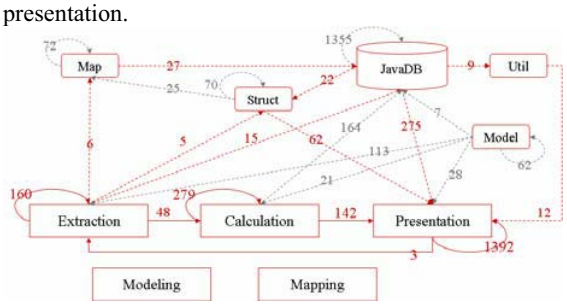


Figure 5. Dynamic View of Reflexion Model

③ **Execution Analysis:** In this step, the analyst elicits detailed information about the software system. First, we identified what we need to know in the source code, referring to the concrete scenario from the Problem Definition task, "Computing Reflexion Model." Then, we checked which parts are affected by the scenarios when we modify the calculating algorithm to support hierarchical mappings among several layers. Finally, we examined the source code and found the important parts related to the modification requirements.

5. Verification Results

We verified our framework with a case study in order to check that the framework is reasonable and feasible. Two phases, brought from Symphony, helped us arrange the tasks of software architecture reconstruction, without omitting additional tasks such as defining the purpose and studying the software system. A difference between our framework and Symphony is that our framework arranges the static analysis and the dynamic analysis sequentially while Symphony performs the two tasks as one. The case study assured us that the two tasks complement each other, but are not selectable for an analytical purpose. In some minor aspects, though the analysis tasks define an extraction step before an abstraction step, similarly with QADSAR, the tool supporting the steps has integrated the two steps. Thus, one of a user's main tasks is to define the mapping between source code and conceptual architecture. In addition, finding appropriate mapping between the model in our mind and the model in source code was done iteratively. With respect to tool application, we noticed that current tools were not mature enough to support analysis and abstraction of an execution flow of a software system. Therefore, we need to build such a tool in order to apply our framework to practical situations.

6. Conclusions

In this paper, we have proposed our framework for reconstructing software architecture from open source (SAROS). We utilized Symphony as the basic framework, and then incorporated QADSAR into it. We also integrated the procedures used in several case studies of software systems. SAROS can be a guideline for developers to analyze source code and reconstruct software architecture. Since some tasks of software architecture reconstruction are tedious and repetitive, we examined tools in our case study. Then we verified our framework (SAROS) through a case study of Reflexion Model, an open source software system. During the case study, we found and repaired several defects in the early versions of SAROS. In addition, the case study added a practical guideline that helps developers understand and utilize our framework by showing a concrete example of a software system. Developers can find a conceptual model, a concrete model and several views embodied in the case study.

We propose the following future research directions. First, we should survey tools more thoroughly and find a way to acquire several advanced tools. There are many reverse engineering tools, but the tools we used were limited to Java language. Many research papers report the result of reverse engineering in Java. However, the tools discussed in those research papers have not been published yet. This has limited our research. Second, we should perform more case studies in order to consolidate our framework. Our case study of Reflexion Model targeted a specific environment and a specific subject. The program was an Eclipse plug-in, analyzing source code and presenting a high level model. We need to consider other subject areas. Performing several case studies for a variety of subjects would prove that our framework is indeed widely applicable. In the long term, we want to develop our own toolset to reconstruct software architecture for modifying source code that is based on the SAROS framework.

7. References

[1] Bass, L., P. Clements and R. Kazman, *Software Architecture in Practice*, 2nd Ed., Addison-Wesley, 2003.

[2] Clements, P., F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, *Documenting Software Architectures: Views and Beyond*, Addison-Wesley, 2003.

[3] Stoermer, C., L. O'Brien, and C. Verhoef, "Practice Patterns for Architecture Reconstruction," *Proc. 9th Working Conference on Reverse Engineering*, pp.151-160, Virginia, USA, October 29-November 1, 2002.

[4] Deursen, A., C. Hofmeister, R. Koschke, L. Moonen, and C. Riva, "Symphony: View-Driven Software Architecture Reconstruction," *Proc. 4th Working IEEE/IFIP Conf. on Software Architecture,* pp.122-132, June 12-15, 2004.

[5] Deursen, A., and C. Riva, "Software architecture reconstruction," *Proc. 26th Int'l Conf. Software Engineering*, pp.745 – 746, May 23-28, 2004.

[6] Stoermer, C., L. O'Brien, and C. Verhoef, "Moving towards quality attribute driven software architecture reconstruction," *Proc. 10th Working Conference on Reverse Engineering,* pp. 46-56, 2003.

[7] Gorton and L. Zhu, "Tool Support for Just-in-Time Architecture Reconstruction and Evaluation: An Experience Report," *Proc. 27th Int'l Conf. on Software Engineering,* 2005.

[8] Bowman, T.R, C. Holt and N. V. Brewster, "Linux as a Case Study: Its Extracted Software Architecture," *Int'l Conf. on Software Engineering,* Los Angeles, May 1999.

[9] Gröne, B., A. Knöpfel, R. Kugel, "Architecture recovery of Apache 1.3 - A case study," *Proc. Int'l Conf. on Software Engineering Research and Practice,* pp. 87-93, Las Vegas, 2002.

[10] Lattanze, A.T., "The Architecture Centric Development Method," CMU-ISRI-05-103, February 2005.

[11] Rosso, C.D., "Performance analysis framework for large software-intensive systems with a message passing paradigm," *Proc. 2005 ACM symposium on Applied computing,* Santa Fe, New Mexico March 13-17, 2005.

[12] Riva C. and J. V. Rodriguez, "Combining Static and Dynamic Views for Architecture Reconstruction," *Proc. 6th European Conference on Software Maintenance and Reengineering*, IEEE Computer Society Press, pp. 11-13, Budapest, Hungary, March 2002.

[13] Murphy G. C. and D. Notkin, "Reengineering with Reflexion Models: A Case Study", *IEEE Computer* 30, 8, pp.29-36, 1997.

[14] Murphy G. C., Reflexion Models, *http://www.cs.ubc.ca/~murphy/jRMTool/doc/,* December 2003.

[15] Reverse Engineering with Reflexion Model: TTCN Translator, *http://www.cs.cmu.edu/~aldrich/courses/654-sp05 /tools/ keum-reflexion-05.pdf,* 2005.

[16] Doxygen, http://www.doxygen.org/

[17] Sun, HPROF, http://java.sun.com/developer/technicalArticles/Programming/HPROF.html

[18] Pacione, M. J., and M Roper, M Wood, "A Comparative Evaluation of Dynamic Visualisation Tools" *Proc. of 10th Working Conference on Reverse Engineering*, Victoria, BC, Los Alamitos, CA: IEEE CS Press, 2003.

[19] T. Systä, "Understanding the Behavior of Java Programs" *Proc. of the 7th Working Conference on Reverse Engineering*, Brisbane, Australia, November 2000.

[20] Xie T., and D. Notkin, "An Empirical Study of Java Dynamic Call Graph Extractors," *University of Washington Department of Computer Science and Engineering Technical Report*, UW-CSE-02-12-03, December 2002.

What's in Constructing a Domain Model for Sharing Architectural Knowledge?

Rik Farenhorst
Vrije Universiteit, Amsterdam
rik@few.vu.nl

Remco C. de Boer
Vrije Universiteit, Amsterdam
remco@few.vu.nl

Robert Deckers
Philips Research, Eindhoven
robert.deckers@philips.com

Patricia Lago
Vrije Universiteit, Amsterdam
patricia@few.vu.nl

Hans van Vliet
Vrije Universiteit, Amsterdam
hans@few.vu.nl

Abstract

To promote knowledge dissemination in the software architecting process, it is crucial to deploy sound mechanisms to manage and share architectural knowledge. Sharing knowledge requires a common frame of reference. To this end, we constructed a domain model for sharing architectural knowledge. In this paper, we outline the iterative process used to construct this model. The main ingredients of this process are natural language text, packages, domain objects and actions, and object-interaction models. The recipe used to combine these ingredients consists of seven consecutive phases that iterate over three distinct stages of knowledge work: knowledge elicitation, knowledge structuring, and knowledge specification. Together, ingredients and recipe help in constructing a model that captures both static and dynamic aspects of a domain.

1. Introduction

A software architecture embodies the early design decisions of a system. These early decisions determine the system's development, deployment, and evolution.

Design decisions are like material floating in a pond. When not touched for a while, they sink and disappear from sight. These sunken decisions are the most difficult to change at a later stage. In particular, during evolution one may stumble upon them, try to undo them or work around them and get into trouble when the change turns out to be very costly, if not impossible. The future evolutionary capabilities of a system can be better assessed if these decisions were explicit. It is thus important to develop ways to effectively share knowledge pertaining to these design decisions. Design decisions, their rationale, their result, the people involved, and other relevant information are collectively referred to as architectural knowledge.

Sharing architectural knowledge is getting more and more attention recently. The primary focus of research on sharing architectural knowledge is on design decisions and their rationale [2, 13, 14]. Software architecture approaches that primarily focus on components and connectors fail to document the design decisions as well as the reasoning underlying the design decisions. Since the 'rationale' of a software architecture manifests itself through the design decisions, a growing number of researchers share the opinion that a software architecture can (or should) be viewed as the collection of these decisions [6], or as the design decisions plus the resulting design [8].

Sharing knowledge about a domain, i.e. an area of activity, requires a common frame of reference. This frame of reference provides the concepts that are used to express and communicate that knowledge. In particular, a frame of reference should address what can happen and what can exist in a domain.

To construct such a frame of reference that can be used to share architectural knowledge, we used a domain modeling method that was originally developed within Philips Research [3] as an extension of the KISS method [7]. In this paper, we present an iterative approach for building a domain model using this method. We illustrate the approach with the construction of our domain model for sharing architectural knowledge. The method targets both domain objects and domain actions. The resulting domain model captures both static and dynamic aspects of the architecting process, as well as the relations between them.

This paper is organized as follows. First, we describe the ingredients of a domain model. Next, the phases of the 'recipe' followed to construct a domain model for sharing architectural knowledge are elaborated upon. We conclude this paper with a reflection on our experience by listing some important lessons learned while modeling the architectural knowledge domain.

2. Ingredients for a Domain Model

The basic stages for constructing a domain model are knowledge elicitation, knowledge structuring and knowledge specification. The method from [3] uses the following ingredients: natural language for elicitation, packages, objects, and actions for structuring, and an object-interaction model plus definitions as specified knowledge. Each of these ingredients is presented in more detail below.

2.1. Elicitation: Natural language

Apart from very specialized fields, for instance mathematics, communication usually takes place using natural language. Natural language text can be acquired in numerous ways, e.g. by analyzing (transcriptions of) interviews with domain experts, through scenario authoring, or through brainstorm sessions. Written documents can also be taken into account.

Grammatical analysis of the acquired text provides a way of identifying candidate concepts. This analysis uses the sentences that express either some activity or a relation, such as specialization, composition, or aggregation. These sentences show how domain objects are changed, and which static relations exist between them. The direct object and indirect object(s) of a sentence are candidate objects in the domain. The verb(s) correspond to candidate actions when they indicate a change for the associated objects. Candidate actors are the subjects of the sentences.

Given for instance the sentence 'An architect discusses requirements with the customer, and they agree on a specification.', the candidate actors are 'architect' and 'they', candidate objects are 'requirements', 'customer', and 'specification', and the candidate actions are 'to discuss' and 'to agree'.

2.2. Structuring: Packages, objects, and actions

Knowledge structuring takes place at two levels: structuring the domain itself in packages, and structuring the knowledge contained in each individual package.

Packages provide a general grouping mechanism. By grouping domain concepts, potentially large and complex models can be developed and managed more effectively. The packages divide the modeling efforts into coherent subdomains with a limited scope.

One of the advantages of using packages is that it provides a means to 'divide & conquer'; it is often easier to focus on the contents of a single package instead of the whole domain. Packages also help in disambiguation of terminology, since they provide a 'namespace' for their contents.

Structuring a domain in packages has another benefit. It can also aid in assessing the model quality. By viewing each domain concept as 'owned' by a single package, even if it also used in other packages, a dependency between packages emerges. This dependency can be further analyzed. For instance, cyclic dependencies might indicate a need to reorganize package contents.

Within each of the packages, the elicited knowledge is structured in more detail. This is done by identifying domain objects and domain actions from the set of candidates. Domain objects represent the static concepts in a domain. Domain actions change the state of one or more domain objects, and reflect the dynamic concepts in the domain. Domain objects and actions are related by a participation relation, i.e. objects participate in an action.

To guide the knowledge structuring process, the method in [3] defines the following modeling rules with respect to the introduction and elimination of domain concepts.

- A domain object must have a unique identity, and may only be introduced when it undergoes actions in the domain that are specific to that object.
- A specialization relation may only be introduced when two or more objects share a participation relation. A good indication of a specialization relation is a juxtaposition of nouns, e.g. a 'design artifact' is an 'artifact'.
- Every object must have an associated instantiating action. Moreover, the objects should be unique, which means that synonyms from the elicitation stage should be merged. Homonyms within the same package, or namespace, should be renamed to provide unambiguous terminology.

2.3. Specification: Object-Interaction model

The knowledge that has been elicited and structured in the previous stages is specified in an object-interaction model, which captures the possible objects and actions in the domain. The object-interaction model is accompanied by a list of definitions, that specify the exact meaning of the objects and actions in the model.

Knowledge specification is done using UML with an additional profile for the domain actions. Domain objects are represented as UML classes. Domain actions are depicted as UML classes of the stereotype 'domain action', represented by the standard UML activity symbol. Participation relations are drawn as UML associations between domain objects and actions. The association name corresponds to the role of the object in the action. Specialization relationships are represented as standard UML generalizations. In addition to these standard UML constructs, instantiating participations are represented by an arrow at the end of the instantiated object.

The object-interaction model corresponding to the sentence from section 2.1 is depicted in Fig. 1.

When validation of the resulting object-interaction model indicates missing information, the model may be further refined. Validation and refinement are aided by the application of the rules that are listed in the previous section.

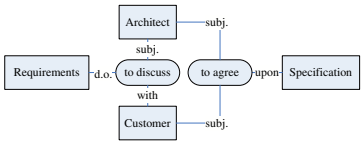


Figure 1. Example object-interaction model.

This refinement is another iteration of knowledge structuring, which can be preceded by further knowledge elicitation when necessary.

3. Recipe for a domain model

In this section, we outline the process we followed - the recipe - to combine the ingredients from the previous section. A schematic overview of the recipe can be found in Fig. 2. The abbreviations 'E', 'St', and 'Sp' used below denote elicitation, structuring, and specification of knowledge respectively.

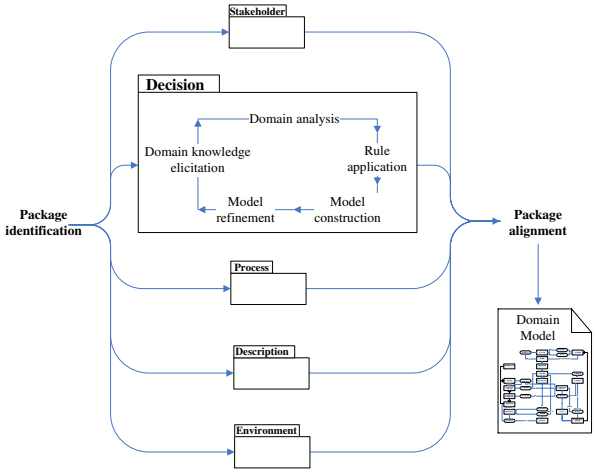


Figure 2. Outline of the recipe followed.

1) Package identification [St]: We used the package construct to split the architectural knowledge domain into various subdomains. Each package covers part of the domain.

A software architecture is used as a means to communicate with stakeholders. The stakeholders should be convinced that the solution satisfies their needs. This means that the stakeholders are central to the architecting process. In the architecting process, decisions are taken that shape or influence the architecture. These decisions are influenced by the technical, business, organizational, political, and economic environment in which the decisions are taken, and are reflected in architectural descriptions and other design artifacts. Together, the stakeholders, their decisions, the architecting process, the architectural descriptions, and the environment sketch the outline of the architectural knowledge universe of discourse. We initially identified five packages, corresponding to these areas; 'Stakeholder', 'Decision', 'Process', 'Description', and 'Environment'. For each of these packages, the following activities have been carried out.

2) Domain knowledge elicitation [E]: Textual domain knowledge was acquired from various sources, including interviews with software architects, literature, standardization bodies, and our own experience.

Besides experience with software architecture - partially captured in interview transcripts, partially our own - literature and standards proved to be an excellent source of domain knowledge. The IEEE has published a standard for describing the architecture of software intensive systems [5], which we used as a basis for the 'Description' and 'Stakeholder' packages. The OMG's Software Process Engineering Metamodel [10] standard was the foundation of the 'Process' package. Recent research on design decisions, such as [13], helped shaping the 'Decision' package. Lacking information on the contents of the environment of software architectures modeled, the contents of the 'Environment' package remained open.

Note that our aim was to construct a model that describes what can happen and what can exist in the domain of architectural knowledge. In this, we were not particularly interested in the possible actors of an action, but rather in the action itself. To this end, the distilled sentences were 'anonymized' by replacing the subjects with 'someone'. So, instead of 'the architect takes a decision', we write 'someone takes a decision'.

In this phase redundant sentences are not yet filtered out. For example, the sentences "Someone takes a decision", "Someone chooses an alternative from a set of solutions", and "Someone picks an option" probably bear the same meaning. Chances are that these sentences will be merged later on, which will likely result in one or more candidate concepts not surviving that phase.

3) Domain analysis [E]: Grammatical analysis of the relevant sentences led to a set of domain objects as well as the most important actions on these objects. Based on the example sentences above, candidate objects are decision, alternative, solution, and option. Candidate actions for our domain model are 'to take' (a decision), 'to choose' (an alternative), and 'to pick' (an option).

4) Rule application [St]: Although listed as a separate phase, rule application took place for a large part in combination with domain analysis. In this phase, we judged among others the relevance of the sentences identified in the knowledge elicitation phase. This influences the sentences that are subject to the grammatical analysis in the previous phase.

For instance, we quickly determined the sentences

”Someone chooses an alternative from a set of solutions” and ”Someone picks an option” to have exactly the same meaning; both the concepts ’to pick’ and ’to choose’ as well as ’alternative’ and ’option’ are interchangeable. Therefore, we eliminated one of the sentences from the set. The remaining candidate concepts were then further scrutinized. This led to the insight that ’alternative’ and ’solution’ refer to the same object. Since the alternative is chosen from a set of solutions, the alternative itself must be a solution. Furthermore, taking a decision can be seen as choosing from a number of alternatives. In other words, the chosen alternative *is* the decision. However, not every alternative is, or will be, a decision. This means that they both are unique domain objects.

5) Model construction [Sp]: By adhering to the diagramming instructions outlined in Section 2.3, the (initial) object-interaction model was drawn per package. See Fig. 3 for an example of the ’Decision’ package.

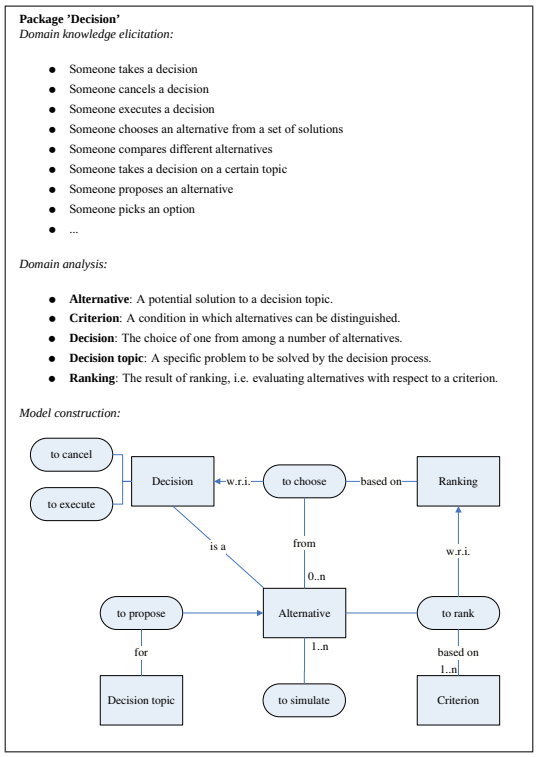


Figure 3. Constructing package ’Decision’

6) Model refinement [St]: For most packages, the object-interaction model from the previous phase was not yet mature enough to be considered the end result. In the model refinement phase, we took an initial model and analyzed it for inconsistencies, and missing or superfluous information. This phase can be seen as another iteration of the previous phases.

As part of this phase, we also presented the model to the members of our research team that were not directly involved in the modeling process. We provided them not only with the graphical representation of the model, but also with the exact sentences reflected in the diagrams. These sentences can be reconstructed by generating a sentence for each domain action and its associated objects depicted in the model. For instance, the action ’to choose’ in the model depicted in Fig. 3 corresponds to the sentence ”Someone chooses from a number (0..n) of alternatives based on a ranking, which results in a decision”.

Based on this analysis, we solved a number of problems with the initial version of the model. Recurring problems were objects without instantiating actions, the need for a specialization relation, and domain objects or actions we had initially overlooked. Objects without instantiating actions are a direct violation of the rule that every object must have an associated instantiating action. Instantiating actions were therefore introduced for these objects. The need for a specialization relation is often indicated by one object having the same association with two distinct objects. This need is satisfied by adding an object with a specialization relation to the latter two objects.

7) Package alignment [St/Sp]: During the refinement of the individual packages, more and more overlap between the packages started to emerge. Domain objects that were present in one package were also needed in other packages. In particular, (candidate) objects for the ’Stakeholder’ package - which had not yet been completely refined - started to appear in all of the ’Decision’, ’Description’, and ’Process’ packages. Furthermore, cyclic dependencies were starting to emerge. This was the trigger to try and align the individual packages into one coherent structure.

The alignment led to some further insights, and can perhaps be regarded as another refinement phase. For instance, the criterion that is used for ranking in the decision making process coincides with a stakeholder’s concerns. Similarly, a stakeholder’s concern is tightly linked to a decision topic that needs to be addressed.

Up until this moment, our efforts to model the ’Environment’ package had only resulted in a list of possible external forces that might impact the architecture. Since the ’Environment’ package had to be the representation of the ’outside world’, it had in essence an unlimited scope. Therefore, it was possible to come up with new domain objects time and again. It turned out to be very hard to devise a structure that sensibly represents the whole environment. We decided that the best solution would be to capture the environment in a single domain object, dubbed ’External event’.

The result of this package alignment phase, our domain model for sharing architectural knowledge, is described in more detail in [1].

4. Lessons learned

According to [11], the description of a design process is a rational reconstruction of that process. This also holds for the recipe outlined in Section 3. For instance, while working on the domain model, the domain analysis and rule application phases were interleaved. It is only in hindsight that we describe them as being distinct.

Our method helps in constructing a model that captures both static and dynamic aspects. By capturing dynamic aspects of a domain, the resulting model not only focuses on what exists in a domain (e.g. design decisions, architectural descriptions), but also on how things happen (e.g. the decision making process). This allows us to model the architecting process as a whole, including dynamic relations between otherwise static objects.

The package alignment had an unexpected outcome. We initially expected to have the five packages that were the result of the package identification phase, to also show up in the end result. However, the current integrated result is more appealing than a collection of loosely related packages. Having a single model makes the interactions and relationships more visible. For instance, our model exposes a loop from decision making ('Decision') to architectural descriptions ('Architectural Model'), via roles and responsibilities ('Artifact', 'Role', 'Stakeholder'), and back to decisions again ('Concern', 'Decision topic'). This corresponds to the recursive nature of decisions. A decision may give raise to new concerns of specific stakeholders. To resolve these concerns, new decisions have to be taken. This recursive problem-solving nature of the architecting process is central in existing architecture methods [4]. Such a loop is less obvious with multiple packages involved.

A prime factor enabling the alignment of packages and aggregation into a single model is the existence of a primary package ('Stakeholder') that acts as a focal point to which the other packages are linked.

We further learned that model refinement is an important part of the process. Phase 6) in Section 3 lists problems that indicated the result was not yet mature enough. Some of these problems pop up in an object-interaction model, while they would have been difficult to catch in an earlier stage. A good example is the lack of an instantiating action, which shows itself in an object-interaction model as a domain object without incoming arrows attached to it. This is a direct violation of the rule that every object must have an associated instantiating action. The object-interaction model makes spotting this violation easy, whereas pinpointing it through analysis of the text upon which the model has been based would be much more cumbersome.

Because of our background in software architecture, we sometimes acted both as knowledge engineer and as domain expert. This proved to be somewhat difficult to combine. In particular, domain knowledge elicitation tends to spread throughout the whole process. This makes it harder to adhere to the stepwise process described in Section 3. Therefore, it is crucial to remain conscious about which hat one carries. Either you are working as a knowledge engineer on structuring or specifying the model itself, or you are acting as a domain expert, assessing the domain knowledge reflected in the model.

Since one of the intended uses of our domain model is that of a common frame of reference within our research consortium, all of its members contributed to our domain model as domain experts. Furthermore, all researchers have been directly involved in the modeling process as knowledge engineers as well. This leads not only to strong commitment of all parties to the resulting domain model, but also to a feeling of 'shared ownership' of the result. Although we are confident that our domain model is broadly applicable, these side-effects of involving all 'stakeholders' in the construction of the model ensure we have an excellent baseline for our current case studies. In these case studies, we further explore how architectural knowledge is and could be shared within the organizations of our industrial research partners.

5. Related work

A modeling notation that is often used in software engineering is UML. This notation, which consists of graphical symbols and text, is among others used as a means to specify and communicate domain models.

UML contains constructs to describe static aspects of a domain, e.g. through class diagrams. It can also describe dynamic aspects, e.g. through activity diagrams. UML, however, does not allow the combination of these two aspects in a single diagram at the class level.

The object-interaction model we use does combine static and dynamic aspects of a domain at the class level . The method from [3] prescribes the use of an object-interaction model for exactly that purpose. The object-interaction model is already introduced in [7], of which [3] is an extension. This extension introduces packages as a construct to manage the modeling process by applying a 'divide & conquer' technique to this process.

An object-interaction model contains constructs familiar from UML. In fact, the object interaction model is drawn by using a UML profile for domain actions.

A well-known knowledge engineering methodology is CommonKADS [12]. CommonKADS distinguishes three stages in the process of model construction; knowledge identification, knowledge specification, and knowledge refinement. These stages are present in the recipe from Section 2 as well. Knowledge identification would roughly correspond to phases 2) to 4) from our recipe. Phase 5) corresponds to the CommonKADS knowledge specification stage, while knowledge refinement is done in phase 6). Note

that CommonKADS does not use the notion of packages, so phases 1) and 7) are not directly related to any of the three stages.

The need for a domain model for sharing architectural knowledge stems from the increasing attention to managing and sharing architectural knowledge. In recent literature, architectural knowledge is acknowledged to be largely represented by the architectural design decisions [2, 13, 14] that are incorporated in the software architecture. Since these decisions shape the design, software architecture can be viewed as the collection of these decisions [6], or as the design decisions plus the resulting design [8].

In our approach to effectively manage and share architectural knowledge, we focus on a combination of static and dynamic aspects of the architecting process. Our domain model focuses on both the design decisions including the underlying rationale, and the result of this process, reflected in design artifacts such as architectural descriptions. This focus enables us to capture the recursive nature of this process, that is described in [4]. Other approaches tend to focus on either the decision taking process, without explicitly looking at the process in which the decisions are being taken (e.g. [9] and its descendants), or on the design artifacts (e.g. [5]).

6. Conclusion

Crucial information on why a software architecture is the way it is tends to disperse when the knowledge pertaining to the architecture is allowed to disappear from sight. To preclude this problem, sharing architectural knowledge is of paramount importance. We developed a domain model that can act as a frame of reference for sharing architectural knowledge. In this paper, we describe in detail the process we followed to construct this model.

Our domain model results from a combination of three ingredients. For knowledge elicitation natural language input is used in various ways. Knowledge structuring benefits from the package construct to divide the domain in 'namespaces', and uses domain objects and actions to structure the contents of each individual package. To specify the static and dynamic aspects of the domain, an object-interaction-model is used.

We present a process consisting of seven consecutive phases that use these ingredients. These phases constitute the recipe we followed to construct our domain model. These seven phases have strong links with the three distinct stages of knowledge work, i.e. the input, throughput and output stage.

Based on our experience with this process, several insights are listed in Section 4. These lessons are useful for other knowledge engineers, especially those working in an area in which not only static aspects, but also dynamic aspects of the domain under consideration are important.

The resulting domain model provides us with a baseline for our current case studies, in which we further explore how architectural knowledge is and could be shared within the organizations of our industrial research partners.

Acknowledgement

This research has been partially sponsored by the Dutch Joint Academic and Commercial Quality Research & Development (Jacquard) program on Software Engineering Research via contract 638.001.406 GRIFFIN: a GRId For inFormatIoN about architectural knowledge.

References

[1] R. C. d. Boer, R. Farenhorst, V. Clerc, J. S. v. d. Ven, R. Deckers, P. Lago, and H. v. Vliet. Structuring software architecture project memories. In *8th International Workshop on Learning Software Organizations*, Rio de Janeiro, 2006.

[2] J. Bosch. Software architecture: The next step. In *Software Architecture: First European Workshop (EWSA 2004)*.

[3] R. T. C. Deckers. Functional specification method for CE product families. Technical report, Philips Research.

[4] C. Hofmeister, P. Kruchten, R. L. Nord, H. Obbink, A. Ran, and P. America. Generalizing a model of software architecture design from five industrial approaches. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA 2005)*, Pittsburgh, Pennsylvania, 2005.

[5] IEEE. IEEE recommended practice for architectural description of software-intensive systems. Standard 1471-2000, IEEE, 2000.

[6] A. Jansen and J. Bosch. Software architecture as a set of architectural design decisions. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA 2005)*, Pittsburgh, Pennsylvania, 2005.

[7] G. Kristen. *Object Orientation: The KISS Method*. Academic Service, 1995.

[8] P. Kruchten, P. Lago, H. v. Vliet, and T. Wolf. Building up and exploiting architectural knowledge. In *5th Working IEEE/IFIP Conference on Software Architecture (WICSA 2005)*, Pittsburgh, Pennsylvania, 2005.

[9] W. Kunz and H. Rittel. Issues as elements of information systems. Technical report, Institute of Urban and Regional Development, University of California.

[10] OMG. Software process engineering metamodel specification. Technical Report formal/05-01-06, Object Management Group, January 2005.

[11] D. L. Parnas and P. C. Clements. A rational design process: How and why to fake it. *IEEE Transactions on Software Engineering*, 12(2):251–257, 1986.

[12] G. Schreiber, H. Akkermans, A. Anjewierden, R. d. Hoog, N. Shadbolt, W. Van de Velde, and B. Wielinga. *Knowledge engineering and management, the CommonKADS methodology*. MIT press, 2000.

[13] J. Tyree and A. Akerman. Architecture decisions: Demystifying architecture. *IEEE Software*, 22(2):19–27, 2005.

[14] J. S. v. d. Ven, A. G. J. Jansen, J. A. G. Nijhuis, and J. Bosch. Design decisions: The bridge between rationale and architecture. In A. H. Dutoit, R. McCall, I. Mistrik, and B. Paech, editors, *Rationale Management in Software Engineering*. Springer-Verlag, 2006.

A Pattern Taxonomy for Business Process Integration Oriented Application Integration

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

Application Integration is a work and cost intensive part of both, software development projects and ongoing software maintenance. Due to its inherent feature of decoupling applications in a robust manner, message-oriented integration techniques are in use for a long time. Recently, higher-level styles that provide an abstraction closer to application issues are under discussion. Because of its high abstraction level providing a global view, business-process-integration based application integration is a promising candidate to start with. This paper presents a refined approach of Business Process Integration Oriented Application Integration (BPIOAI). It facilitates the task of application integration by means of defining standard integration processes based on a taxonomy of integration patterns and a set of integration services. The identified taxonomy takes the structure induced by the integration processes into account. It provides a standardization that is especially suitable for centralizing the application integration development in central development centers of large organizations. Besides these benefits, this work provides an approach toward a domain-specific language for the domain of application integration.

Keywords: *application integration, business processes, integration pattern, standards*

1. Introduction

As application orientation has been a crucial part of everyday work in almost all application areas of IT for some time, there is a bunch of proposed techniques and approaches to cope with its complexity in a manner that accelerates new developments and lowers maintenance costs. Whereas software development in general has seen different maturity levels and paradigms, the task of system integration has faced a lot of paradigms itself. In an overview given by Linthicum [8], different *integration styles* that describe how integration is managed are distinguished, i.e., information-oriented, business process integration-oriented, service-oriented and portal-oriented application integration.

For real-life application integration, the paradigm of *message-based* integration, has been considered most useful due to it's characteristics of decoupling systems in a manner that make compound systems more tolerant against failures and, hence, more reliable. In this context, message oriented integration patterns have been analyzed in much detail as well. Even if message-based application integration is applied as an *information-oriented application integration* (IOAI) approach, the messaging basically provides the technical infrastructure, i.e., the means for reliable and asynchronous communication.

However, integration styles of a more higher level of abstraction are on their way. The most promising recent paradigm of application integration, *Business Process Integration Oriented Application Integration* (BPIOAI), provides a far more abstract as well as global view to integration. This is achieved by introducing a business process controlling the order of how participating application systems are invoked. The process layer of BPIOAI does not solely make direct use of application systems. Furthermore, conventional IOAI based systems can be used as long as they expose a well-defined interface to the business process. Since BPIOAI works on-top of IOAI, the basic exchange of information is handled with messages, too.

A crucial part of supporting integration in everyday work, is the identification of situations, problems and lessons learned from experience and *best-practice* knowledge that occur and apply frequently despite the different specifics of the problem at hand. *Pattern* as introduced by Alexander [1] and applied to the domain of software engineering by the Gang of Four [2], are a well-known means to capture and re-use such kind of knowledge. Because re-use requires knowledge about existing patterns, guiding effective work with patterns and their proper combination by classifying, grouping and naming patterns with the help of a pattern language is even more desirable. However, a pattern language may be strictly tuned to a restricted application area. Hence, for more complete application domains, like the application integration area, a domain specific language (DSL) should be used [3].

In order to apply patterns to the domain of message oriented application integration, Hohpe and Woolf propose 65 standard *Enterprise Integration Patterns* [4] grouped accord-

*Helge Hofmeister is an external PhD student with the Distributed and Mobile Systems Group at Bamberg University.

ing to 6 different root pattern and describe how these patterns can be applied to build message-based integration systems in a standardized way. Additionally, guidelines in terms of descriptions that allow to compose a solution for a given integration problem by sequencing several patterns around a message-based communication channel are provided. Since most of Hohpe's message-oriented patterns provide additional functionality on-top of a messaging-system and more abstract integration styles are usually based on a messaging layer, most of these patterns should be as well useful in the context of other integration styles.

This paper presents a refined approach of Business Process Integration Oriented Application Integration (BPIOAI) that defines a set of standard integration processes and services and uses this setting to categorize the messaging patterns more strict as it is done by Hohpe and Woolf [4]. Our work has been carried out in the context of the central governance of a customer for systems that are implemented using the BPIOAI middleware from SAP - the Exchange Infrastructure. By adding two context specific modelling languages on-top of Hohpe's basic patterns, categories are built that empower the basic patterns to describe a complete Application Integration Language (AIL).

Besides discussing related work in section 2, the rest of this paper outlines our approach by defining standardized integration processes (section 3), discussing so-called *idioms* for the most important integration pattern (section 4) and illustrating the approach by means of a simple case study (section 5). Section 6 discusses issues of future work.

2. Related Work

The Enterprise Integration Patterns of Hohpe and Woolf [4] define the starting point and setting of our work. Van der Aalst discusses different patterns in the area of workflows [13] by distinguishing data ([9]) and resource ([10]) patterns. These patterns are designed independently of application domains and are not specific to application integration. Nevertheless, in the case of BPIOAI, generic workflow patterns support the analysis and even more the implementation of integration specific patterns because they provide help in implementing workflow systems that are in turn a building block of BPIOAI based integration systems ([8]).

Jayaweera et al. focus at the functional level of process models in order to facilitate the generation of e-commerce systems out of business models [6]. In doing so, they describe how existent business models are transformed into process models. This work is based on the theory of *Language Actions* and aims at the generation of processes. The abstraction level of this business-centric work is being located one conceptual level above our work. This is because we aim to guide the implementation of the integration systems automating the processes as they are generated by the approach of Jayaweera or other business process methodologies.

3. Standardized Integration Processes

The aim of our work is to standardize system integration in such a way, that both the implementation of composite applications is eased and the conventional application integration is facilitated. This is why we add a behavioural layer on top of the pattern language of Hohpe that basically deals with data aspects [1]. This layer can be understood as a replacement of the pipe-architecture that is used by Hohpe [4] to categorize patterns for message based application integration.

Because our scope of integration ranges up to cross-system service orchestration, the behavioral layer defines generic and configurable integration processes instead of a simple pipe. These integration processes allow to describe the logic of cross-system composite applications on a business logic centric process layer on top that handles the detailed tasks due to distributed and heterogeneous systems. These single steps of the standard integration processes were surveyed among integration specialists within a large German IT services company and are understood as loosely coupled functionality that is coarse grained in terms of integration functionality. This kind of tasks is handled by so-called pre-defined *integration services* (IS). Based on the taxonomy that is provided by the disjoint functionalities of the IS (cf. section 4), Hohpe's integration patterns are used to support the implementation of these services. Hence, BPIOAI focused integration tools such as IBM Websphere [5], SeeBeyond's ICAN Suite [12] or SAP's Exchange Infrastructure [11] can easily re-use the functionality that is provided by the basic patterns and the integration services as well.

We introduce *two integration processes that orchestrate the IS*: the so-called *Integration In-Flow* (IIF) reads data from and the so-called *Integration Out-Flow* (IOF) stores data to connected legacy systems. The IIF and IOF steps are described in the activity diagrams of Fig.1 and Fig.2, respectively. Since the integration processes need to be generic, they should be capable to handle different communication semantics. For the same reason, the Integration Process needs to expose a generic interface to the business application on top of it. Thus, the IIF starts with the reception of either a synchronous or the reception of an asynchronous call or intrinsically. Subsequently, an *Event* is created by an IS. This *Event* is used as a ticket that is passed to subsequent IS as well as to a possible composite application on top of the integration processes. The data that should be transferred by the IIF is read by an IS called *Receiver Service*. This service has information about how to connect to the legacy systems as well as how data should be filtered, collected or reassembled. Optional steps of the processes are depicted as having a decision block as their predecessors. Thus, the subsequent step of acknowledging the request is optional. For synchronous calls, this acknowledgment would be the answer to the request. For asynchronous calls, this would be a subsequent message that is sent back to the originator of the request. If configured accordingly, the received data could be validated. Whether this solely involves

[1] At least the part we are using basically deals with data transformation

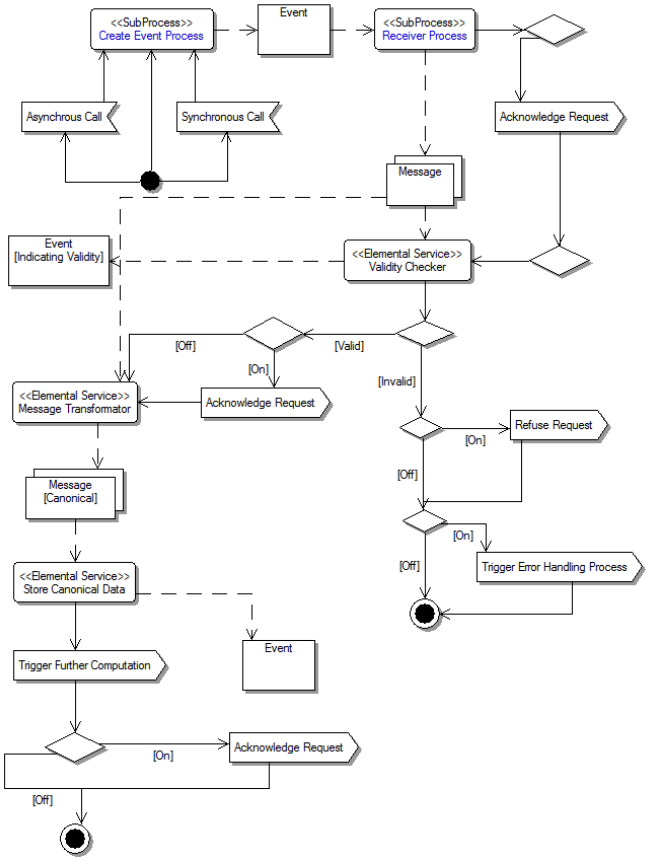


Figure 1. Generic Integration-In Flow

syntactic schema checking or additional semantic checks as well depends on the implementation of the *Validity Checker*. Based on the validity the process may either continue, the request could be refused or a more sophisticated error handling process may be triggered. If the data is valid it is transformed into a canonical data model ([7]). This intermediary data format is used to decrease the need for different implementations of the *Transformator Service*. Finally, the canonical data is stored in a data store and the ticket identifying this data is either directly passed to the IOF or to the composite application. Subsequently, components may solely deal with this ticket. For that sake the Event Service may attach certain information to the *Event*. This is a similar mechanism to the *slip* [4]. But in contrast to the message slip, not all information has to be included because the data store is accessible with the *Event* as the key. Thus, components could look up data as needed.

The IOF process for updating application systems (cf. Fig. 2) is generic as well. Note that a synchronous call to the coupling system may still be active. Thus, the IOF either continues the execution of a synchronous or an asynchronous call. If a synchronous call is continued, the termination of the request is handled by the final acknowledgment step of the IIF. Subsequently, a service is used to fetch the actual data based on the *Event* from the data store that keeps the canonical data. Depending on the configuration, it may be required to pre-transform the canonical data. If so, a *Transformator Service* may be invoked. Subsequently, a dedicated IS attaches routing information to the *Event*. Based on this information the data could be transformed into the partner specific format by another Transformator Service. Afterwards the data is written to the target system by the Updater Service. In case of a failure that may occur while updating the target system, the request may be refused and error handling may be triggered.

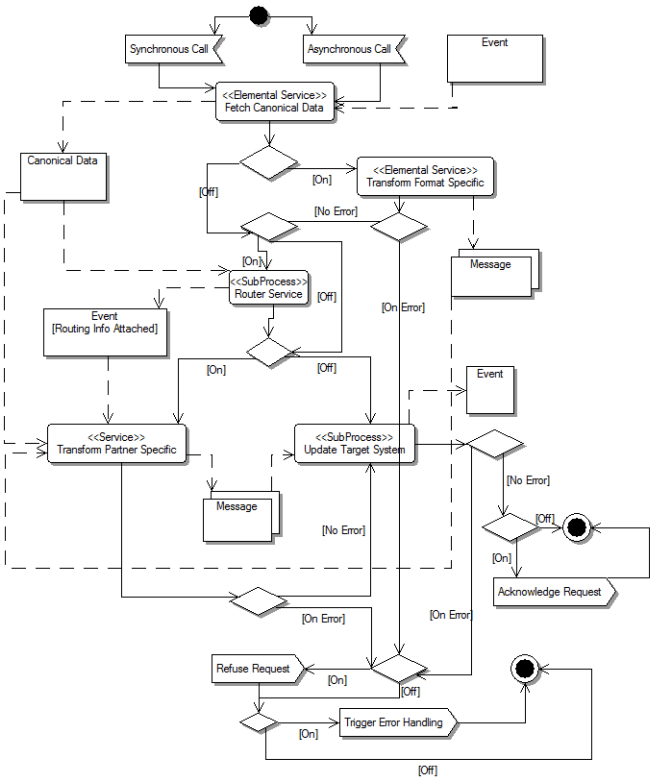


Figure 2. Generic Integration-Out Flow

4. Idioms for the Integration Services

In order to standardize the implementation of the coarse-grained integration services, we use Hohpe's [4] integration patterns to distinguish different IS by integration *idioms*. In the following, we introduce and discuss the Integration Services for the In-Flow and Out-Flow as well as the idioms that may be used to implement them if required.

The **Event Creator Service** incorporates the concept of *triggering* the coupling-system. This may happen intrinsically or extrinsically. In case of intrinsic event generation, the coupling-system itself generates an *Event* independently of the connected application system's state. Extrinsic event generation describes that the connected application system triggers the coupling-system by an internal state-transition. In either case the output of this service is always an unique event. The following integration patterns are useful to the Event Creator Service[2]:

- In the case of extrinsic triggering, the *Selective Consumer* may act as a filter and block events that are not

[2] All the pattern and pattern names used here are due to Hohpe [4]

relevant for the coupling-system. This minimizes the complexity within the application system since it does not need to filter itself.

- The *Polling Consumer* that is able to detect state transitions by itself, is used whenever the sending application system should not be aware of being coupled with another application system, but should act as the initiator of interactions.

The **Receiver Service** de-couples the functionality of reading data from the functionality of triggering the system in order to decrease the complexity of the single IS and to increase their re-usability. This is why the Receiver Service is disjoint from the Event Creator Service. The implementation of the Receiver Service heavily concerns the connectivity to the connected application systems. Anyway, in our description we rely on the integration middleware and solely focus on the functional creation of messages. The *Message Sequence* pattern may be used here as well as the *Splitter*, *Aggregator* and *Resequencer* pattern. *Message Sequence* is distinguished from the *Splitter* pattern, since the purpose of splitting a message is different. However, the real implementation may be identical.

The additional integration services to be used in our inflow process, i.e., the **Validity Service** and the **Message Transformator** cannot be supported by the integration patterns according to [4]. The same holds for the **Data Fetcher** and the **Message Transformer** services that are used in the outflow processes. Implementation support can be found here for the

- **Router Service** through the *Content Based Router*, *Dynamic Router* and *Recipient List* pattern as well as for the
- **Updater Service** that is supported by a wide range of patterns including *Message Sequence*, *Aggregator*, *Resequencer*, *Event Message*, *Document Message*, and *Command Message*.

The additional integration services to be used in our outflow process, i.e., the **Data Fetcher** and the **Message Transformer** cannot be supported by the integration patterns according to [4].

Implementation Dependencies

Patterns and idioms describe best practices and solution descriptions in a standardized form. Nevertheless, the description is not formal and the actual implementation of the idioms is still to be performed case by case. But the implementation itself is under some circumstances reusable. In order to provide support for a central design center, we analyzed the idioms in terms of their dependence to certain factors. With this information, the implementation effort estimation can be improved since the identification of the required integration services, idioms and dependency factors are already known at design time. This information can be used to determine which elements could be re-used and which need to be re-implemented. The dependency information is shown in Table 1. The single rows show the idioms that may be required for a specific service. The columns show the different dependency factors:

- **Communication semantics**: Most IS are called by a service orchestration layer and usually the result of a service needs to be returned synchronously. In some cases, asynchronous processing may be appropriate, if the communication between the connected application systems is asynchronous as well. Thus, in some cases the implementation of the idioms forming the integration services may be dependent to the communication semantics.
- **Effective data format**: Although one of the motivations for the integration processes is the standardized data-format independent way of connecting to application systems, some integration services are nevertheless dependent to the data format that is used by a connected system. Thus, some of the idioms need to be implemented for every single data format of a connected system. Note that this dependency factor solely describes the dependency from the format. It does not need any statement about whether the sender or the receiver uses this format.
- **Canonical data format**: Usually, a canonical data format is desirable because it reduces the amount of required data mappings. If a certain idiom does not have the purpose of data transformation, it is dependent on a canonical data format as well. As long as the anonical data format is not changed, this factor is uncritical.
- **Sender** and **Receiver**: Integration processes are designed to be executed in some sort of integration server middleware. This kind of middleware usually that provides adapters to handle the effective communication with application systems. In other scenarios the integration server as well as the application systems may be able to deal with Web Services or other ways of communication like, e.g., CORBA. Anyway, certain aspects of the implementation may be dependent on the connected system. The *Sender* (*Receiver*) factor indicate that an idiom's implementation may be dependent on the sending (receiving) application system, respectively.

5. Business Case

To demonstrate the benefit of our categorization, we give a real-life scenario that is used to integrate multiple ERP systems of an industry company *I* with a portal application as well as external suppliers of that company. In doing so we focus on the value that comes with the approach in terms of standardization and re-usability. We present our example as the governance of a company may proceed in alignment with the integration processes we defined.

Multiple divisions of company *I*, possibly working with different ERP systems, order certain goods and services

Pattern/ Service	Communication Semantics	Actual Data Format	Canonical Data Format	Sender	Receiver
Selective Consumer	$\approx$	$\bowtie$	$\approx$	$\approx$	$\approx$
Polling Consumer	only applicable for async. processing	$\bowtie$	$\approx$	$\bowtie$	$\approx$
Splitter	$\bowtie$	$\bowtie$	$\approx$	$\approx$	$\approx$
Message Sequence	$\bowtie$	$\bowtie$	$\approx$	$\approx$	$\bowtie$
Aggregator	$\bowtie$	$\bowtie$	$\approx$	$\approx$	$\approx$
Resequencer	only applicable for async. processing	$\bowtie$	$\approx$	$\bowtie$	$\bowtie$
Validity Check	$\approx$	$\bowtie$	$\approx$	$\approx$	$\approx$
Message Transformator	$\approx$	$\bowtie$	$\bowtie$	$\approx$	$\approx$
Store Canonical Data Format	$\approx$	$\approx$	$\bowtie$	$\approx$	$\approx$
Fetch Canonical Data Format	$\approx$	$\approx$	$\bowtie$	$\approx$	$\approx$
Content Based Router	$\approx$	$\approx$	$\bowtie$	$\approx$	$\bowtie$
Dynamic Router	$\approx$	$\approx$	$\bowtie$	$\approx$	$\bowtie$
Recipient List	$\approx$	$\approx$	$\bowtie$	$\approx$	$\approx$
Event Message	$\bowtie$	$\bowtie$	$\approx$	$\approx$	$\bowtie$
Document Message	$\bowtie$	$\bowtie$	$\approx$	$\approx$	$\bowtie$
Command Message	$\bowtie$	$\bowtie$	$\approx$	$\approx$	$\bowtie$
Independent = $\approx$	Dependent = $\bowtie$				

Table 1. Implementation Dependencies

within their ERP systems. These orders are sent to a composite application within a common portal. Additionally, suppliers are being notified about new orders. Afterwards, suppliers may log-on to the portal, check changes and accept/decline orders. Relevant actions of the supplier, such as accepting an order, are then posted to the according ERP system. This logic is described as an event-driven business process that controls the integration (cf. Fig.3) and is carried out by an integration middleware that is capable of executing processes. Note that the business logic is implemented within the portal application and the processes within the integration middleware are solely integration workflows.

Keeping the standard processes, services and patterns in mind, the central governance of *I* can easily design the coupling-system by analyzing the requirements of the business process. Afterwards, the design can easily be implemented by an IT supplier.

This design is a direct result of requirements of the business process: The ERP system of the purchasing unit and the portal application need to exchange purchase orders and order changes. In addition, the supplier has to be informed about new orders. As a result of a questionnaire presented to the process owner, the governance identifies that this is to be done via email. Additionally, only the ERP system is capable of writing orders to a relational database and for receiving order changes by retrieving them from the database. In turn, the portal computes purchase orders that are specified in the XML-schema of *I's* industry [3]. The same applies for generated order changes.

Following the IIF, it is identified first that the ERP system does not trigger the coupling-system actively and that not all orders need to be transferred to the portal since internal orders are handled differently. Thus, the *Event Creator Service* applies the *Polling Consumer* as well as the *Selective Consumer* idiom. As these idioms are solely dependent on the data format of the sender and on the type of sending system, the implementation can be re-used for similar scenarios involving the same type of ERP system. Required functionality for the *Receiver Service* is checked next. Here, only a database adapter with basic read operations is used and, hence, no idioms are applied. Due to the data-constraints applicable for the relational database used, invalid messages are not considered a problem by the governance and the *validity check* is not used. In order to avoid double computation of orders, the *acknowledge request* option is used. [4]

For transforming the in-memory representation of database entries into the canonical data format, the governance dictates the use of the *Transformator Service* for purchase orders. Since the data is always stored into the data store in the canonical format, the according service is generic and it's reuse is recommended. Afterwards, the business process requires an email to be sent to the supplier. Therefore, the IOF is called asynchronously with the *Event* as the single parameter. The questionnaire to the process owner identifies that multiple mailboxes need to be informed via email, i.e. the customer as well as the purchasing unit that wants to monitor all emails that are sent to external suppliers. The detailed design of the IOF is as follows: The *Router Service* applies the *Recipient List* pattern. From an outside point of view, the Router Service splits the message in two and sends them to the *Transformator Service* in order to transform them into email-messages according to a given XML schema. For the update step, the

[3] This format is used as the canonical data format within the company.

[4] This acknowledgement is realized by writing an additional entry in the table of the ERP system using the IOF that connects to the ERP system.

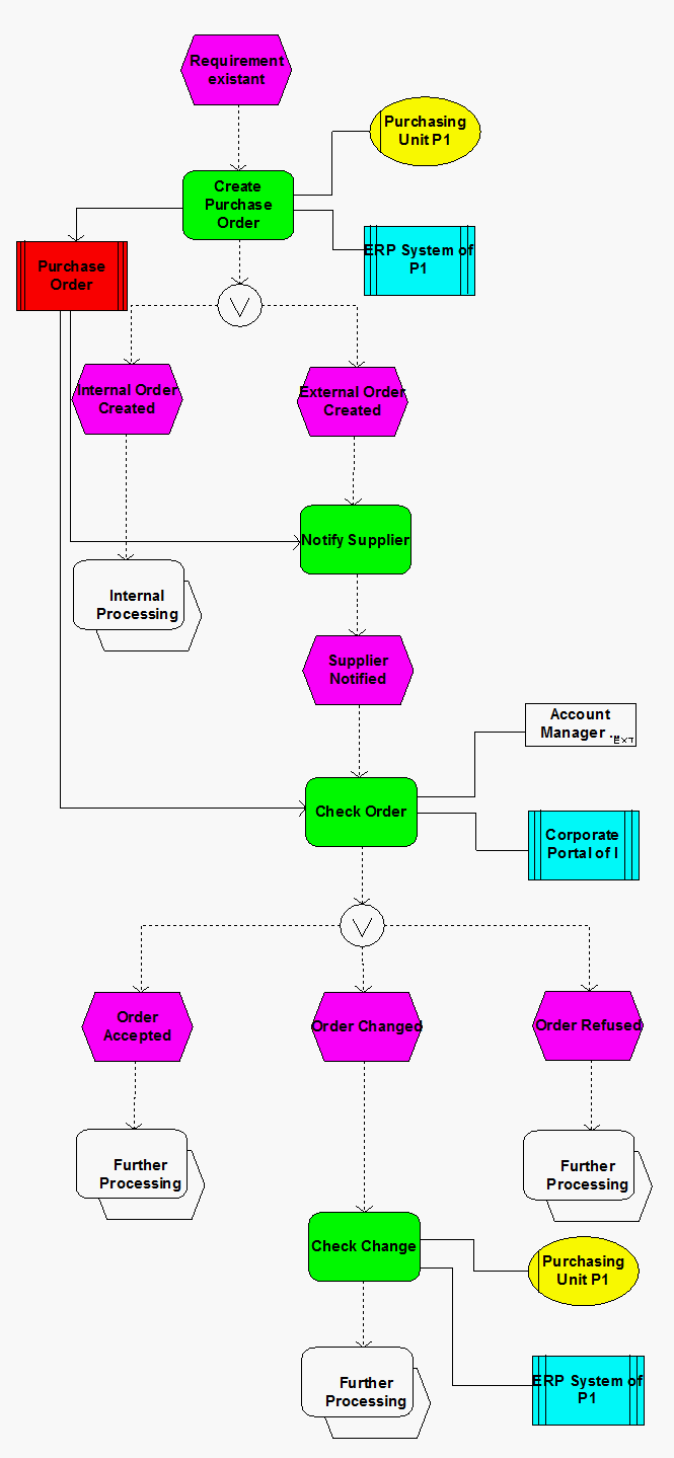


Figure 3. Business Process of the Example

design only recommends the usage of an SMTP-mail adapter that deals with the specific XML-message.

The next step is checking the order by the supplier by logging into the portal and reviewing the order. Since this is not an integration task, it is not described by the patterns and processes here.

However, it is an important point to stress that the portal application uses the data that are stored into the coupling-system's data store and, hence, has to be able to compute the data format of the coupling system [5].

As the final part, the design should include the propagation of the order change back to the ERP system. Therefore, the design is aligned with the integration process again. We omit the details since the IOF for the update of the ERP system does not introduce the need for additional idioms.

6. Conclusions

Our work standardizes the design of coupling systems by using a pattern taxonomy within the central governance of a customer for systems that are implemented using the BPIOAI middleware from SAP - the Exchange Infrastructure. This way of designing coupling systems also brings value to the implementation of composite applications.

Currently, we are collecting requirement classes from different real-life case studies. In doing so, we aim to define a formal language that transforms the requirements of a business process in conjunction with a dynamically created questionnaire into the implementation of a BPIOAI coupling system using the taxonomy presented here.

References

[1] C. Alexander. *A Pattern Language*. Oxford University Press, 1977.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison Wesley, 1996.

[3] J. Greenfield and K. Short. *Software Factories*. Wiley Publishing, Inc., Indianapolis, Indiana USA, 2004.

[4] G. Hohpe and B. Woolf. *Enterprise Integration Patterns*. The Addison Wesley Signature Series. Pearson Education Inc., 2004.

[5] IBM. Websphere Integration and application infrastructure software, 2006.

[6] P. Jayaweera, P. Johannesson, and P. Wohed. Process patterns to generate e-commerce systems. In H. Arisawa, Y. Kambayashi, V. Kumar, H. C. Mayr, and I. Hunt, editors, *ER (Workshops)*, volume 2465 of *Lecture Notes in Computer Science*, pages 417–431. Springer, 2001.

[7] G. Kaufman. Pragmatic ecad data integration. Technical Report 1, New York, NY, USA, 1990.

[8] D. S. Linthicum. *Next Generation Application Integration*. Addison-Wesley, Boston, MA USA, 2004.

[9] 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.

[10] N. Russell, D. E. ter Hofstede, Arthur H.M., and W. M. van der Aalst. Workflow resource patterns. *BETA Working Paper Series*, WP 127, 2004.

[11] SAP. SAP Exchange Infrastructure XI, 2006.

[12] Sun Microsystems. SeeBeyond ICAN Suite, 2006.

[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.

[5] This is the line we draw between a legacy application and a composite application. In the case of a legacy system, the IOF would be used to write the data to the application system.

A model-based design-for-verification approach to checking for deadlock in multi-threaded applications

Beata Sarna-Starosta, R. E. K. Stirewalt, and Laura K. Dillon
Software Engineering and Network Systems Laboratory
Dept. of Computer Science and Engineering
Michigan State University
East Lansing, Michigan 48824

Abstract

This paper explores an approach to design for verification in systems that are built atop a middleware framework designed to separate synchronization concerns from the "core-functional logic" of a program. The framework is based on a language-independent compositional model of synchronization contracts, called Szumo, which integrates well with popular OO design artifacts and provides strong guarantees of non-interference. An approach for extracting models from Szumo design artifacts and analyzing the generated models to detect deadlocks is described. A key decision was to use Constraint Handling Rules to express the semantics of synchronization contracts, which allowed a transparent model of the implementation logic.

1. Introduction

A key problem in the verification of concurrent software is the need to extract from design artifacts a *finite state model* that is amenable to analysis with respect to a property of interest. Approaches under the category *design for verification* (D4V) concede some degree of freedom during design (e.g., prescribing design rules [15], or requiring designers to instantiate a predefined set of patterns [12, 5]) to automate the derivation of such models. This paper explores an approach to D4V in systems that are built atop a middleware framework designed to separate synchronization concerns from the "core-functional logic" of a program.

The SynchroniZation Units MOdel (Szumo) framework associates each thread with a *synchronization contract* that governs how it must synchronize with other threads. At run time, schedules are derived by *negotiating* contracts among threads, so that a thread is scheduled only if its contract has been successfully negotiated. Applications using Szumo execute atop a middleware layer that implements the details of negotiation and scheduling. The contracts themselves are formed by conjoining module-level *synchronization constraints*, which a programmer declares in the module's interface. A novel characteristic of Szumo is that programmers declare these constraints in lieu of writing explicit synchronization code in the module's implementation, i.e., deferring the mechanics of synchronization to the middleware.

The middleware negotiates contracts in a manner that guarantees freedom from a large class of data races [13] and deadlocks. Consequently, these properties need not be verified. However, a model of the application is required to demonstrate freedom from all forms of deadlock and to verify more general safety properties. This paper describes an approach that leverages the separation of concerns provided by the Szumo framework to simplify extraction of models from a design that will be implemented using Szumo. The generated models are analyzed to detect deadlock.

The remainder of this paper describes our D4V approach. We build finite-state models from UML diagrams extended to represent synchronization constraints (Section 2). A critical component is the verifier customized to solve systems of synchronization constraints (Section 3). We found it natural to specify the synchronization semantics in the language of Constraint Handling Rules (CHR). We conclude with a summary of our approach and its comparison with related work (Section 4).

2. Background

We designed Szumo to support development of multi-threaded object-oriented (OO) systems, for which a key problem is to synchronize threads that operate over shared data. Thread synchronization logic is inherently complex, as it involves reasoning over state spaces that grow non-linearly with the size of the program. Without proper synchronization, concurrent access to shared objects can lead to race conditions, and incorrect synchronization logic can lead to starvation or deadlock. Szumo is a

language-independent model of synchronization contracts and middleware-based contract negotiation [3].[1] To date, we have integrated it into an extension of the Eiffel language and, more recently, as an object-oriented framework in C++. Szumo is the basis for the D4V approach described in the sequel, which implements the verification engine using CHR. This section supplies background on relevant D4V artifacts, Szumo and CHR.

2.1. D4V Artifacts

A fundamental D4V decision involves the granularity of sharing among threads: To improve the efficiency of analysis, a designer may opt for coarse-grain sharing; however, doing so limits concurrency. In Szumo, designers choose the granularity of sharing by deploying program objects into *synchronization units*, which are "object-like" containers of one or more program objects. When a program object is created, the middleware deploys it to exactly one synchronization unit, where it remains throughout its lifetime. Moreover, a thread that holds exclusive access to a synchronization unit holds exclusive access to all program objects contained within that unit. A synchronization unit is object-like in that it may encapsulate state, provide operations, and reference other synchronization units whose operations it uses to implement its own, i.e., in a client–supplier fashion. Because sharing occurs only at the level of units, concurrency analyses need not consider how units are composed of program objects. Thus, in the sequel, the "objects" we refer to will always be synchronization units and not program objects.

Synchronization units are identified with instances of types, called *unit classes*. In addition to standard operations, a unit class may declare (zero or more) *unit variables*, *condition variables*, and synchronization constraints. Unit variables are references in a client class C to units that serve as suppliers to units of type C. Condition variables are boolean-valued variables in a unit class. The *synchronization-relevant state* of a unit is represented by an assignment of values to the unit's condition variables and unit variables. Finally, the synchronization constraints declare the suppliers to which a client unit requires exclusive access as functions of the client's synchronization-relevant state. A client unit c is said to *entail* a supplier unit s when c requires exclusive access to s.

To illustrate these ideas and Szumo design artifacts, we show a UML design for a solution to the classic dining philosophers problem (Fig. 1). A *unit-class diagram* (top left) documents unit classes and relationships between them. The stereotypes ⟨⟨`synchronization`⟩⟩ and ⟨⟨`process_root`⟩⟩ designate unit classes whose instances are, respectively, units that are sharable among multiple threads, and units that serve as non-shared "roots" of a given thread. We show condition variables as class attributes, unit variables as directed associations, and synchronization constraints as limited propositional formulae over condition variables and unit variables, formed using an *entailment operator* "`==>`". The set of synchronization constraints associated with a client class is shown adjacent to that class. Thus, in this design, philosopher units execute in different threads and may invoke operations on shared fork units bound to their `left` and `right` unit variables. Associated with each philosopher, condition variable `eating` signifies when the philosopher needs exclusive access to its forks. The philosopher's synchronization constraint asserts that, if `eating` is true, the philosopher entails its left and right forks.

A *sync-state diagram* (Fig. 1, bottom left) shows how operations affect the values of condition variables. The sync-states are annotated with the condition variables they map to true. The transitions carry annotations to designate when they are taken. An arrow with no source sync-state marks the initial sync-state. Thus, `eating` is false in `s0`, the initial sync-state of a philosopher unit, and true in `s1`. When in `s0`, a philosopher transitions into `s1` (immediately) before invoking its `eat` operation, and transitions back to `s0` (immediately) upon returning. Together, the unit-class and sync-state diagrams document that a philosopher entails its forks while executing an `eat` operation.

Unit-class and sync-state diagrams are reusable over many different programs. In contrast, a *unit-instance diagram* (Fig. 1, right) captures a particular configuration of synchronization units representing the initial state of a design to be checked for deadlock.

2.2. Overview of Negotiation

In a Szumo design, the set of units that a thread needs to access can be inferred at run time. For instance, from the design artifacts in Fig. 1, we infer that a thread needs only its root unit, except when executing the root's `eat` operation, when it needs the `Fork` units referenced by the root's `left` and `right` unit variables. More generally, a thread needs its root unit, as well as any unit entailed by a unit that it needs. The conjunction of the synchronization constraints associated with the units that a thread needs defines the thread's synchronization contract. The contract changes dynamically as the units that the thread executes modify condition variables and unit variables.

At run time, the Szumo middleware associates with each thread a set of units, called the thread's *realm*. When a thread is executing, it is allowed to access all units in its realm, and prevented from accessing any units not in its realm. Szumo provides a strong guarantee of non-interference by preventing realms from overlapping—i.e.,

[1] called the "universe model" in [3]

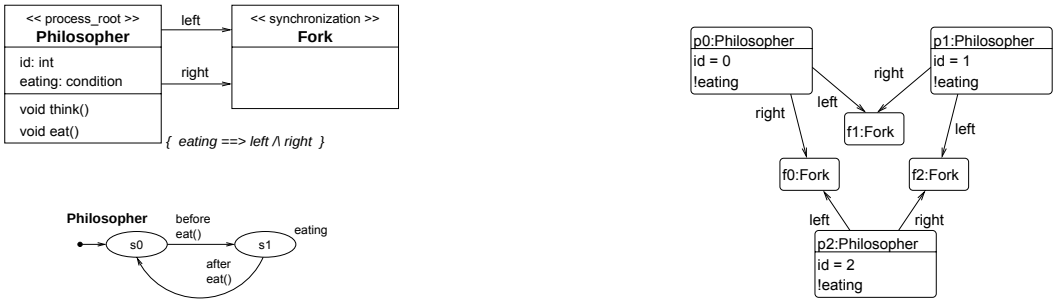


Figure 1. Unit-Class (top left), Sync-state (bottom left), and Unit-Instance (right) Diagrams

by preventing any unit from simultaneously being in the realm of more than one thread. Changes in a unit's synchronization-relevant state may result in a thread's realm containing units that it no longer needs. To maximize concurrency, such units should be *released*, or migrated out of the thread's realm. Changes in a unit's synchronization-relevant state may also result in a thread needing some units that are not in its realm. Such units must be *acquired* and migrated into the realm before the thread can be scheduled. Because some needed units may be contained in the realms of other threads, it may not be possible to immediately acquire all needed units. When a thread's realm contains exactly the set of needed units, we say that the realm is *complete*; otherwise we say it is *damaged*.

From a user's perspective, a thread executes within a complete realm until it performs an operation that changes the synchronization-relevant state of a unit, thereby damaging the realm. For instance, consider a thread whose initial realm contains just a philosopher unit p in sync-state `s0`. Then p's entailment is the empty set and so, according to the thread's contract, this initial realm is complete. However, by invoking the `eat` operation, the thread changes p's sync-state to `s1`, thereby changing its entailment and damaging the realm. The semantics of Szumo dictate that a thread with a damaged realm must block until the realm can be *repaired*. Repairing a damaged realm involves releasing and/or acquiring units to make the realm complete. In our example, the units referenced by p's `left` and `right` unit variables would need to be acquired and migrated into the realm containing p. To safely migrate units, a thread must negotiate with other threads.

In a Szumo program, the middleware enforces and negotiates contracts on behalf of threads. It enforces a thread's contract by confining the thread to execute within its realm.[2] The negotiation of a thread's contract proceeds according to a decentralized two-phase protocol. In the initial *contraction phase*, the middleware releases from the realm all units that are no longer needed. Then, in the *expansion phase*, the middleware attempts to incrementally acquire units not in the realm but in the entailment of some unit in the realm and migrate them into the realm. The expansion phase is complicated by the need to prevent the introduction of concurrency errors in incrementally acquiring units.

[2] it traps accesses by a thread to a unit, raising a run-time exception if the unit is not contained in the realm

2.3. Szumo Negotiation Predicates

For concurrency analysis, we do not model the full mechanics of negotiation, but only the effects of negotiation on the realms and execution status of threads. We define such effects using four predicates. The first predicate represents the entailment information derived from a design: *entails*(u, v) asserts that u entails v, for units u, v. This predicate is used to define completeness: A set of units R is complete for a thread t if it is the least set satisfying

- $r \in R$, where r denotes the root unit of t and
- $u \in R$ and *entails*(u, v) implies $v \in R$, for units u, v

The remaining predicates involve a thread t and a unit u:

- *holds*(t, u) asserts that u is in the realm of t
- *needs*(t, u) asserts that, if a set R is complete for t, then $u \in R$
- *waits*(t, u) asserts that t needs u but that some other thread t' holds u, where $t \neq t'$

When the realm of a thread t is first damaged, the realm may contain units u such that *holds*(t, u), but not *needs*(t, u). Operationally this means that the realm must be contracted. During expansion, any units u satisfying *needs*(t, u), but not *holds*(t, u), must be migrated into the realm. The realm is complete when *holds*(t, u) and *needs*(t, u) are equivalent, for all u. A deadlocking state is one in which there exists a cycle of dependencies between the *waits* and *holds* relations of two or more threads.

2.4. Constraint Handling Rules

Constraint Handling Rules (CHR [8]) is a declarative language extension designed to write application-tailored constraint solvers. It specifies multi-headed guarded rules that re-write constraints until they reach a solved form. The rules define valid transformations of constraints collected in a dynamically changing *constraint store*. When a constraint is *posed*, it is added to the constraint store, and then the rules are applied exhaustively until either no more transformations can be performed, in which case the evaluation succeeds, or the constraint store becomes inconsistent, in which case the evaluation fails.

A CHR rule has the form

```
Label @ Head T Guard | Body.
```

where `Label` names the rule; `Head` identifies stored constraints subject to transformation; `T` signifies the kind of transformation to be performed, either *simplification*, written "`<=>`", or *propagation*, written "`==>`"; `Guard` is a condition that must hold for the rule to apply; and `Body` specifies constraints to be posed.

A rule is applied if the posed constraint matches a constraint in the rule's head, the store contains constraints that match the other head constraints, and the rule's guard is satisfied. A simplification rule requires that all head constraints be removed from the constraint store and the constraints in the body of the rule be posed. A propagation rule requires that the body constraints be posed but the head constraints remain in the store. Thus, simplification replaces some stored constraints with others, while propagation adds to the store new constraints which may cause further simplifications.

Table 1 shows some of the rules that we use in defining the Szumo negotiation protocol (Section 3). `Lift` states that for every pair of stored constraints of the form `entails(U,`U_1`)` and `holds(T,U)`, a constraint `needs(T,`U_1`)` should be posed. `NonInt` applies whenever the store contains `holds(`T_1`,U)` and `holds(`T_2`,U)` such that $T_1 \neq T_2$, in which case the evaluation fails. `Acquire` and `Block` replace stored constraints matching the left-hand side with the constraints indicated by the corresponding right-hand side.

3. Deriving models from designs

We model the behavior of a Szumo system as an *extended finite automaton* (EFA) whose states represent configurations of threads, each executing within a unit in its realm. Transitions of this EFA reflect synchronization-relevant changes in the state of one or more threads and thus must respect the negotiation semantics of unit migration as dictated by synchronization constraints. This semantics is

```
Lift @ entails(U,U1),holds(T,U)
       ==> needs(T,U1).
Acquire @ needs(T,U) <=> holds(T,U).
NonInt @ holds(T1,U),holds(T2,U)
         <=> T1≠T2|fail.
Block @ needs(T,U) <=> waits(T,U).
```

Table 1. CHR for thread negotiation

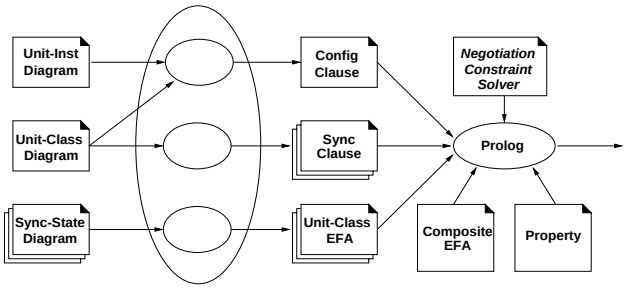


Figure 2. Tools and artifacts in our process

obtained by extending the EFA-based verification framework of [14] to support inter-thread negotiation.

Fig. 2 outlines the high-level architecture of our D4V framework. Szumo design artifacts (far left) are translated into Prolog modules, which instantiate a generic deadlock-detection framework with the details of the system to be analyzed. Below we describe each of the generated Prolog modules. As of the time of writing, the translation (unlabelled ovals) from Szumo design artifacts to Prolog modules is performed by hand.

3.1. Problem-specific framework modules

For each unit class, a *Unit-Class EFA* module defines two relations to use in modeling the units' behaviors: `trans` and `inv`. The `trans` relation represents the transition relation defined in the unit class' sync-state diagram. For example, the following rules are generated from the sync-state diagram for a philosopher (Fig. 1)

```
trans(philosopher, P, s0, before_eat, s1).
trans(philosopher, P, s1, after_eat, s0).
```

where `philosopher` names the associated unit class, `P` identifies a specific unit-class instance, and `s0`, `s1` and `before_eat`, `after_eat` correspond to sync-state labels and transition labels, respectively.

The `inv` relation associates each sync-state with an *invariant* specifying the values of the unit's condition variables in the sync-state. For example,

```
inv(philosopher, s0, [], [eating]).
inv(philosopher, s1, [eating], []).
```

signifies that, when a philosopher unit is in `s0`, `eating` is false, and when the unit is in `s1`, `eating` is true.

Each synchronization constraint is translated into a *Sync(hronization) Clause* which is used in computing a unit's entailments. For example, "`eating ==> left` $\wedge$ `right`" yields

```
synch(P,eating,[L,R]) :-
  left(P,L), right(P,R).
```

where `P` identifies a philosopher unit, `L` and `R` reference the unit's left and right fork, and `left` and `right` are CHR constraints, posed here to retrieve the units bound to the corresponding unit variables.

The unit-instance diagram manifests as a *Config(uration) Clause* which records the inter-unit associations through unit variables. For example, the unit-instance diagram in Fig. 1 translates into

```
left(P0,F1).  left(P1,F2).  left(P2,F0).
right(P0,F0). right(P1,F1). right(P2,F2).
```

3.2. Generic framework modules

Unit-level EFAs are aggregated into thread-level EFAs, which are then composed to form a global EFA. The composition is performed by the *Composite EFA* module, which computes successor states in the global EFA based on the successor states in the thread EFAs and the protocol of inter-thread negotiation defined by the *Negotiation Constraint Solver* (Section 3.3).

A transition in the global EFA is constructed from a single *execution step* in the unit-class EFA of a distinguished *active* unit. An execution step is a unit-level transition into a new sync-state followed by an update of the active unit's condition variables in accordance with the unit's `trans` and `inv` relations. These updates trigger the addition and/or removal of `entails` constraints to reflect changes in the unit's entailment.

The *Property* module checks each newly computed system state for deadlock. It searches the constraint store for a set of constraints defining a cycle of *holds* and *waits* dependencies.

3.3. Negotiation Constraint Solver

A change to the entailment of a unit may damage the realm of the thread that owns it. Repairing a damaged realm requires inter-thread negotiation. The *Negotiation Constraint Solver* is a Prolog module reflecting the semantics of negotiation. It is used at each step of system execution to update constraints representing the predicates in Section 2.3.

We generate the negotiation constraint solver from a CHR program. Table 1 shows some of the rules in this program. Intuitively, the `Lift` rule poses `needs(T,`U_i`)` for each stored `entails(U,`U_i`)`. In posing `needs(T,U)`, the `Acquire` rule attempts to replace `needs(T,U)` with `holds(T,U)` signifying that `U` is in `T`'s realm. However, if the constraint store already contains a constraint `holds(`T'`,U)` for some thread $T' \neq T$, the `NonInt` rule causes this attempt to fail. In this case, the `Block` rule replaces `needs(T,U)` with `waits(T,U)`, signifying that `T` blocks waiting for `U`.

4. Discussion and Related Work

This paper builds upon the separation of concerns and run-time guarantees provided by Szumo and the declarative nature of CHR to enable an approach for detecting deadlock in designs of multi-threaded systems. Our D4V approach extends the EFA-based verification system of [14] to simplify extraction of finite state models from Szumo designs. In Szumo designs, synchronization concerns are separated from the "core-functional logic" of programs and expressed as synchronization contracts, which are to be automatically negotiated at run time by middleware. The verification system is extended with a constraint solver that is generated from a CHR program encoding the semantics of synchronization contracts. The simplicity of this encoding contributes to the transparency of our approach and facilitates traceability between designs and their models. Additionally, the verification system constructs models on-the-fly [10], elaborating execution paths incrementally and only to the point where either a deadlock is detected or extending a path would violate a synchronization contract. Although not discussed in this paper, our D4V approach also supports checking of more general properties than just deadlock by substituting appropriate property modules in Fig. 2.

Concurrency analysis tools such as Bandera [6] and Java PathFinder (JPF) [9] extract finite state models directly from code, instead of designs, to ensure fidelity between the code and the model. JPF translates programs written in a Java subset to PROMELA in order that the SPIN model checker [10] can detect deadlocks and violations of boolean assertions. Bandera incorporates a variety of program analysis, abstraction, and transformation techniques for extracting conservative finite-state models from Java source code, and a variety of backends targeting input languages of different model checkers. The translations implemented by JPF and Bandera are complex, obstructing transparency and complicating traceability. The models automatically generated by these tools also tend to be fairly low-level. To counter this latter problem, Bandera relies on a software analyst to indicate additional abstractions it should perform. In contrast, our D4V approach generates models from de-

sign artifacts that express synchronization concerns declaratively and at a higher level than code. Such artifacts should be more amenable to transparent generation of models at a level of abstraction suitable for concurrency analysis.

Several other approaches separate synchronization concerns for purposes of verification. One notable approach, SyncGen, generates synchronization code that is woven into a subject program from declarative specifications of *region invariants* [7]. A second, synthesizes *concurrency controllers* to coordinate threads in accessing protected resources from global policy specifications, expressed using high-level guarded commands based on a set of pre-defined patterns, and controller interface specifications, expressed as finite state machines [4]. In both approaches, the separation of concurrency concerns affords more modular reasoning. However, the specifications of concurrency concerns in these approaches are global in that they describe possible accesses made by all threads to supplier objects. As such, they are more expressive than, but they lack the compositionality of synchronization contracts. Furthermore, neither of the approaches provides protection from data races if an error is made in specifying concurrency concerns and neither directly supports analysis for deadlocks.

Magee and Kramer propose a model-based approach to D4V that separates synchronization and functional concerns [11]. The designer first models a system as the parallel composition of finite state processes (FSP) and analyzes the model for safety and progress properties using their LTSA tool. Once verified, the processes are implemented as monitors in Java. The translation from an FSP model to Java is idiomatic, not automatic. Additionally, the FSP model is intentionally operational.

An approach that supports D4V extends Java extends Java with annotations to associate locks with program fields and methods in order to support the detection of race conditions [1]. The approach uses static analysis to track the set of locks held at each program point and supports both client- and supplier-side synchronization. Annotations are supplied by a designer or inferred automatically, which, for the price of false positives, makes the technique applicable even for very large programs. Our approach, by contrast, uses analysis to detect deadlock leaving protection against data races to the middleware.

Finally, we note that others have used CHR to reason about concurrent interactions. Alberti and colleagues verify that multi-agent configurations behave according to prescribed protocols [2]. However, they do not use constraint programming to directly model the semantics of agent interaction, as we use it to model the semantics of thread negotiation.

References

[1] M. Abadi, C. Flanagan, and S. N. Freund. Types for safe locking: Static race detection for java. *ACM Trans. Program. Lang. Syst.*, 28(2):207–255, 2006.

[2] M. Alberti et al. Specification and verification of agent interaction protocols in a logic-based system. In *Proc. ACM Symp. Applied Computing*, pages 72–78, 2004.

[3] R. Behrends and R. E. K. Stirewalt. The Universe Model: An approach for improving the modularity and reliability of concurrent programs. In *Proc. ACM SIGSOFT Symp. Foundations Softw. Eng.*, 2000.

[4] A. Betin-Can and T. Bultan. Verifiable concurrent programming using concurrency controllers. In *Proc. Conf. Automated Softw. Eng.*, pages 248–257, 2004.

[5] T. Bultan and A. Betin-Can. Scalable software model checking using design for verification. In *Proc. IFIP Working Conf. Verified Softw.: Theories, Tools, Experiments*, 2005.

[6] J. C. Corbett et al. Bandera: extracting finite-state models from java source code. In *Proc. Intl. Conf. Softw. Eng.*, pages 439–448, 2000.

[7] X. Deng et al. Invariant-based specification, synthesis, and verification of synchronization in concurrent programs. In *Proc. Intl. Conf. Softw. Eng.*, 2002.

[8] T. Frühwirth. Theory and practice of constraint handling rules. *Jrnl. Logic Prog.*, 37(1-3):95–138, October 1998.

[9] K. Havelund and T. Pressburger. Model checking Java programs using Java PathFinder. *Intl. Jrnl. Softw. Tools for Technology Transfer*, 2(4), Apr. 2000.

[10] G. Holzman. *The SPIN Model Checker*. Addison-Wesley Inc., 2004.

[11] J. Magee and J. Kramer. *Concurrency: State Models and Java Programs*. John Wiley and Sons, 2000.

[12] P. Mehlitz and J. Penix. Design for verification using design patterns to build reliable systems. In *Proc. ICSE Workshop Component-Based Softw. Eng.*, 2003.

[13] R. H. B. Netzer and B. P. Miller. What are race conditions?: Issues and formalizations. *ACM Letters Prog. Lang. and Systems*, 1(1):74–88, Mar. 1992.

[14] B. Sarna-Starosta and C. Ramakrishnan. Constraint-based model checking of data-independent systems. In *Proc. Intl. Conf. Formal Eng. Methods (ICFEM)*, pages 579–598, 2003.

[15] N. Sharygina, J. C. Browne, and R. P. Kurshan. A formal object-oriented analysis for software reliability: Design for verification. In *Proc. FASE*, pages 318–332, 2001.

A PVS Approach to Verifying ORA-SS Data Models

Scott Uk-Jin Lee, Gillian Dobbie, Jing Sun
Department of Computer Science
The University of Auckland
Private Bag 92019
Auckland, New Zealand
{scott,gill,jing}@cs.auckland.ac.nz

Lindsay Groves
School of Mathematics, Statistics
and Computer Science
Victoria University of Wellington
Wellington, New Zealand
lindsay@mcs.vuw.ac.nz

Abstract

The rapid growth of the World Wide Web has resulted in a dramatic increase in semistructured data usage. This creates a growing need for ensuring consistency of the data especially when applications or databases change the schema of semistructured data. In this paper, we demonstrate an approach to formally define and verify the ORA-SS data model for semistructured data design. A mathematical semantics for the ORA-SS notation is defined using the PVS formal language. With this semantics, it is possible to perform automated verification via the PVS theorem prover to identify the inconsistencies in designing ORA-SS data models. The verification is based on criteria for semistructured data verification at both schema and data instance levels.

1. Introduction

The rapid growth of the World Wide Web and its technologies has resulted in enormous amounts of data being used over the Internet by Web Services and Web-based applications. The increase in semistructured data usage is not limited to Web applications but expands into various other applications such as digital libraries, biological databases and multimedia data management systems. This expansion of semistructured data usage creates the need for effective and efficient utilization of semistructured data [16].

With such a rapid increase in its usage semistructured data needs to be stored, manipulated, and queried to be utilized properly by various applications and tools. For these purposes, many researchers have proposed to design and develop adequate database systems for semistructured data. As a result, several database systems have already been developed for eXtensible Markup Language (XML) [6], which is a common representation for semistructured data, while traditional database companies, such as Oracle, have provided XML support for their existing database systems.

In order to verify whether tools that transform the schema of semistructured data produce correct results, we need to have a standard representation of schemas and the transformation operators that are used to transform schemas. This process can be achieved by describing a formal model for semistructured data schemas and verifying that instances of schemas conform to the schema model. Then the basic transformation operators can be formally defined on this schema model. In this paper we undertake the first step of the process defined above.

The aim of this paper is to model the Object Relationship Attribute model for semistructured data (ORA-SS) data modeling language [5, 10] in a formal manner, and demonstrate that it is possible to verify data models in this notation using automated tool support. We adopt the Prototype Verification System (PVS) [11] and its theorem prover to achieve this goal.

The ORA-SS data modeling language is used because it not only captures the constraints that are represented in textual languages such as XML Schema [13] but also it is a diagrammatic notation which can be used for conceptual modelling. For formal verification, PVS is used since it has proven its effectiveness by providing precise formal definitions and powerful automated verification support in various other research projects [12, 7, 14]. Using PVS to define and verify the mathematical semantics of ORA-SS and its schema instances will surely provide a solid stepping stone for verifying the results of schema transformations for semistructured data.

There has been other research that provides a formal semantics for semistructured data. For example, the formalization of DTD (Document Type Definition) and XML declarative description documents using expressive description logic has been presented by Calvanese et al. [3]. Anutariya et al. presented the same formalization using a theoretical framework developed using declarative description theory [1]. Also spatial tree logics has been used to formalize semistructured data by Conforti and Ghelli [4].

More recently, hybrid multimodal logic was used to formalize semistructured data by Bidoit et al. [2]. We also applied a similar approach to formalize semistructured data using ORA-SS and Z/EVEs [8]. While this work has helped us develop a better understanding of the semantics of semistructured data, none of it has applied automated verification. In another research we presented a formalization of the ORA-SS notation in Alloy [15]. Although the automated verification was available using the Alloy Analyzer, it had a scalability problem making the verification impossible for a large set of semistructured data. Hence, none of the related work can be adequately used to automatically verify whether a set of transformations on a schema for semistructured data produces the correct results.

The rest of the paper is organized as follows. Section 2 presents the background knowledge on ORA-SS notation and the PVS formal language. Section 3 presents the verification criteria for ORA-SS data models. Section 4 presents a formal semantics of the ORA-SS language in the PVS formal language. The standard PVS syntax [11] is used in specifying formal semantics of the ORA-SS language and examples. In section 5, we demonstrate the verification of ORA-SS schema model using the PVS theorem prover. Section 6 concludes the paper and describes future work.

2. Background

2.1. ORA-SS Data Modeling Language

The Object-Relationship-Attribute model for Semistructured data (ORA-SS) is a semantic enriched data modeling language for semistrctured data design [5]. It has been used in many XML related database applications [9, 10]. The ORA-SS notation consists of four basic concepts: object classes, relationship type, attributes and references.

- An object class represents an entity type in a data model. It is denoted as a labeled rectangle in an ORA-SS diagram.
- A relationship type represents a nesting relationship among object classes. It is described as a labeled edge by a tuple (*name, n, p, c*), where the *name* denotes the name of relationship type, integer *n* indicates degree of relationship type, *p* represents participation constraint of parent object class in relationship type and *c* represents participation constraint of child object class in relationship type.
- Attributes represent properties and are denoted by labeled circles. An attribute can be a key attribute which has a unique value, which is represented as a filled circle. Other types of attributes include single valued attribute, multi-valued attribute, required attribute, composite attribute, etc. An attribute can be a property of an object class or a property of a relationship type.
- An object class can reference another object class to model recursive and symmetric relationships. This can reduce redundancy especially in many-to-many relationships. References are represented by labeled dashed edges.

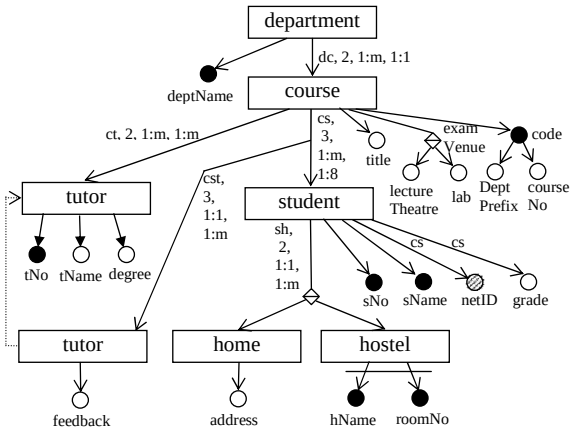


Figure 1. An ORA-SS Schema Diagram.

For the design of semistructured data, an ORA-SS schema diagram specifies the relationships, participation and cardinality constraints among the instances of the object classes in a semistructured data model. For Example, Figure 1 presents an ORA-SS schema diagram of a *Department-Course* data model. In the above schema diagram, there are three semantic errors - (1) the degree of relationship '*cs*' is described as 3, representing a ternary relationship where it actually is binary; (2) there are two attributes described as primary key for the object class '*student*'; (3) the candidate key '*netID*' is represented as an attribute of the relationship '*cs*' where it really is an attribute of the object class '*student*'. These errors can be picked up by examining the diagram since it is a relatively small sized model. However, examining complicated and large ORA-SS schema model for semistructured data is almost impossible. Furthermore, manual diagrammatic checking does not guarantee the consistency of the schema since it is likely that inconsistencies are not revealed when the schema is large and complicated. Therefore, automated verification support based on the formal specification of ORA-SS semantics can be used to reveal the inconsistencies in these schema models as well as in corresponding data instance models.

2.2. Prototype Verification System (PVS)

PVS is a typed higher-order logic based verification system where a formal specification language is integrated with support tools and a theorem prover [11]. It provides formal specification and verification through type checking and theorem proving. PVS has a number of language constructs including user-defined type, built-in types, functions, sets, tuples, records, enumerations, and recursively-defined data types such as lists and binary trees. With the provided language constructs PVS specifications are represented in parameterized theories that contains assumptions, definitions, axioms, and theorems. Many applications have adopted PVS to provide formal verification support to their system properties [7, 12, 14].

3. Verification Criteria for ORA-SS

As we mentioned earlier, ensuring that there are no possible inconsistencies in either the ORA-SS schema diagram or the data instance is a major concern for applications and databases that uses semistructured data especially when the schema of the semistructured data is transformed.

3.1. ORA-SS Schema Verification

Verification of schema for semistructured data involves checking whether a customized ORA-SS schema diagram is consistent according to the semantics of the ORA-SS language. Some guidelines for verifying an ORA-SS schema diagram are as follows.

- In a relationship type, the child object class must be either related to another parent object class to form a binary relationship or related to another sub-relationship type to form a relationship type of degree 3 or more.
- The degree of a binary relationship is 2, ternary is 3 and n-ary is n.
- In a disjunctive relationship type, the child participants are a set of disjunctive object classes.
- A composite attribute or disjunctive attribute has an attribute that is related to two or more sub-attributes.
- A candidate key of an object class is selected from the set of attributes of the object class.
- A composite key is selected from 2 or more attributes of an object class.
- There can only be one primary key per object class. And it can be either a candidate key or a composite candidate key.
- Relationship attributes have to relate to an existing relationship type.
- An object class can reference one object class only, but an object class can be referenced by multiple object classes.

The above are some of the criteria for verifying a correct schema diagram against the ORA-SS notation. For example, the above criteria can be applied to provide consistency checking for the *Department-Course* schema diagram in Figure 1. The inconsistencies in the example include representing the degree as 3 for binary relationship, representing object classes with multiple primary keys, and representing a relationship attribute as a candidate key.

3.2. ORA-SS Instance Verification

The ORA-SS instance verification is aimed to check whether there are any inconsistencies in a semistructured data instance with regard to the designated ORA-SS schema model. Some guidelines for validating an ORA-SS instance are as follows.

- Relationship instances must conform to the parent and child participation constraints, i.e., the number of child objects related to a single parent object or relationship instance should be consistent with the parent participation constraints; and the number of parent objects or relationship instances that a single child object relates to should be consistent with the child participation constraints.
- In a disjunctive relationship, only one object class can be selected from the disjunctive object class set and associated to a particular parent instance.
- For a candidate key (single or composite), its value should uniquely identify the object that this key attribute belongs to.
- Each object can have one and only one primary key.
- All attributes have their own cardinality and the number of attributes that belong to an object should be limited by the minimum and maximum cardinality values of the attribute.
- For a set of disjunctive attributes, only one of the attribute choices can be selected and associated to an object instance.

4. Formal Semantics of ORA-SS Data Model

The effective and efficient verification of any semistructured data schema and its instance requires semantics of a

semistructured data modeling language to be specified in a formal manner. In this section, we present a PVS semantics of the ORA-SS language, which encodes the previously defined criteria to enable the required verification to be carried out.

4.1. Basic Types

Initially basic types used in the ORA-SS data modeling language must be identified and defined prior to constructing the formal representation.

```
OC: TYPE+                 oc: VAR OC
O: TYPE+                  o: VAR O
ATT: TYPE+                att: VAR ATT
ATTVALUE: TYPE+           attVal: VAR ATTVALUE
```

The object types and attribute types defined above represent the non empty set of object classes, object instances, attributes and attribute values respectively.

4.2. Relationship Type

A relationship type is defined as a list of finite set of object classes.The Axiom of the relationship describes that object classes can be related to other object classes as well as to other relationships.

```
no_cycle_oc(loc): RECURSIVE bool = CASES loc OF
  null: TRUE,
  cons(ocs, subloc): (∀(subocs: finite_set[OC]):
  member(subocs, subloc) ⊃ disjoint?(ocs, subocs))
  ∧ no_cycle_oc(subloc)
ENDCASES MEASURE length(loc)

Relationship: TYPE =
  {ocsList: list[finite_set[OC]] | (ocsList ≠ null)
  ∧ (length(ocsList) > 1) ∧ (no_cycle_oc(ocsList))}

Relationship_Ax: AXIOM
  ∀(rel: Relationship): (length(rel) > 2) ⊃
    (∃ (subRel: Relationship): subRel = cdr(rel))
```

In an ORA-SS schema diagram, there are two types of relationship, i.e., a normal relationship where the child participant is a single object class; and a disjunctive relationship where the child participant is a set of disjunctive object classes. The above definition includes both cases.The '*no_cycle_oc*' function is defined as a recursive predicate function to disallow repetition of object classes in a relationship.The relationship axiom states that for any relationship with more than two elements the tail of the list forms another sub-relationship type.

4.3. Instances of Object Class

In ORA-SS data model, an object class has instances which are objects.

```
objInstance(oc): set[O]

objInstance_Ax: AXIOM
  ∀(oc1, oc2: OC): oc1 ≠ oc2 ⊃
    disjoint?(objInstance(oc1), objInstance(oc2))
```

The above function takes in an object class as an argument and returns a set of objects which refers to all the instances of the object class. The axiom of the object instance function defines that any two different object classes should have different set of objects as its instances.

4.4. Instance of a Relationship

In ORA-SS data model, a relationship type also has its instances, which consists of the set of object instances from their corresponding object classes in the relationship.

```
relInstance(rel): set[ObjRelationship] =
  {oRel: ObjRelationship | length(oRel) = Degree(rel)}

relInstance_Ax1: AXIOM
  ∀(ocRel: Relationship), (oRel: ObjRelationship):
    (oRel ∈ relInstance(ocRel)) ⊃ ∃_1((singleton[boolean]
      ((∃ (oc1: OC): (oc1 ∈ car(ocRel)) ∧
      (car(oRel) ∈ objInstance(oc1))))) ∧
      (IF (Degree(ocRel) = 2) THEN (∃ (oc2: OC):
        (oc2 ∈ car(cdr(ocRel))) ∧
        (car(cdr(oRel)) ∈ objInstance(oc2)))
      ELSE (cdr(oRel) ∈ relInstance(cdr(ocRel)))
      ENDIF)

relInstance_Ax2: AXIOM
  ∀(rel1, rel2: Relationship): rel1 ≠ rel2
    ⊃ disjoint?(relInstance(rel1), relInstance(rel2))
```

The above defines relationship having instances as a function where it takes in a relationship as an argument and returns a set of object relationships. The first axiom for relationship instance defines that child object instance should be an instance of the associated child object classes in the relationship. It also provides more definitions about parent object instance using two cases: if the degree of the relationship is binary, the parent object instance should be an instance of the parent object class; if the degree of the relationship is ternary or more, the second part of the predicate recursively defines that the sub-relationship instance is an instance of the sub-relationship type. Finally, the second axiom defines that any two different relationship types should have their own disjoint set of instances.

4.5. Participation Constraints on Relationship Type

Every relationship type in an ORA-SS schema diagram has its associated constraints on its participating objects shown using the `min:max` notation. It constrains the number of child objects that a parent object can relate to and vice versa.

```
parentSet(orSet: set[ObjRelationship], lo):
  set[ObjRelationship] = {oRel: ObjRelationship |
    ((oRel ∈ orSet)) ∧ (cdr(oRel) = lo)}

parentConstraints(rel): [nat, posnat]

parentConstraints_Ax: AXIOM
  ∀(ocRel: Relationship), (oRel: ObjRelationship):
    (oRel ∈ relInstance(ocRel)) ⊃
      is_finite(parentSet(relInstance(ocRel), cdr(oRel)))
    ∧ PROJ_1(parentConstraints(ocRel)) ≤
        Card(parentSet(relInstance(ocRel), cdr(oRel)))
    ∧ PROJ_2(parentConstraints(ocRel)) ≥
        Card(parentSet(relInstance(ocRel), cdr(oRel)))
```

The above defines parent constraints as a function where it takes a relationship type as an argument and returns the tuple of natural number and positive natural number which refers to a `min:max` pair. The axiom of the function defines that the number of relationship instances for each object of the parent object class or each relationship instance of the sub-relationship type should be within the boundaries defined in the relationship. The child constraints of the relationship and constraints on attribute values associated with objects and relationships can be defined in a similar way.

4.6. Candidate Key and Primary Key

An object can have an attribute or set of attributes that have a unique value for each instance of an object class called a candidate key.

```
CandidateKey: TYPE = {cKey: [OC, set[ATT]] |
  (∃ (oAtt: ObjAttribute): PROJ_1(cKey) =
   PROJ_1(oAtt) ∧ (PROJ_2(oAtt) ∈ PROJ_2(cKey)))}

CandidateKey_Ax: AXIOM
  ∀(cKey: CandidateKey),(a: ATT):(a ∈ PROJ_2(cKey))
    ⊃(∀(o1, o2:O):((o1 ∈ objInstance(PROJ_1(cKey)))∧
    (o2 ∈ objInstance(PROJ_1(cKey)))∧(o1≠o2))
      ⊃ IF (oAttValSet(cKey, o1) ≠ oAttValSet(cKey, o2))
        THEN (o1≠ o2) ELSE o1 = o2 ENDIF)
```

The above defines the object having a candidate key as a tuple type consisting of an object class and a set of attributes. The tuple contains a set of attributes that denotes both a candidate key and a composite candidate key. The axiom for candidate key defines two facts. It states two objects are different when values of the candidate key for each object are different; and two objects are the same when values of the candidate key for each object are the same, where the values of the candidate keys belongs to the set of attribute values of the object attributes. In ORA-SS schema diagrams, an object class has a primary key which is selected from the set of candidate keys. The primary key is defined as a tuple type consisting of an object class and a set of attributes which refer to the primary key that belongs to the object. Due to the space limit, not all the semantics definitions are presented here. A complete PVS sematics of the ORA-SS constructs can be found at `www.cs.auckland.ac.nz/~scott/files/seke06/PVS-ORA-SS.pdf`.

5. Formal Verification of ORA-SS data model

With the formal PVS ORA-SS semantics, verification of ORA-SS schema and its data instance can be performed using the PVS theorem prover. For example, we can represent the *Department-Course* schema diagram in Figure 1 and verify it. [1] The PVS theorem prover returns an error message as shown below.

```
Incompatible types for cs
   Found: orass.Relationship
   Expected: orass.OC
```

The error message indicates that '*studentCandKey3: CandidateKey = (cs, netID)*' is an invalid representation. Since the PVS semantics for candidate key only allows objects to have a candidate key, representing the relationship attribute '*netID*' as a candidate key can be immediately picked up by the PVS type checker.

After removing the above incorrect representation, type checking the specification will return several Type Correctness Conditions (TCCs) which represents obligations that need to be proved. The TCCs produced by the above specification includes an unproved TCC on '*studentPrimKey2: PrimaryKey = (student, sName)*'. Using the PVS theorem prover to prove the TCCs with various commands and related axioms will result in an unprovable state as shown in Figure 2.

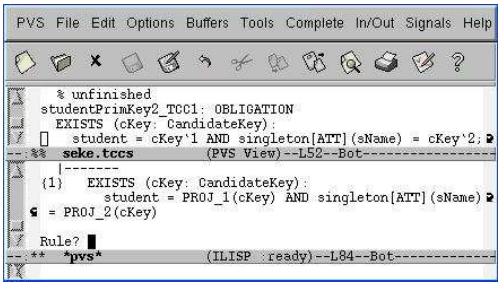


Figure 2. Verifying the Primary Key of the Student Object Class

It shows that according to the '*PrimaryKey*' definition it requires '*sName*' to be a candidate key of '*student*' for '*sName*' to be a primary key of 'student'. But there is no axiom or specification, stating '*sName*' as a candidate Key of

[1] Due to the space limit, the complete PVS representation of schema diagram in Figure 1 can be found at `www.cs.auckland.ac.nz/~scott/files/seke06/PVS-Dept-Course.pdf`.

'*student*'. Hence the TCCs results to be unprovable. Thus the definition '*studentPrimKey2: PrimaryKey = (student, sName)*' is verified to be incorrect, which is consistent with our manual inspection.

The verification of data instances can be achieved similarly to the schema verification. First, an ORA-SS data instance is translated into our PVS representations. Then the translated data instance can be verified using the PVS type checker and theorem prover. With the automated verification support the definitions of ORA-SS semantics can be utilized in a much wider scope.

6. Conclusion

In this paper, we describe an approach to define and automatically verify semistructured data using the ORA-SS data modeling language and the PVS theorem prover. The modeling could not be done before guidelines for verifying the ORA-SS data models were defined at both the schema diagram level and the data instance level. With the guidelines, we defined a formal mathematical semantics for the ORA-SS diagrammatic data modeling notation in PVS. The resulting formal semantics is useful for providing a rigorous formal foundation for the ORA-SS language. Furthermore, such a semantics can be adopted by many semistructured data applications which use the ORA-SS data model. Lastly, we demonstrate some verifications using the PVS ORA-SS semantics in verifying customized ORA-SS schema diagrams. Automated proof steps are presented through a simple ORA-SS data model. More complicated proofs can also be constructed for verifying large semistructured documents. The automated verification provided by PVS empowers our definition of ORA-SS semantics. Without automated verification the application of the definitions would be restricted since manual verification for large documents is time-consuming if not impossible.

In the future, we plan to extend our work on defining the basic transformation operators that are used to transform ORA-SS schemas and providing verification for transformed schemas of semistructured data. We are in the process of developing a theorem library to support the definition of the transformation operators and verification of ORA-SS schema diagrams resulting from transformations. By doing so, verifying the results of applications or databases that transforms the schema of semistructured data can be possible. In addition, we plan to define and verify the extended PVS semantics of the ORA-SS language to model the normalization process in semistructured data design.

References

[1] C. Anutariya, V. Wuwongse, E. Nantajeewarawat, and K. Akama. Towards a Foundation for XML Document Databases. In *EC-Web*, pages 324–333, 2000.

[2] N. Bidoit, S. Cerrito, and V. Thion. A First Step towards Modeling Semistructured Data in Hybrid Multimodal Logic. *Journal of Applied Non-Classical Logics*, 14(4):447–475, 2004.

[3] D. Calvanese, G. D. Giacomo, and M. Lenzerini. Representing and Reasoning on XML Documents: A Description Logic Approach. *Journal of Logic and Computation*, 9(3):295–318, 1999.

[4] G. Conforti and G. Ghelli. Spatial Tree Logics to reason about Semistructured Data. In *SEBD*, pages 37–48, 2003.

[5] G. Dobbie, X. Wu, T. Ling, and M. Lee. ORA-SS: Object-Relationship-Attribute Model for Semistructured Data. Technical Report TR 21/00, School of Computing, National University of Singapore, 2001.

[6] E. R. Harold and W. S. Means. *XML in a Nutshell*. O'Reilly, Sebastopol, 3rd edition, 2004.

[7] M. Lawford and H. Wu. Verification of real-time control software using PVS. In *Proceedings of the 2000 Conference on Information Sciences and Systems*, volume 2, pages TP1–13–TP1–17, Princeton, NJ, mar 2000. Dept. of Electrical Engineering, Princeton University.

[8] S. U. Lee, J. Sun, G. Dobbie, and Y. F. Li. A Z Approach in Validating ORA-SS Data Models. In *3rd International Workshop on Software Verification and Validation*, Manchester, United Kingdom, 2005.

[9] T. Ling, M. Lee, and G. Dobbie. Applications of ORA-SS: An Object-Relationship-Attribute data model for Semistructured data. In *IIWAS '01: Proceedings of 3rd International Conference on Information Integration and Web-based Applications and Serives*, 2001.

[10] T. W. Ling, M. L. Lee, and G. Dobbie. *Semistructured Database Design*, volume 1. Springer-Verlag, 2005.

[11] S. Owre and J. M. Rushby and and N. Shankar. PVS: A Prototype Verification System. In *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, jun 1992. Springer-Verlag.

[12] M. Srivas, H. Rueß, and D. Cyrluk. Hardware Verification Using PVS. In *Formal Hardware Verification: Methods and Systems in Comparison*, volume 1287 of *Lecture Notes in Computer Science*, pages 156–205. Springer-Verlag, 1997.

[13] H. S. Thompson, C. Sperberg-McQueen, N. Mendelsohn, D. Beech, and M. Maloney. XML Schema 1.1 Part 1: Structures. http://www.w3.org/TR/xmlschema11-1/.

[14] J. Vitt and J. Hooman. Assertional Specification and Verification Using PVS of the Steam Boiler Control System. In J.-R. Abrial, E. Boerger, and H. Langmaack, editors, *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165, pages 453–472. Springer-Verlag, 1996.

[15] L. Wang, G. Dobbie, J. Sun, and L. Groves. Validating ORA-SS Data Models using Alloy. In *The Australian Software Engineering Conference (ASWEC 2006)*, Sydney, Australia, 2006.

[16] X. Wu, T. W. Ling, M. L. Lee, and G. Dobbie. Designing Semistructured Databases Using the ORA-SS Model. In *WISE '01: Proceedings of 2nd International Conference on Web Information Systems Engineering*, Kyoto, Japan, 2001. IEEE Computer Society.

Decision Support for Resource-Centric Software Release Planning

Jim Mc Elroy & Guenther Ruhe

Laboratory for Software Engineering Decision Support, University of Calgary
2500 University Drive NW, Calgary, AB T2N 1N4, Canada
{mcelroy, ruhe}@cpsc.ucalgary.ca

Abstract. Release planning for incremental software development assigns features to releases such that most important technical, resource, risk and budget constraints are met. Most organizations perform release planning in an ad-hoc manner, ignoring proper consideration of resources. The result of this process is the consideration of plans that are likely to fail or at least have to be updated several times in the course of their realization.

This paper addresses better integration of the resource management aspects into the decision-making process. For that purpose, it studies scenario-driven analysis of bottleneck resource situations in order to support the process of finding solutions accommodating possible future changes. A hypothetical case study is undertaken to illustrate the approach and to show the added value from a decision-making perspective.

Keywords: Software engineering decision support, release planning, software evolution, resource optimization.

1 Motivation

Release planning for incremental software development assigns features to releases such that most important technical, resource, risk and budget constraints are met. Most organizations perform release planning in an ad-hoc manner, ignoring proper consideration of resources. The result of this is the adoption of plans that are likely to fail or at least have to be updated several times in the course of their realization.

The notion of release planning varies from informal descriptions as performed in agile development to more formalized approaches. A discussion of two fundamental approaches and existing solution methods is given in [16]. Informal planning approaches as described in [3] mainly rely on communication and expert knowledge. Any formalized approach goes beyond that and tries to provide an explicit and quantitative description of the problem. Based on that, mathematical optimization is applicable to generate optimal or approximate solution alternative. This approach is used as the kernel of an evolutionary problem solving method called EVOVLE* [15].

Decision support in software engineering is still in its infancy. Although established already in other field such as transportation, control, health care or defense, there is still a lack of user acceptance as observed by [6]. Decision support in software engineering tries to approach the decision-making nature of many development processes (such as in requirements engineering [1]). It aims at making decisions more effectively and more efficiently. However, do users really accept this paradigm?

An empirical evaluation to compare ad hoc planning and optimization-based planning was done in [7]. According to this, the formal approach performed better in terms of its effectiveness and efficiency. Another focus of this research was to evaluate the human perception of results in terms of confidence, understanding and trust. The results were not so obvious here. An improvement in understanding was achieved when providing more explanation related to the computer-based solutions. However, there was still hesitation to trust (have confidence in) those solutions more than the solutions obtained from the manual procedure.

This paper continues in the research effort to provide meaningful support to the human decision-maker. It studies scenario-driven analysis of bottleneck resource situations intended to support the process of finding solutions accommodating future changes. We study several solutions for improving resource utilization in release plans. These solutions include an objective function for minimizing cost, and an objective function for minimizing wasted resources.

The paper is structured into five sections. Section 2 gives a formal description of resource-centric release planning. The scenario-driven decision support method is described in Section 3. A hypothetical case study is studied in Section 4 to illustrate the approach and to show the added value form a decision-making perspective. Finally, we discuss the results and provide an outlook for future research.

2 Resource-centric Release Planning

In this section, a formal description of release planning as used in this paper is given. The solutions gained from solving this problem are offered to the human decision-maker to get more confidence in solving “the right problem”. Technically, this means either an acceptance of one of the proposed solutions or an adjustment of the formalization. This evolutionary approach of problem

solving for software release planning was introduced in [15].

2.1 Features and Related Decision Variables

The concept of a feature is applicable and important for any software development paradigm. However, it is especially important for any type of incremental product development. Features are the "selling units" provided to the customer. Definitions of features are abundant in the literature. In the context of this research, we follow the definition given by [18] which defines features to be "a logical unit of behavior that is specified by a set of functional and quality requirements".

We assume a set of features F = {f(1), f(2), … , f(n)}. The goal is to assign the features to a finite number K of release options or to decide to postpone the feature. A release plan is characterized by a vector of decision variables x = (x(1), x(2), …, x(n)) with

x(i) = k if feature f(i) is assigned to release option k ∈{1,2,…K}, and

x(i) = K+1 if the feature f(i) is postponed (e.g., not contained in one of the next K releases)

2.2 Stakeholders

Stakeholders are extremely important for performing realistic and customer-oriented release planning. We assume a set of stakeholders S = {S(1),…,S(q)}. Each stakeholder S(p) can be assigned a relative importance λ(p). In general, the scale for expressing relative degree of stakeholder importance is free. Realistically, an ordinal scale seems to be most appropriate. We are using a nine-point scale that gives us sufficient freedom to differentiate. That means each stakeholder importance λ(p) is taken from the set {1,..,9}. The underlying meaning is

- λ (p) = 1 means extremely low importance
- λ (p) = 3 means low importance
- λ (p) = 5 means average importance
- λ (p) = 7 means high importance
- λ (p) = 9 means extremely high importance.

The interpretation of all the even values in between is that they are refinements of the values above and below. We have chosen a nine-point ordinal scale for expressing the degree of importance. This gives sufficient detail to differentiate the importance between stakeholders. However, the whole approach will be applicable in the same way for another scale.

2.3 Prioritization of Features by Stakeholders

In order to select and schedule features, there must be an agreed upon statement of priorities for features. In our proposed model, prioritization by each stakeholder S(p) can be done with respect to different criteria. We define them here again for an ordinal nine-point scale. However, the approach remains applicable if we decide to use other scales. Different criteria for prioritization are possible (but are not limited to that):

- Value (denoted value(i,p) ∈ {1…9}) assigned by stakeholder S(p) to feature f(i). Value refers here to the impact of the feature on the business value.
- Urgency addresses how time-critical a feature is considered. This is expressed by a variable urgency(i,p) ∈ {1…9}) assigned by stakeholder S(p) to feature f(i). An increasing value of urgency corresponds to an increasing degree of urgency.
- Risk risk(i,p) describes the risk assigned by stakeholder S(p) to feature f(i). An increasing value corresponds to a decreasing risk perception. Risk is a fuzzy term incorporating a variety of different aspects. For pragmatic reasons, we keep it simple here and define the risk of a feature as the degree of impact on its realization. Again, we assume that risk(i,p) describes the risk assigned by stakeholder S(p) to feature f(i). An increasing value of risk corresponds to a decreasing risk perception.
- Requirements volatility refers to the stability of a requirement and the likelihood of change. This also has to do how well the requirement is already understood and described. We denote by volatility(i,p) the volatility assigned by stakeholder S(p) to feature f(i). An increasing value corresponds to a decreasing volatility perception.

2.4 Technological Constraints

A study of requirements repositories in the telecommunications domain by Carlshmare et al. [5] concluded that only about 20% of the features were singular or independent of each other. The authors have identified different types of dependencies between features. We consider three types of technological constraints where features can either be in a coupling relation C, in a weak precedence relation WP, or in a strong precedence relation SP. All these relations are subsets of the product set F x F.

Definition 1: Two features f(i) and f(j) are coupled (written as (i,j) ∈ C)) if they are required to be implemented in the same release. This dependency can be due to implementation or usage issues. In terms of the introduced decision variables, this means that

$$x(i) = x(j) \text{ for all } (i,j) \in C \subset F \times F \text{ (Coupling)} \quad (1)$$

Definition 2: Feature f(i) is in weak precedence relation to feature f(j) (written as (i,j) ∈ WP) if feature f(j) can not be delivered in a release earlier than f(i). In terms of the introduced decision variables, this means that

$$x(i) \leq x(j) \text{ for all } (i,j) \in WP \subset FxF \text{ (Weak Precedence)} \quad (2)$$

Definition 3: Feature f(i) is in strong precedence relation to feature f(j) (written as (i,j) ∈ SP) if feature f(j)

has to be delivered in a release later than f(i). In terms of the introduced decision variables, this means that

$x(i) < x(j)$ for all $(i,j) \in SP \subset FxF$ (Strong Precedence) (3)

2.5 Resource Constraints

Different resources are required for the implementation of features. In addition, there are capacity bounds on the amount of resources available in each release cycle. In the general model, we have considered R type of resources tentatively involved in the implementation of features. Correspondingly, we define resource capacities Cap(r,k) for each resource type r = 1,…,R and all releases k = 1,…,K. To become a feasible plan, decision variables must satisfy

$$\sum_{x(i)=k} \text{resource}(i,r) \leq \text{Cap}(k,r) \quad (4)$$

for all releases k=1,…,K and all resource types r=1,…,R.

2.6 Objective Function

Release planning technique must have an approximate definition of objectives. Typically, it is a mixture of different aspects such as value, urgency, risk, satisfaction/dissatisfaction, return on investment, etc. The actual form of the explicit function tries to bring the different aspects together in a balanced way. The objective is the maximization of a function F(x) among all release plans x satisfying the above technological and resource constraints (1) – (4).

F(x) is composed of the weighted average priority vector WAP(j) defined for each feature f(j). Therein, the weighted average priority is a function including the different possible criteria and applying operators such as +, *, power, log, exp, Min, or Max to combine the criteria. For each release option k, parameter $\xi(k)$ describes the relative importance of the release option and its relative impact to the objective function.

$$F(x) = \Sigma_{k=1\ldots K}\, \xi(k)\, [\Sigma_{\,j:\, x(j)=k}\, WAP(j)] \quad (5)$$

Definition 4: The set of release plans x satisfying all constraints (1) – (4) is called feasible area X. A solution $x^* \in X$ maximizing a given objective function F(x) is called optimal. Given an optimal solution x^*, a feasible solution x is called α-qualified, $\alpha \in (0,1)$, if

$$F(x) \geq \alpha F(x^*) \quad (6)$$

3 Solution Approach

3.1 Overview

Addressing resource issues in release planning has been somewhat sparsely covered in the literature. Karlsson and Ryan [10] covered a cost-value approach for prioritizing requirements, but the focus of their paper was prioritizing requirements for release plans via the Analytical Hierarchy Process AHP [17]. The discussion of resources in that paper was largely limited to using resource consumption as the cost factor in the AHP process.

Jung [9] pointed out flaws in the AHP approach, and described release planning as a knapsack optimization problem. That paper described a two-pass approach in which the objective function was first run to place features in a release plan in order to maximize stakeholder value. A second objective function was then used to minimize the resource costs of the project, using the maximized stakeholder value (or a value just below it) as a constraint in the second objective function. However Jung's paper stopped with that solution, and did not explore other factors in resource allocation and utilization.

Bagnall et al. [2] focused largely on the algorithmic issues of solving the release planning problem, describing the problem as an optimization problem. Resources were discussed, but only in terms of resource capacities serving as constraints on the objective function of maximizing stakeholder satisfaction. Greer & Ruhe [8] enhance the discussion of release planning as an optimization problem, describing different types of resource capacities as constraints on the problem and allowing stakeholder involvement in the process of prioritization. Genetic algorithms were applied for solving the problem.

Ngo-The & Ruhe [12] investigate the operational feasibility of strategic release plans. Resources are taken from a pool of available resources. Each person in the pool has more than one task that they are capable of performing, and has a skill level (weight) associated with each task. Distribution of the weighted resources in the most efficient manner allows for redistribution of unused resources in one skill area to another. The approach in [12] is complementary to this paper. It explores better utilization of resources via multiple task assignment, while this paper focuses on better resource utilization via exploring minimum cost, minimum slack, cost balancing, and bottleneck analysis solutions.

The approach proposed in this paper goes beyond the ideas presented above by providing further insight from performing pro-active analysis of solutions based on some anticipated scenarios. For a given set of candidate solutions, each solution is evaluated against these scenarios. Technically, the decision support method called SCENARIO-RP consists of three main steps which are described in more detail in the following sections:

Step 1: Modeling and definition of scenarios

Step 2: Generation of qualified alternative solutions

Step 3: Evaluation of solution alternatives against pre-defined scenarios

3.2 Modeling and Definition of Scenarios

Problem modeling summarized all the effort to describe the problem in the format given in (1) – (5). The purpose of that is the application of optimization algorithms to formally determine best solutions (solving "the problem right"). Optimization techniques can be used

to find optimal solutions to release planning along one dominant dimension of the problem. This dominant dimension is typically the stakeholder satisfaction with the proposed plans. Once solutions are generated along this dominant dimension, other problem dimensions can be explored in the context of the optimal solution, or in a set of near-optimal solutions. While this exploration can be left completely to the human decision maker, a better answer is to offer automated decision support in terms of exploring different scenarios along the other dimensions.

A scenario basically involves "what if" or "what would happen" questions. Formally, it is any variation of one of the parameters of the original problem described by the 9-tuple <F,S,C,WP,SP,WAP,ξ,λ, CAP> where

F – set of features

S – set of stakeholders

C – set of coupling constraints

WP- set of weak precedence constraints

SP – set of strong precedence constraints

WAP – weighted average priority vector

ξ - vector of degree of importance of releases

λ - vector of stakeholder weights.

CAP – resource capacity vector

This paper focuses on scenarios related to changes of the capacity vector. For the sake of brevity, we consider three types of scenarios:

Scenario type 1:
Capacity expansion for bottleneck resources
Justification: What could happen if additional resources would be provided compared to the baseline situation?

Scenario type 2:
Capacity reduction for bottleneck resources
Justification: What could happen if resources would be removed compared to the baseline situation?

Scenario type 3:
Cost-efficient expansion of capacities to achieve a pre-defined target level of stakeholder satisfaction.
Justification: What needs to be done to achieve a pre-defined level of stakeholder satisfaction compared to the baseline situation?

The definition of scenarios is not limited to the above content and can cover any variation of the above parameters. For all theses scenarios, we consider the given parameters as the baseline situation. The process of defining scenarios is illustrated in Section 4 for the hypothetical case study.

3.3 Generation of Qualified Alternative Solutions

Identification of qualified alternative solutions as studied in this paper is an extension and supplement of a hybrid intelligence approach called EVOVLE* introduced in [16]. Diversification of solution alternatives is a technique to facilitate solving the "right problem". This principle is part of research in decision support for the release planning problem in software engineering and has been implemented as part of an intelligent decision support system called ReleasePlanner [14]. This implementation consists of two phases:

- Phase 1: Pick up a set of α-qualified solutions $X' \subset X\alpha$ with M solutions contained in X'.
- Phase 2: Identify the set X* with m solutions having the maximum diversification.

In the current implementation, we have chosen α = 95%, M = 30 and m = 5 as default. The rationale of the selection of α and m is that for the uncertainty of the data we have, it makes little sense to require a higher accuracy. On the other hand, five alternative solutions is a reasonable size for final human decision-making. As for the selection of M we have observed that increasing M gives little improvement in terms of diversification but incurs a substantial increase in time.

For phase 1, we use the classic Branch and Bound algorithm [20] with a heuristic to identify a feasible solution at each node. Whenever an α-qualified solution is identified, it is added to the set X'. The process stops when the time limit is reached. If the optimal solution is reached before the time limit then the algorithm will backtrack [20] and continue the search of α-qualified solutions. If there are more than M solutions, then only the best ones are kept.

3.4 Evaluation of Solution Alternatives against Scenarios

To further qualify decision-making, we propose additional analysis based on supplementary criteria among a set of qualified solutions. The definition of scenarios may vary. In this paper, we have defined them with focus on resource consumption.

Definition 5: Bottleneck resources with respect to a given solution x0 are those resources whose capacity limits prevent one additional feature from being placed in a release.

Definition 6: The distance dist(x1,x2) between two release plans x1 and x2 is defined as the Euclidean distance between the two vectors, e.g.

$$\text{Dist}(x1,x2) = \left(\sum_{i=1..n} (x1(i) - x2(i))^2\right)^{1/2} \quad (7)$$

Among the set of candidate solutions, we are aiming at finding the one that accommodates at best possible future situations. For that, we determine the optimal solutions for the different scenarios. Assuming a set SCENARIO with

SCENARIO = {scenario1, scenario2,.., scenarioL}

of scenarios, we solve the respective instances of the release planning problem and solutions

$X_{scen} = \{x_{scen1}, x_{scen2}, \ldots, x_{scenL}\}$.

To determine the solution most likely best to accommodate future changes, we determine for each

qualified candidate solution $x \in X\alpha$ to sum Total_distance(x) of all the distances to the solutions from $X_{scen,}$ e.g.

$$\text{Total\_distance}(x') = \sum_{x \in X_{scen}} \text{Dist}(x,x') \quad (8)$$

Finally, the suggestion for making the release plan decision is x* satisfying

$$\text{Total\_distance}(x^*) = \text{Min}\{\text{Total\_distance}(x): x \in X\alpha\}$$

4 Hypothetical Case Study

4.1 Modeling and Scenario Definition

In what follows we will illustrate some key ideas of the proposed approach. For that purpose, we present a sample project with key project data summarized in Appendix 1, located on the web at: http://pages.cpsc.ucalgary.ca/%7Emcelroy/SEDS/downloads/appendix1_2_ResourceCentricRelease%20Planningv.pdf.

. The trial project encompasses 25 features to be planned for the next three releases. There are seven stakeholders involved.

According to the project manager, the relative importance of releases 1, 2 and 3 are given as $\xi(1) = 0.43$, $\xi(2) = 0.33$, and $\xi(3) = 0.24$, respectively. That means there is a focus on the importance of release 1, than release 2, and release 3 being of least importance.

There are eight constraint types involved in the realization of the features. Each of them has total capacity bounds for the three release periods. We observe that the total demand of all features exceeds the total capacity. This means that not all features can be implemented in the next three releases.

There is one coupling and one precedence constraint:

$\{(SYS\text{-}SW06,SYS\text{-}SW07\} \in C$ and
$\{(SYS\text{-}SW01,SYS\text{-}SW02\} \in WP$.

We define two scenarios for each of the three categories:

Scenario 1A: Determine bottleneck resources and expand all bottleneck capacities by 5% of their current level.

Scenario 1B: Determine bottleneck resources and expand all bottleneck capacities by 10% of their current level.

Scenario 2A: Determine bottleneck resources and reduce all bottleneck capacities by 5% of their current level.

Scenario 2B: Determine bottleneck resources and reduce all bottleneck capacities by 10% of their current level.

Scenario 3A: Determine a cost-minimal resource expansion to achieve a 5% increase in overall stakeholder satisfaction.

Scenario 3B: Determine a cost-minimal resource expansion to achieve a 10% increase in overall stakeholder satisfaction.

4.2 Qualified Alternative Solutions and their Evaluation against Scenarios

The set of 95% qualified alternative solutions is shown in Appendix 2, located on the web at the address given the previous section. This was obtained from running the optimization algorithms of ReleasePlanner. Appendix 2 also contains the solutions of the six scenarios as defined above. Table 1 gives the results from determining the distances (with equal weight for all the scenarios). We can see that solution 2 is most preferable because of having the minimum sum of all distances to accommodate future situations.

Table 1. Distances between qualified alternative and scenario solutions.

Scenario	ReleasePlanner Alternative Solutions					
		1	2	3	4	5
	1A	13	19	29	17	29
	1B	73	65	65	77	63
	2A	86	62	62	86	72
	2B	83	71	71	87	69
	3A	70	50	46	74	64
	3B	68	54	52	72	56
Total Distance		**393**	**321**	**325**	**413**	**353**

5 Summary and Conclusions

Uncertainty is an important issue in decision making. This is particularly true in the context of software engineering where uncertainty is one of its key characteristics. We argue that coping with uncertainty is essentially based on bringing in complementary knowledge and to appropriately handle this knowledge during the decision making process.

In this paper, we have presented a resource-centric decision support approach aimed at supporting human decision making in planning of software releases. The proposed method SCENARIO-RP is a three staged process. It initially generates a set of qualified alternative solutions. Based on that, different customized scenarios are defined to evaluate the candidate solutions against.

The importance of the paper is three-fold:

(i) Further support in determining a solution that is more likely to solve the "right problem" by exploiting diversification and performing pro-active scenarios.

(ii) The proposed solution is likely to be more robust against future changes in resource capacities.

(iii) The method is not limited to both resource-centric scenarios and planning software releases. Instead, it can be applied in decision support to other questions as well, e.g., selecting most appropriate designs or COTS.

Further research is necessary to provide tool support to define scenarios form a user perspective. In addition, empirical evaluation of SCENARIO-RP and the acceptance of the proposed solution(s) are needed.

Acknowledgment

The authors would like to thank the Alberta Informatics Circle of Research Excellence (iCORE) for its financial support of this research. Special thanks to Kornelia Streb for technical support in running the case study.

References

[1] Aurum, A., Wohlin, C., "The Fundamental Nature of Requirement Engineering Activities as a Decision Making Process", Information and Software Technology, Vol. 45, No. 14, 2003, 945-954.

[2] Bagnall, A. J., Rayward-Smith, V. J., and Whittley, I. M., "The Next Release Problem", Information and Software Technology, Vol. 43 (2001), 883-890.

[3] Beck, K., "Extreme Programming Explained", Addison Wesley, 2001.

[4] Bhawnani, P., Ruhe, G., "Kudorfer, F., and Meyer, L. Intelligent Decision Support for Road Mapping - A Technology Transfer Case Study with Siemens Corporate Technology", Accepted for Workshop on Technology Transfer in Software Engineering, ICSE 2006.

[5] Carlshamre, P., "Release Planning in Market-driven Software Product Development - Provoking an Understanding", Journal of Requirements Engineering, Vol. 7, No. 3, 2002, 139-151.

[6] Carlsson, C. and Turban, E., "Decision Support Systems: Directions for the Next Decade". Decision Support Systems Vol. 33, 2002, 105-110.

[7] Du, G., McElroy, J., and Ruhe, G., "Ad hoc versus Systematic Planning of Software Releases - A Three-Staged Experiment", Proceedings of 7th International Conference on Product Focused Software Process Improvement (PROFES06), Amsterdam 2006.

[8] Greer, D., G. Ruhe, "Software Release Planning: An Evolutionary and Iterative Approach", Information and Software Technology Vol. 46 (2004), 243-253.

[9] Jung, H-W., "Optimizing Value and Cost in Requirements Analysis", IEEE Software, Vol. 15 (1998), No 4, 74-78.

[10] Karlsson, J., and Ryan, K., "A Cost-Value Approach for Prioritizing Requirements", IEEE Software, Vol. 14(1997), No. 5, 67-74.

[11] Lethola, L., Kauppinen, M., and Kujala, S., "Requirements Prioritization Challenges in Practice", Proceedings of 4th International Conference on Product Focused Software Process Improvement (PROFES'04), Lecture Notes on Computer Science, Vol. 3009, 2004. 497-508.

[12] A. Ngo-The, G. Ruhe, "Optimized Resource Allocation for Incremental Software Development", submitted to IEEE Transactions on Software Engineering. 2006

[13] Penny, D. A., "An Estimation-Based Management Framework for Enhancive Maintenance in Commercial Software Products", Proceedings of International Conference on Software Maintenance (ICSM'02), Montreal, Canada, 2002, 122-130.

[14] www.releaseplanner.com

[15] Ruhe, G., and Ngo-The, A., "Hybrid Intelligence in Software Release Planning", International Journal of Hybrid Intelligent Systems, Vol. 1, No. 2, 2004, 99-110.

[16] Ruhe, G., and Saliu, O., "The Art and Science of Software Release Planning", IEEE Software, Vol. 22 (2005), No. 6, 47-53.

[17] Saaty, TL, "The Analytic Hierarchy Process", McGraw-Hill, New York, 1980.

[18] van-Gurp, J., Bosch, J, and Svahnberg, M, "Managing Variability in Software Product Lines", Proceedings of LAC'2000, Amsterdam.

[19] Wiegers, K., "Software Requirements", Microsoft Press, Redmont, 1999.

[20] Wolsey, LA and Nemhauser, GL, "Integer and Combinatorial Optimization", New York, John Wiley 1998.

Managing Uncertainty in Agile Release Planning

K. McDaid[1], D. Greer[2], F. Keenan[1], P. Prior[1], P. Taylor[2], G. Coleman[1]
1.Dundalk Institute of Technology, Dundalk, Co Louth, Ireland
{kevin.mcdaid, frank.keenan, paul.prior}@dkit.ie
2. Queens University Belfast, Belfast, BT7 1NN
{des.greer, p.taylor}@qub.ac.uk

Abstract

A clear and realistic release plan is central to the strategic planning activities of the firm developing the software. This paper supports existing agile methods by developing a novel but relatively simple statistical methodology to predict the time to develop selected functionality. In so doing it provides the product owner with a decision support mechanism to determine the likelihood of completing releases on time for any combination of stories. In this way it is consistent with the best Extreme Programming (XP) practice of selecting stories of two types for a quarterly release, ones that are key and must be delivered and ones that are considered as "Slack" and that can be developed if time permits. A case study is used to explain the proposed methodology.

1. Introduction

Release planning is the process of deciding when in the future a releasable software product should be available and what functionality should be included in each released version of the software product. It refers to the creation of a high-level plan that covers a period of typically up to three or four releases into the future. Crucially, it guides the strategic planning activities of the organization. Product companies producing regular releases to satisfy key customers are particularly reliant on release planning,

One approach to release planning is to fix a set of release dates and then determine how much functionality can be achieved by those dates. Alternatively, one could start by choosing the functionality and then derive the release dates. Irrespective of the approach taken, the software organization must assess the value of the functionality against the cost and time to develop the system. Unfortunately, these can be highly uncertain characteristics, and any framework must acknowledge this truth.

Software development projects, including those utilizing agile methods, have a clear need for reliable release plans. Projects with contracts and those that must meet a firm deadline and include a reasonably firm set of functionality are examples that come with greater consequences of being wrong. Similarly, projects that are planned far in advance or ones for which requirements are not well understood involve a significant amount of uncertainty which should be allowed for so that missed deadlines that will impact on a company's reputation are to be avoided. Small and newly established companies are acutely aware of the need to protect their reputations by delivering the functionality promised by the dates indicated. These same companies are the keenest proponents of agile development methods such as Extreme Programming [2]and SCRUM [9]

Building on current planning practices in agile methods this paper proposes a new mechanism to improve the release planning for such companies by allowing for the uncertainty in the time to complete the development work specified in the release plan. The methodology proposed is novel in that it assumes a two stage structure, the first allows for the uncertainty in the size of tasks and the second for the uncertainty in the project velocity. The work is demonstrated through a simple case study.

The paper is structured as follows. Section 2 introduces agile methods and the current approaches to release planning. Section 3 explains the background and the rational for the proposed methodology. A case study in Section 4 applies the methodology to a real problem and assesses its usefulness to the small company under examination. Section 5 concludes the paper and includes a short discussion as to how the business value of the functionality can be included in the methodology.

2. Release Planning For Agile Methods

Agile supporters have tackled the problem of release planning. In Extreme Programming this was termed the "Planning Game" [2]. This activity refers to the decision process as to which user stories to include in the next and

future releases, where 'user stories' are a high level description of user requirements. A release is normally developed over a series of iterations, typically of one or two week duration, with feedback obtained from the product owner at the end of some or all of the iterations. It is important to note that release planning does not include detailed iteration planning that would indicate, amongst other things, which developers will work on which stories or plans.

The planning game starts with the listing of all potential stories identified by both the product owner and the development team. For a bespoke project the product owner will naturally be the customer. However, in the case of product development, the product owner can be a member of the sales or management team who has expert knowledge of the market and the needs of current and potential customers. In the case of small companies, as in our case study, the product owner is more likely than not to be the chief executive officer.

Once the stories have been identified the size of each story is estimated by the development team. This size can be expressed in terms of ideal development days or in terms of story points. In this paper we concentrate on ideal development days, also known as perfect engineering days. It gives the number of days it would take a developer to complete the story assuming that they are in a position to engage solely in development activity. To indicate the amount of development work possible in an iteration, the team provide the product owner with the team velocity. This is often based on previous projects and, in effect, gives the number of ideal development days that the team will manage on average over the period of an iteration. Similar planning approaches are used in other agile methods with extensive guidance given in [2] and [3].

Based on this information the product owner must then select user stories for each release and the appropriate release dates. These key decisions involve the prioritization of features for inclusion in releases and iterations. This, in effect, requires the customer or product owner, customer, to perform a sophisticated cost benefit analysis to decide the viability of features within release and budget constraints.

One simplistic approach is to assign a business value, or priority, to each story and to select stories to maximize value over time. Typically highest value cards are selected earliest and if the release date is set up front, cards are added so as to deliver maximum value by then. Business value is established from business deliverables. Ideally, the product owner and the sponsoring organization would negotiate this in the presence of the customer.

Whatever approach is used, it is important that an agile method provides the customer with estimates for the time to complete releases that can be achieved with a very high level of certainty. It is also important, particularly for the incremental development methods key to agile processes, that these predictions can be updated as more information becomes available. Such a method is presented in this paper.

Other relevant approaches include those on requirements prioritization, reviewed in [5]. The methods could be adjusted to suit IID or even agile methods, but the effort required becomes excessive as the number of requirements rises. Several methods have been specifically built for an IID approach, such as those reviewed in [8]. One such method is EVOLVE [4],[7] which takes into account priorities from stakeholders, dependencies between requirements and effort limits for increments. On the face of it this method is quite suitable for agile planning, but requires coordination of stakeholders in prioritizing requirements and a trust in that judgement. Further, the effort estimation is based on requirements and not explicitly on the constituent tasks that deliver those requirements. Given that user stories are a common vehicle for representing user requirements in agile approaches and that these tend to be at a high level, estimation is more suitable after the tasks involved have been identified.

3. An Improved Methodology for Managing Uncertainty

The need for honest release plans within agile methods that acknowledge the actual time it takes to complete tasks, and the inherent variation in these estimates, is well-understood. Without these, the developing team has the tendency to over-commit and consequently under-deliver. These broken promises lead to reduced morale and to distrust between developers and the product owner. These considerations are to the forefront of the minds of today's customers chastened by accounts of cost and time overruns in the development of software systems.

This is highlighted in the latest set of key practices for Extreme Programming [10] which includes the practice, termed "Slack", of only signing up to for what the team is confident of achieving. Within this approach it is always possible to add more stories, time permitting, thus delivering more than was actually promised. This practice acknowledges that there is a significant amount of uncertainty in the estimated time to complete releases. Naturally the question arises as to how much functionality should be scheduled for each release.

Recent attempts to address schedule risk management for projects with a high level of uncertainty are based on the use of schedule and feature buffers [3]. Feature buffering involves the identification of "must have" user stories, representing up to 70% of the planned effort, which are given priority in the release. Other stories are only developed once these priority ones have been completed.

Alternatively, and more commonly, schedule buffers are used. First, it is necessary to quantify the uncertainty through the assignment of a duration range for a user story rather than a single estimate. This, as outlined in [3], can be achieved through the specification of 50% and 90% time points which represent the likely and pessimistic time for completion of user stories. This yields an estimate of the likely variation in the time. If a normal distribution is assumed, than an overall measure of variation for combinations of user stories can be easily calculated. User stories are selected provided the likely pessimistic time for completion of the selected combination of stories is within the proposed timeframe for the release. The sizing of a buffer in this way has been previously suggested in [12], [13] & [14].

We propose an alternative, but related, methodology to deal with uncertainty in release planning for agile methods. The novel approach in our view better reflects the sources of uncertainty and can thus yields more accurate and useful results. When combined with a usable tool, the methodology has the capacity to empower the product manager to explore many combinations of stories. In so doing the research supports the product manager in his/her key role in the selection of requirements for future releases. Furthermore the method can be extended to aid selection of the assignment of stories to iterations within releases.

The methods starts with the current practice for estimating the real or calendar time it will take a development team to complete a story or set of stories. As explained earlier this requires, for a single story, two distinct estimates as follows:

1. Estimation of the "size" of a story as represented by ideal days to develop the story in question.
2. Estimation of the likely project velocity represented by the ideal development days that will be completed by the team over an iteration.

These are combined to find the number of iterations required to complete the story.

Naturally, both of these estimates are subject to uncertainty. There is a variety of factors that can affect the level of this uncertainty. In the case of estimates of story size, some of the factors at play are the complexity of the story, the level of understanding of the requirement and experience of the development team. Similarly, project velocity can change from iteration to iteration during which differing amounts of developers time is spent on activities other than development such as team meetings or communication with customers and managers. This uncertainty can be even more pronounced in the case of small software product companies who may have to drop all development work during an iteration to concentrate on sales activities or on maintenance and support actions for key existing customers.

To date work [3], as outlined earlier in this section, has focused only on the uncertainty associated with estimation of story size. Building on existing work we propose to both improve the methods used to represent uncertainty in story size and to propose a mechanism to represent the uncertainty in project velocity over an iteration. These building blocks can then be combined to yield overall uncertainty in the time to develop individual or selected combinations of stories. In this way a set of stories can be chosen for future releases with the confidence that there is a minimal chance that they will not deliver on their commitments.

The detailed description and associated research for the methodology, we feel, is best presented through a case study conducted with a real company which we label "Company Z". This is presented in Section 4. Section 5 suggests how the method can be expanded to allow for the uncertainty in the business value of stories.

4. Case Study

Company Z (so-called for anonymity) is a small software firm that develops a single very high value product for a small number of very large organizations. They operate on the basis of quarterly releases of their main product. Typically a release would include new functionality driven by the needs of key customers and new functionality designed to attract new customers. They wish to be able to plan two or three releases in the future, selecting from a wide range of possible functionality for their innovative product.

As a relatively young product company with a small number of key important customers they are acutely aware of the need to deliver releases on time with the promised functionality. Of course, as a small company they have to do this within a tightly restricted budget. This calls for reliable release planning.

This goal is complicated by the uncertainty they face in the planning of projects. They use an agile development process but find that, while they have a good understanding of the functionality that could be added to the project, they have in the past struggled to provide accurate estimates of the size of stories. This has led to time and cost overruns. These overruns have been exacerbated by the need to perform ongoing maintenance and repair work driven by the needs of their key customers, a practice that impacts on the amount of time that can be spent during iterations on new development.

For these reasons the company need a reliable release planning mechanism that allows for the uncertainty in both the estimation of story size and in the estimation of project velocity given by the amount of development conducted during iterations. Furthermore, to reflect the highly complex decision problem as to which functionality to select, management desires a methodology that allows them to experiment with various

combinations of stories across releases. In particular the method should present accurate estimates of the time to complete each release.

Presently the method does not automate the optimal selection of stories and release times based on benefit values for the competing stories. This is a very complicated question that requires the manager to collate and compare a large number of disparate and uncertain factors. We return to this issue in Section 5 and for the moment focus on providing useful estimates for the release time given a selection of user stories.

We begin by assuming that the product owner has identified a number of stories for inclusion in upcoming releases. Agile methods would naturally require the development team to produce single value estimates of the story size in story units or in ideal days. These would be developed by first splitting the story in a small number of sub-stories or tasks and aggregating the estimates for each of these.

We propose a similar approach with the proviso that the level of granularity is chosen to allow the team to isolate activities that overlap between stories. However, to allow for uncertainty in the size of stories, the development team must provide two estimates, a most likely and a pessimistic value, in ideal days for the size of each sub-story. It is assumed (after discussion with domain experts) that there is only a 10% chance that the actual size of the sub-story exceeds the specified pessimistic value. It is also assumed that the chance that the story size is more than the most likely time is the same as the chance it is less than that value.

Table 1: List of Stories and Tasks

Story	Tasks
1	1, 2, 3, 4, 5, 6, 7, 8.
2	1, 2, 3, 4, 7, 8, 9
3	2, 7, 10, 11, 12.
4	2, 3, 5, 7, 12, 13.
5	14, 15, 16, 17, 18, 19, 20
6	21, 22, 23, 24, 25, 26, 27, 28

The stories and constituent sub-stories (called tasks) selected by Company Z are shown in Table with most likely and pessimistic values for sub-stories given in Table 2. The common tasks are identified through their numbers. Details of stories are not given for confidentiality reasons. Note that details of 6 of the 10 stories are shown. These involve 28 tasks with significant overlap between stories one, two, three and four.

We use a lognormal distribution to model the variation possible in the real size of a task or sub-story using ideal development days as the measure. This is based on work in [15] which shows that the actual time taken to develop a piece of functionality relative to the estimated time follows a lognormal distribution. The analysis is based on extensive data some of which is provided in [17]. The parameters of the distribution for each task can, through standard statistical methods, be estimated from the most likely and pessimistic values. For example the lognormal curve for task 6 is shown in Figure 1 with the height of the curve at a time point representing the likelihood that the task will take this length of time to complete in ideal development days. A lognormal distribution is chosen as it reflects the tendency of a development teams to significantly underestimate the size of stories. This is in contrast with the symmetrical shape of the normal distribution often used to represent ideal completion times. Alternatives are the beta and triangular distributions but these require greater amounts of data and assume an upper bound on the completion time which is difficult to specify.

Table 2: **Size of tasks in ideal development days**

Task	Most Likely	Pessi-mistic		Task	Most Likely	Pessi-mistic
1	0.5	0.75		15	3	6
2	1.5	2		16	2	4
3	1	1.25		17	3	4
4	20	30		18	0.5	1
5	0.25	0.3		19	2	3
6	6	10		20	2	4
7	2	2.25		21	1.5	1.75
8	3	5		22	2	2.25
9	5	6		23	1	3
10	1.5	2		24	1.25	2
11	5	6		25	6	12
12	3	5		26	3	5
13	5	6		27	5	6
14	1	1.25		28	3	5

Once the two estimates are provided for each task, lognormal parameter values for each task are then determined that specify the probability distribution for the size of the task. The distribution for the size, in terms of ideal development days, of a release based on the selected stories is then found by simulating from each of the task distributions that make up the selected stories. These sizes are aggregated and the set of aggregated times is used to give the distribution for the overall size. In this way the uncertainty in the size estimates for the individual tasks generates the uncertainty in the overall size of the functionality selected for a release.

To illustrate through an example, assume that Company Z wish to develop Stories 1, 2 and 5 during the next release. This would involve the completion of tasks 1 to 9 and 14 to 20. We simulate from the lognormal distributions for these tasks and add the results. This gives one possible size in ideal days for the release.

Repeating this process a large number of times, 10,000 say, results in a distribution for the size of the release.

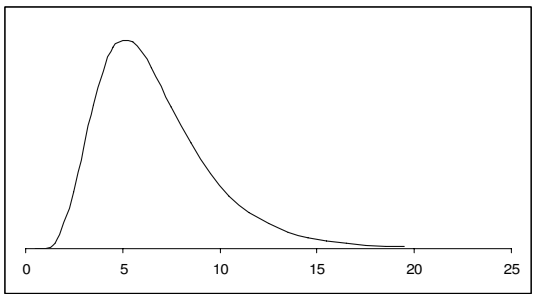


Figure 1: Lognormal distribution for size of task 6

This distribution for stories 1, 2 and 5 is shown in Figure 2. Although the average size of the release is 55 ideal development days the curve shows that there is an appreciable chance that the size could exceed 80 days. The curve indicates that there is almost no chance that the size will be less than 30 days or more than 90 days.

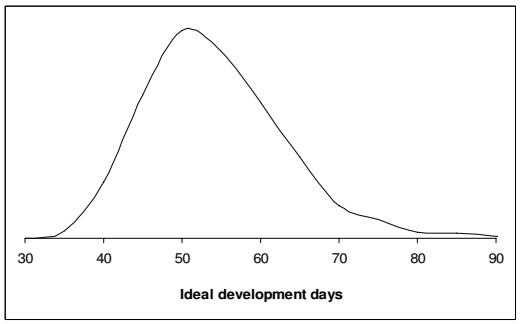


Figure 2: Distribution of Ideal time to complete stories 1, 2 and 5

Given the size of the combined stories in ideal days, it is still unclear as to how many calendar days or weeks it will take to develop this functionality. Agile methods calculate the likely duration of the release by dividing the size of the release by the project velocity, expressed as the number of ideal development days work completed in an iteration [3].

There is natural variation from iteration to iteration in the amount of time the team spends on development work. This is particularly true for Company Z where the small team is often called to deal with urgent maintenance work for existing releases. This leads to significant uncertainty in the project velocity.

In contrast to other research, the methodology proposed here allows for this variation in the project velocity. In the absence of supporting research, we choose the well-understood normal distribution to reflect the likely variation in the project velocity vary from one iteration to the next. The parameters of this model, namely the mean and standard deviation, could be specified by the development team and management or calculated from previous project velocity values if these are available and considered to be representative of what may happen in the upcoming release.

We make the assumption that these project velocity values are independent in that one low or high value does not mean the following iteration will also result in a low or high value. It would not be overly difficult to include a step in the methodology to allow for this dependency, likely to be relatively weak. However the need to keep the method as lightweight as possible [1] overrides this consideration.

Company Z, employing 2 full time developers, has estimated that the project velocity mean is 6 ideal days per iteration with a standard deviation of 1 day where an iteration is a one week period. Taking this into account we can now produce a distribution for the likely duration for developing stories 1, 2 and 5 in the upcoming release. The distribution for the calendar time to develop these stories is shown in Figure 3 with a range from 20 to 80 days. The average time is found to be 46 days or 9.2 weeks or iterations.

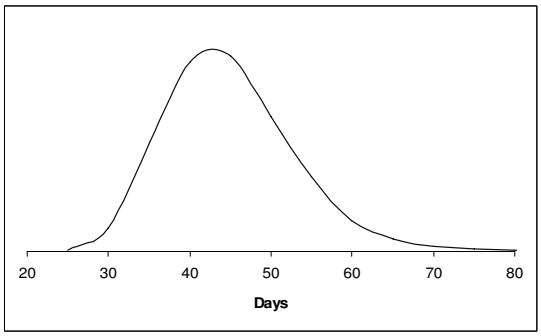


Figure 3: Calendar time to develop stories 1, 2 and 5.

Management need to know the likelihood that the release will be completed by the end of each iteration. Figure 4 shows, through a cumulative distribution, the likelihood that the release is completed by the end of each 5 working day iteration.

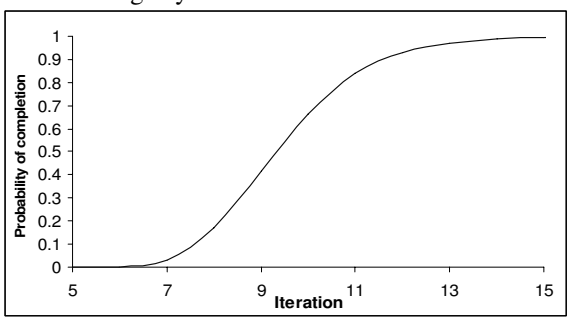


Figure 4: Likelihood that release is completed by the end of each iteration

Table 3 shows these values for the iterations from 8 to 13. While the average time to complete is 9.2 iterations, the table shows that, to be 90% certain of completing the desired functionality, management would have to plan a release date 12 iterations into the future. The additional three week period represents the slack

time that should be included in the plan to ensure that the release is completed on time. Of course, additional stories can be added as stories are completed on time. The question at to when the decision to add additional functionality can be taken is an interesting one currently under investigation.

Table 3:Likelihood of release completion by iteration

Iteration	8	9	10	11	12	13
Probability	.24	.47.	.71	.86	.95	.98

5. CONCLUSION

To this point the research explains a mechanism that allows customers or product owners to select combinations of stories and to choose resulting release times with the confidence that releases will be completed on time.

The important question remains however as to which stories should be selected. As outlined earlier this is a complicated decision based on the business value of the proposed stories with high value stories selected first. Usually, and certainly in the case of product companies, the business value of a story is a highly uncertain measurement, and any mechanism for selecting stories must acknowledge this uncertainty. Also, the selection of features should be based on a return on investment (ROI) measure allowing for the cost of development of stories.

The methodology presented in this paper can be extended as follows to allow for these added complications. First a distribution for the business value of each story would be developed, again through elicitation of an optimistic, expected and pessimistic value. Following this a distribution for the ROI of a story, incorporating the uncertainty, is found by simulating from both the business value and size distributions and dividing one by the other.

The authors of this paper propose a lightweight methodology for the planning of releases that reflects the main sources of uncertainty in the estimation of development time for companies following an agile process. The approach is illustrated through a case study.

The method is consistent with the Extreme Programming practice of selecting a conservative set of stories for development in a release to ensure that the firm delivers on their promises. This is achieved through scheduling of a slack period and this paper explains how this can be specified with confidence.

Although the case study looks at a single release, the approach can be applied to any number of releases into the future. Furthermore, the method can be reapplied, even at the end of each iteration, to allow for progress to that point. In this way a reliable release plan can be developed on an ongoing basis.

There are a number of assumptions with the model that may require further consideration. Specifically, the authors assume that any errors in the estimation of stories are independent. It is our belief that this may not be true as teams that understate the size of one story may also underestimate the size of others. While it is straightforward to deal with this complication in the model, the strength and nature of this correlation has not yet been established in the literature. Most importantly this correlation would lead to a higher level of uncertainty. There may also be a weak correlation between the project velocity values from one iteration to the next as support and maintenance activities started in one iteration can carry over to the next. This again would lead to greater uncertainty in overall estimates and could be modelled using a time series approach. That said, the importance of keeping the process as lightweight as possible is well-understood.

REFERENCES

[1] Agile Manifesto, www.agilemanifesto.org.
[2] Beck, K., Extreme Programming Explained, Addison Wesley, Reading, MA, 2001.
[3] Cohn, M., Agile Estimating and Planning, Prentice Hall, 2004.
[4] Greer, D. & Ruhe, G., Software Release Planning: An Evolutionary and Iterative Approach, J. Information and Software Technology, vol. 46, issue 4, pp 243-253, 2004.
[5] Karlsson, J, Wohlin, C and Regnell, B., An evaluation of methods for prioritizing Software Requirements, Information and Software Technology, 39 (1998), pp 939-947.
[6] Larman, C., Basili, V. R.: Iterative and Incremental Development: A Brief History. IEEE Computer, Vol. 36(6), pp. 47 – 56, 2003.
[7] Ruhe, G. & Greer, D. Quantitative Studies in Software Release Planning under Risk and Resource Constraints, Proceedings of the IEEE-ACM International Symposium on Empirical Software Engineering, 262-271, 2003.
[8] Saliu, O., Ruhe, G., Software Release Planning for Evolving Systems. Innovations in Systems and Software Engineering, Vol 1 (2005), Issue 2, pp 189-204.
[9] Schwaber, K. and Beedle, M, Agile Software Development, Prentice-Hall, 2002..
[10] Beck, K. & Andres, C., Extreme Programming Explained (2ed), Addison Wesley, Reading, MA, 2004.
[11] DeMarco, T., Slack: Getting Past Burnout, Busywork, and the Myth of Total Efficiency, Broadway Books, New York, NY, 2001.
[12] Reinertsen, D.G., Managing the Design Factory: A Product Developer's Toolkit. Free Press, 1997.
[13] Newbold, R.C., Project Management in the Fast Lane: Applying the Theory of Constraints, St. Lucie Press, 1998.
[14] Leach, L.P., Critical Chain Project Management, Artech House, 2000.
[15] Little, T., Agility, Uncertainty and Software Project Estimation, http://www.agilealliance.com
[16] /articles/littletoddagilityunce, accessed 1 March 2006.
[17] DeMarco, T., Controlling Software Projects, Prentice-Hall, 1982

A decision modelling approach for analysing requirements configuration trade-offs in time-constrained Web Application Development

Sven Ziemer[1], Pedro R. Falcone Sampaio[2] and Tor Stålhane[1]

[1] Department of Computer and Information Science,
Norwegian University of Science and Technology, N-7491 Trondheim, Norway
[2] School of Informatics, University of Manchester
PO Box 88, Manchester M60 1QD, UK
E-mail: {svenz|stalhane}@idi.ntnu.no, Pedro.Sampaio@manchester.ac.uk

Abstract

Release planning is a key phase in a time-constrained web development process, enabling stakeholders to find an appropriate trade-off between time-to-market and quality/functional characteristics of the delivered artifact. Despite the importance of conducting release planning activities, many web development projects still employ ad-hoc methods for deciding on the next release to be developed. In this paper, we discuss an effective release planning and configuration method underpinned by a decision modelling framework aimed at supporting small development teams to analyze, prioritize requirements, and to find a candidate release configuration that can be developed within the time, quality and functionality constraints relating to the project. The method is value-based, taking into account the stakeholders' belief in what is regarded as the most valuable release configuration. The method is also consensus-driven, seeking to promote a consensual agreement with regard to the next release to be implemented.

1 Introduction

When developing software systems it is important to meet the stakeholders' needs and expectations. Usually, there is a considerable investment at stake and the stakeholders are keen to maximise their return on investment. Many stakeholders also want to realize part of the total value at stake in connection with the software system in the early phases of the development project and not wait until the project is completed in order to benefit from the system properties. One way of meeting stakeholders expectations earlier is to deploy the system as successive functionality chunks, as it is done in incremental development. For every new chunk that is delivered, some part of the system's value will be returned to its stakeholders. In addition, early feedback can be collected and necessary changes can be implemented. Planning for the next release thus becomes an important and complex activity. The complexity stems from the task of finding the optimal release that can be developed within the time constraints and that represents the highest possible value in return for the stakeholders.

Web applications that are developed in competitive environments put a strong emphasis on timely development. Hence, it is important to find the "best possible" release – consisting of both new features and/or improved features – that satisfy customers and that also address competitive forces that may arise in connection with the application context, e.g., a release that covers all functionalities available at a competitor's web site.

Release planning is often done ad-hoc [8]. In interviews with Norwegian small and medium enterprises developing web applications we have found that development practises are informal, ad-hoc and "rush-to-market" [9]. This makes it difficult to develop a release plan strategy. One reason for this is the lack of information and time to identify and analyse candidate releases, or even to have a detailed analysis of the requirements that should be contemplated at each stage. As these companies are depending on the domain knowledge of their developers and other stakeholders, the discussions between stakeholders are based on their experience and opinions. In many cases, these opinions can be conflicting as each stakeholder has his own interest and responsibility. In order to make important decisions, like the next release(s), it is important to find the right balance between several options advocated by the stakeholders.

The remainder of this paper discusses an effective release planning and configuration method underpinned by a decision modeling framework aimed at helping small development teams to analyze and prioritize requirements, and to find a candidate release configuration that can be devel-

oped within the time, quality and functionality constraints relating to the project. The method is value-based, taking into account the stakeholders' belief in what is regarded as the most valuable release configuration. The method is also consensus-driven, seeking to promote a consensual agreement with regard to the next release to be implemented. The paper is organised as follows: A description of the problem of finding the next release is given with some detail in section 2 and an example is presented in section 3. Section 4 looks at how stakeholders can express their beliefs about return on value using qualitative information. This approach is then used in section 5 to analyse, negotiate and decide about requirements and configurations. Related work is presented in section 6 and conclusions are given in section 7.

2 The problem

Developing software in a competitive environment puts a strong emphasis on time-to-market. There are usually more requirements to implement than time available for the next release. Thus, only a subset of the requirements can be implemented. Deciding on the content of releases is a non-trivial task. For a web application it is important to offer features that attract new users and that at the same time satisfies the existing ones. Still, deciding on the content of the next release is a decision that often is taken on the fly, without time for an in-depth analysis. Based on our experience, working with small and medium enterprises in the Websys project, we have observed that this decision is made either by the strongest stakeholder, who is likely to prioritise his own interests, or by the development team, excluding other stakeholders that might have a say in this decision. Two fundamental questions follow in connection to the issue of deciding on the content of the next release to be developed:

Who should be involved in this decision? There are several options about who to involve. The decision can be taken by a single stakeholder such as the project manager or the marketing director, by the project manager and the developers or by all major stakeholders. Involving all major stakeholders is the preferred approach in an environment where a considerable amount of information is handled informally, enabling a comprehensive assessment of all relevant variables to the decision and also sharing the risks linked to the consequences of the decision.

What should this decision be based on? In the literature we find a multitude of approaches and variables used for deciding on what will be contemplated in the next release, such as customer feedback, return of investment, stakeholder weights and strategic urgency. Another important approach, which is discussed in this paper, focuses on the stakeholders perceived return on value associated with a release. This is a criterion every stakeholder can relate to, as it expresses his expected gain from a release.

ID	Description and Interests
S1	Product director. Wants a stable website, that runs smoothly and that has a positive impact on increasing the sales.
S2	Marketing director. Wants a website with a high "newness" and innovation factor and a short time-to-market. Wants to announce special prices on the website every week.
S3	IT manager. Wants a solution that is easy to maintain and extend. Prefers stable versions of tools and technologies that are to be used.
S4	Provider of payment transactions. Wants visibility and reliable services.
S5	Customers. Wants easy access to the online shop, safe payment transactions, reliable transport (shipment) and access to product specs.

Figure 1. The stakeholders to the online shop

The problem to solve then is how to find the most favourable release for all stakeholders. The stakeholders will assess each candidate release and express their perceived return on value for it. This is based on each stakeholders subjective belief. The goal is to find a consensus and to balance the interests from the stakeholders. The following design principles guide the approach discussed in this paper: It must be simple so that it can be understood intuitively by all stakeholders, it must take into account all perspectives and stakeholder information available (which in most cases are the stakeholders opinions and beliefs), and it must be effective in converging to a consensual decision with regard to the next release. The approach proposed is a decision making framework based on the return on value of candidate releases in which a strong emphasis is placed in promoting a consensual agreement among stakeholders.

3 Example – An E-commerce system

In this section we discuss an example of a small electronic shop, based on one of the Norwegian enterprises involved in the Websys project. The small shop for electronic equipment launched an "information only" website in 1999. It allows customers to browse a product catalogue and to find information and prices on products. Lately, the company faced competition from two online retailers and from a local shop that started an online shop. The sales report for the last year shows that the shop is loosing customers to the online shops. To respond to this challenge from the competitors the shop decides to start an online shop itself. The goal of starting a new sales channel is to take back customers from the competing online shops, and also to win over new customers.

In the process of starting the online shop one had to iden-

Req	Description
R1	Improve Browse functionality, add product specs for every entry.
R2	Add product to cart.
R3	Place order. Requires that a user logs in. Edit contents of shopping cart, place orders.
R4	Register user, establish user profile. Profile contains name, username, password, and shipment and payment details.
R5	Change user profile.
R6	Cancel order, if order still is pending.
R7	Specify a wish list.
R8	Browse past orders.
R9	Track shipment.
R10	Search for a product.

Req	Description
R11	Browse through a list of special offers.
R12	Personalise product list. Show the user products he may be interested in, based on previous sales.
R13	Maintain catalogue. Add, change or delete items from the systems catalogue.
R14	Browse orders. For every order, check if the products are available.
R15	Manage orders. Change orders, split orders, etc.
R16	General design makeover.
Q1	Performance
Q2	Reliability
Q3	Security
Q4	Usability

Figure 2. Requirements

tify the stakeholders that should have an influence over the contents of the new website, and to make some organisational changes. The development work was done by a third party development company. A manager for the online shop had to be appointed, who was responsible for all contact with the development company. A new position that is responsible for all administration work for the online customers had to be created. An important decision was the selection of a service provider for internet payment transactions. The number of stakeholders involved in this project, increased substantially compared to the first system. A detailed description of the stakeholders is given in figure 1.

The stakeholders arranged a workshop both to identify a strategy for the online shop and to elicit the requirements. The requirements identified are shown in figure 2.

A development scenario – The marketing director is requesting that the new version of the company website has to be deployed within six weeks. In his view this is crucial in order to provide an online offer that exceeds all features provided by the competitors. The competitors are also in the process of changing their online sites, therefore adhering to the six week time-to-market plan is essential. The first estimate from the development team concludes that it will take at least 12 weeks to implement all the new requirements. The project manager is asked to produce a list of what can be developed within six weeks. He comes up with three candidate configurations, that each can be implemented within six weeks. In order to have an online shop requirements R2 – R6 are mandatory. The configurations are shown in figure 5 and the details on how they where identified are given in section 5.

4 Value assessment

In the example presented in section 3, only a subset of the requirements specification can be implemented within the given time-constraints. We call such a subset a *configuration*. A configuration consists of a number of functional and non-functional requirements that collectively have a meaningful impact on advancing the implementation of the overall business goals (e.g., a configuration that implements a new shopping cart) and that can be developed within the given time constraints. Each candidate configuration represents a potential software release. There is more than one candidate configuration in our example and a decision has to be made on which configuration to implement. As mentioned earlier, in a situation where a considerable amount of information is not available in writing but only as tacit and informal knowledge, this decision should be taken by all the stakeholders together to include all relevant information. This information would typically express the stakeholders' beliefs and opinion about value, cost, benefit, risk, assessment of competition, etc. We will in this paper focus on the return on value.

One way of expressing this tacit information on the fly is the use of attitude measurement. One widely used approach is the summated rating or Likert scale [7]. We use a similar approach, where each stakeholder assess the perceived return on value of a requirement using a single scale from 1 to 5: 1– no value, 2 – little value, 3 – some value, 4 – high value, and 5 – very high value.

We want to clarify some assumptions before describing our approach. The assessment represents the perceived return on value for each stakeholder and is subjective in its na-

ture. We assume, however, that the stakeholders understand their business and the development project. The projects where the approach is applicable are typically conducted by teams of two to five developers and have tight time-constraints (typically between two and 24 weeks). Therefore, an assessment has to be based on every stakeholders judgement. The expressed belief represents each stakeholders expert judgement, which is based on their best information, knowledge and intentions. We recognise that any stakeholder who wants to misinform the other stakeholders to influence the decision making process in his favour could easily do so, but then, this would also be possible when basing the decision on ”hard facts”.

5 Analysing, negotiating and deciding about configurations

In this section we present the overall release decision process, starting with requirement elicitation and resulting in a consensus based decision on which configuration to implement as the next release. This process encompasses five stages, which are described below and shown in figure 3.

Requirement elicitation – The requirements are elicited using a elicitation method or practise. In our example a workshop was arranged. The result from this stage is a list of requirements, including both functional and non-functional requirements (figure 2).

Requirement estimation and assessment – The next stage is to estimate and assess each requirement. This involves the project manager (for the first part) and all stakeholders (for the second part). The result of this activity is shown in figure 4.

For each requirement, the project manager is responsible for providing a time estimate, any dependencies on other requirements and an urgency flag, that, if set, indicates that a requirement has to be included into the next release. All stakeholders assess each requirement using the scale as shown in section 4. They also write a short justification explaining their choice for each requirement. This justification will be used during the Decision on a configuration stage in case no configuration emerges as a clear collective winner. Also, for each requirement the mean value of all stakeholder assessment is calculated.

For our example, the result of this stage is shown in figure 4. Note that the mean value is not shown. The total time to develop all requirements is 133 days. With two developers, the maximum effort possible within the time constraint is 60 days. Requirements R2 – R6 have to be included in every candidate configuration to get a meaningful release.

Identifying candidate configurations – Using the information from the previous stage, a number of candidate configurations are identified. There is no rule about how many candidate configurations should be identified, but as a rule

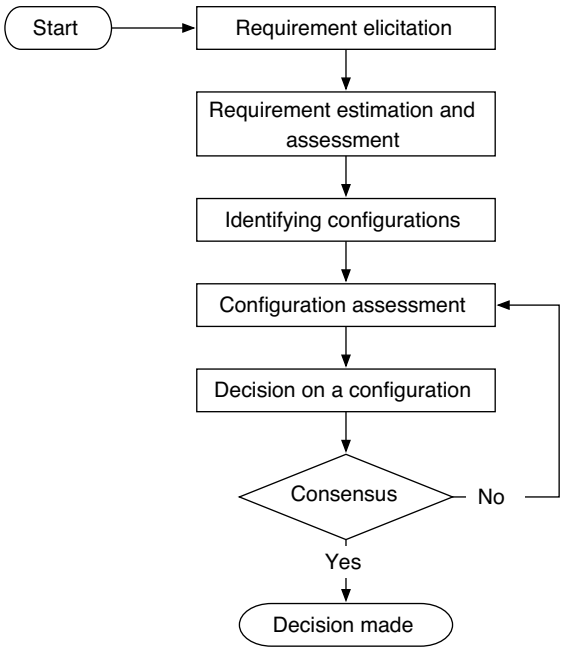


Figure 3. The configuration decision process

of thumb there should be 3 – 7 candidate configurations. To simplify the decision scope, there are some guidelines for identifying candidate configurations:

1. Sort the requirements according to some sorting criterias. For every criteria applied there will be a separate requirement list, which in turn can be used in the subsequent step, thus producing more than one candidate configuration. Common for all sorted requirement lists is that requirements with the urgency flag set have to come first. Two sorting criterias are (1) the mean value of all stakeholder assessments for a requirement, and (2) the number of high scores from the stakeholders.

2. Include into a configuration the requirements in the sorted order, until the estimated totals equals the specified time-constraint or until the addition of still another requirement will exceed this constraint.

3. For each requirement added, include also all requirements this requirement is depending on.

4. In case that not all requirements with the same sorting criteria can be included into a candidate configuration due to the time-constraint, additional candidate configuration can be identified by including requirements that have the same ranking and that are no yet included in any configuration for the given sorted requirement list.

Req	Urgency	Estimat	Dependencies	Stakeholder			
				S1	S2	S3	S5
R1		4 d	R10	4	3	3	5
R2	Yes	3 d		1	2	2	4
R3	Yes	10 d	R2	2	3	2	4
R4	Yes	8 d		2	2	3	4
R5	Yes	3 d	R4	1	3	3	3
R6	Yes	7 d	R3	3	2	2	3
R7		10 d	R2	3	4	2	4
R8		5 d	R3	2	3	3	3
R9		3 d	R3	3	2	4	4
R10		10 d		2	1	2	3
R11		4 d		3	5	2	2
R12		15 d	R3, R4	2	4	2	2
R13		2 d		3	2	4	1
R14		2 d	R3	4	2	3	1
R15		4 d	R3, R14	4	2	3	1
R16		20 d		3	5	3	4
Q1		7 d		4	3	4	3
Q2		3 d		3	3	4	3
Q3		8 d		3	3	5	5
Q4		5 d		3	4	4	4
		133 d					

Stakeholders: S1: Product manager, S2: Marketing director, S3: IT manager, S5: Customer

Figure 4. Return on value assessment

The three configurations identified in our example are shown in figure 5. The first two lines show the requirements included and the effort estimate.

Configuration assessment – When the candidate configurations are identified, they have to be assessed in the same way as the requirements where assessed. Again, stakeholders write a short justification explaining their choice for each configuration. Also, for each configuration the mean value of the stakeholders assessment is calculated. The configurations can be ranked by sorting them on the mean value. The configuration with the highest mean value is the one with the highest return on value according to the stakeholders assessment.

The result of this stage is shown in the two last lines of figure 5. Configuration 3 has the highest median value, followed by configurations 2 and 1. There should, however, not be any automatism in choosing the configuration with the highest rank as the next release. The information that leads to this rank is meant to support a discussion between the involved stakeholders in order to reach a consensus based decision.

Decision on a configuration – The purpose of this stage is to find a consensus based decision on which configuration should be implemented in the next release. The result of the configuration assessment can end with a clear cut configuration that emerges as the ”winner” or with two or more configurations ending up with the same ranking. In both cases, a consensus on which configuration to implement as the next release, is needed.

If there is a winner from the configuration assessment, it is straightforward to reach a consensus. There should be strong arguments for not choosing the winning configuration. When there is no consensus on which configuration to choose, the stakeholders should first present their written justification to gain more insight into the other stakeholders assessment and arrive at a consensus.

When no consensus can be reached at this stage, the configuration assessment step has to be repeated. A Delphi inspired process [5] will be used to guide this reconciliation. To arrive at a consensus concerning the consequences of the possible actions is the main problem in a trade-off situation. The whole decision process will break down if no agreement on a configuration that have an acceptable return on value for all stakeholders, can be reached.

The Delphi feed-back process is used to arrive at a common understanding. As shown in figure 3 this means that the stage Configuration assessment will be repeated. The process broadly runs as follows:

1. Each stakeholder assesses return on value for each configuration, and gives his results in writing. The coordinator sums up all the assessments – again in writing – and distributes them to all the stakeholders.

2. After having read the other stakeholders opinions, each stakeholder gives a new assessment. This assessment is accompanied by a rationale explaining why his assessment differs from that of one or more of the other stakeholders. This new assessment is given to the coordinator.

Step 2 is repeated until a consensus is reached.

6 Related Work

There are a number of methods supporting software releases decisions, such as [8], [4], [3], [6] and [2]. These methods differ both with respect to the stakeholders involved (all major stakeholder, project manager and developers, customers and users), the objectives (Return of Investment, value of requirements, customer satisfaction and benefit of features to customers) and the applied mechanisms (Optimization, Stakholder prioritiszing, AHP). The

Config	1		2		3	
Req´s:	R1 – R6, R10, Q3, Q4		R2 – R6, R7, R9, Q2		R1 – R6, R10, R11, Q3	
Estimat:	58 d		60 d		57 d	
Value:	S1: 3	S2: 2	S1: 3	S2: 3	S1: 3	S2: 4
	S3: 3	S5: 4	S3: 4	S5: 4	S3: 4	S5: 4
Mean:	3		3.5		3.75	

Figure 5. Assessment of configurations

approach presented in this paper involves all stakeholders, and is assessing candidate configurations rather than requirements, thereby reducing the complexity of the decision process.

Value-based software engineering acknowledge the fact that software systems are not developed in a value-neutral environment. A general theory for Value-Based Software Engineering is given in [1]. The approach presented in this paper uses qualitative measures to express the perceived return on value.

7 Conclusions and further work

This paper described a consensus-driven and value based release planning approach that can help small development teams to analyze, prioritize requirements, and decide on the next release in a scenario of time-constrained web application development. The approach is underpinned by a qualitative decision modelling framework that employs scales similar to Likert scales and Delphi techniques to identify release configurations and support stakeholders in the process of assessing, negotiating and choosing the next project release.

The approach was developed based on our experience in working with small and medium Norwegian enterprises in the Websys project. In the context of Websys, companies manifested their interest in a systematic (to avoid the blind ”rush to market”) and inclusive (to cater for the less hierarchical nature of SMEs) approach for deciding on system releases, taking into account the multiple stakeholder perspectives of value attached to system requirements. We are currently experimentally assessing the framework with the consortium of SMEs involved in Websys and preliminary results indicate that the approach has a positive impact in fostering consensus and increasing project commitment between stakeholders. The simplicity of the approach also helps in avoiding typical ”analysis paralysis” situations that arise in connection with complex decision models and/or highly democratic decision processes.

References

[1] S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Grünbacher, editors. *Value-Based Software Engineering*. Springer-Verlag Berlin Heidelberg, 2006.

[2] M. Denne and J. Cleland-Huang. The incremental funding method: Data-driven software development. *IEEE Software*, 43(14):39–47, 2004.

[3] H.-W. Jung. Optimizing value and cost in requirement analysis. *IEEE Software*, 15(4):74–78, 1998.

[4] J. Karlsson and K. Ryan. A cost-value approach for prioritizing requirements. *IEEE Software*, pages 67–74, September/October 1997 1997.

[5] H. Linstone and M. Turoff. *The Delphi Method - Techniques and Applications*. Addison Wesley Publishing Company, Inc., 1975.

[6] D. A. Penny. An estimation-based management framework for enhancive maintenacne i commercial software products. In *18th International Conference on Software Maintenance (ICSM 2002), Maintaining Distributed Heterogeneous Systems, 3-6 October 2002, Montreal, Quebec, Canada*, 2002.

[7] C. Robson. *Real World Research*. Blackwell Publishers, 2002.

[8] G. Ruhe and M. O. Saliu. The art and science of software release planning. *IEEE Software*, pages 47–53, November/December 2005 2005.

[9] S. Ziemer and T. Stålhane. Web application development and quality - observations from interviews with companies in norway. In *Proceedings of Webist 2006*, 2006.

Multi-Model Based Optimization for Stream Query Processing

Ying Liu and Beth Plale
Computer Science Department
Indiana University
{yingliu, plale}@cs.indiana.edu

Abstract

With recent explosive growth of sensors and instruments, large scale data-intensive and computation-intensive applications are emerging, especially in scientific fields. Helping scientists to efficiently, even in real time, process queries over those large scale scientific streams thus has great demand. However, query optimization for high volume stream applications– in particular its core component, the evaluation model– *has not been systematically studied. We observe that evaluating stream query plans should consider three aspects:* output rate, computation cost *and* memory consumption*. However, to our knowledge, no existing research on evaluating stream query plans consider all three metrics. In this paper, we propose a new combined optimization goal which leverages all these aspects and develop a multi-model based optimization framework to accomplish this goal. Specifically, we build three models to evaluate a plan's output rate, computation cost and memory consumption respectively. Based on such three models, we search for an optimal plan while considering systems's computation resource and memory constraints. We also experimentally evaluate our optimization framework.*

1 Introduction

Recently, technological advancements that have driven down the price of handhelds, cameras, phones, sensors, and other mobile devices, have benefited not only consumers but the computational science community. As a result, data-driven computing is emerging, where computationally intensive applications often need real-time responses to data streams from distributed locations. These streaming sources can have vastly varying generation rates and event sizes. Responsiveness, i.e., the ability of a data-driven application to respond in a timely manner is critical. For instance, out-of-date analysis of beam luminosity data can be useless or worse yet may mislead particle physicists. Therefore, in practice, performance plays an important role in stream query processing.

Stream query processing has been an active research area in recent years [1, 2, 8], yet limited work has been done on query optimization for such high volume stream processing. Especially, to our knowledge, the core part of stream query optimization, i.e., the *evaluation model*, has not been systematically studied in this context.

As infinite event sequences, data streams introduce new challenges to query plan selection. First, since cardinality is not available for streams, the cardinality-based cost model loses its usefulness here. Second, queries may not complete within designated service time because bursty streams or limited system resources.

In response to these challenges, we propose a multi-model based stream query optimization framework that considers three metrics: *output rate*, *computation cost* and *memory consumption*. Such a framework includes evaluation models to estimate three metrics respectively. As we will discuss in Section 3, our framework also considers a system's constraints on computation and memory resources. When a system's resource, either computation resource or memory resource, is insufficient to process a query within a designated service time, we reduce arrival rates by selectively dropping events. The selected plan is an optimal or sub-optimal one at the new input rates.

We observed that three metrics are relevant, but they are not dependent variables. For example, two equivalent plans with the same output rate may have different computation cost. As shown in Figure 1, planA and planB are equivalent plans where nodes $S1(S2)$, P and J represent operators of selection, projection and join respectively. The individual operator's selectivity (σ) and cost (c) to process an unit event or event pairs are marked beside the operator. The arrow lines denote intermediate streams whose rates are marked above the lines. Although these two plans have the same output rate r_o, planB's computation cost of 67480 is more than 28 times higher than planA's cost of 2400. The symbols used here are formally defined in Section 2, and the computation formulas are discussed in Section 3.2. Similar relationship exists in any pair of metrics. Due to the space limitation, we will not enumerate all of them.

Therefore, stream query optimization should consider

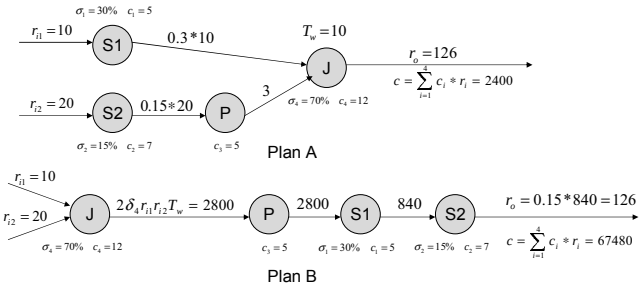


Figure 1. Plans with same output rate but different costs.

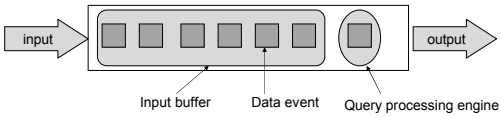


Figure 2. Stream query processing model.

the three metrics and have a combined optimization goal. In response to this, we propose a new optimization goal which considers all three metrics: output rate, computation cost and memory consumption. To accomplish such a goal, we conduct a multi-model based optimization framework, which has three evaluation models to estimate each individual metric. Further, we implement our optimization framework in Calder [20], a grid enabled stream processing system.

The remainder of this paper is organized as follows: In Section 2 we give preliminary definitions used in this paper. Section 3 introduces our multi-model based optimization framework which includes an optimization goal, a search algorithm and evaluation models. Section 4 reports experimental justification of the multi-model based stream query optimization. Section 5 discusses related work and Section 6, future research plans.

2 Preliminary Definition

In this section, we lay the groundwork with definitions that will be used in this paper.

2.1 Stream Query Processing Model

A stream S is defined as a sequence of events, $S = \{e_i\}$ where i is a monotonically increasing integer and $0 < i < \infty$. An *Event* is an atomic unit of data in the stream and the counterpart to a tuple of a traditional database table. In the stream processing system, queries are *continuous query* [6] which will remain in the engine for a period of time specified by the owner of the query and continuously evaluate incoming events. Similar to [10], our *Stream Query Processing Model*, shown in Figure 2, takes streams as input and produces streams. A *processing engine* holds continuous queries and processes stream data events sequentially.

Terminology	Symbol
Input Rate	r_i
Output Rate	r_o
Processing Rate	r_p
Cost for Unit Selection	c_{su}
Cost for Unit Projection	c_{pu}
Selection Cost per event	c_s
Projection Cost per event	c_p
Join Cost per event pair	c_j
Join Memory Consumption	m_j
Event Size	es

Table 1. Terminology definition.

2.2 Evaluation Model Terminology

Given two adjacent events e_j and e_{j+1} in a stream which arrive at time t_j and t_{j+1} separately, we define *Stream Rate* r as $r = 1/(t_{j+1} - t_j)$. Accordingly, r_i and r_o are the input or output *Stream Rate* of the query respectively. A query operator has a $ProcessingRate\, r_p$ which defines the number of events that can be processed by this operator per time unit.

As stream data pass through networks and most processing occurs within main memory, *computation cost* and *memory consumption* become two major consumed resources. In our evaluation model, computation cost, c, is defined as the amount of *CPU time* spent processing events in a *Unit time*. We use m_j to denote memory consumption for a $JOIN$ operation. To compute the memory consumption, we also define an event's size as es.

We further use c_s, c_p and c_j to represent the computation cost to process a data event by selection, projection and join operators respectively. As we will discuss in Section 3.2, these costs are based on the unit cost c_{su} and c_{pu}, where c_{su} is the unit cost to compare a condition and c_{pu} is the unit cost to copy an attribute.

Table 1 lists all the symbols defined in this section.

3 Multi-Model Based Optimization

3.1 Optimization Goal and Search Algorithm

As discussed in Section 1, our optimization framework has a combined goal as *Finding a plan with maximum output rate, minimum computation cost and minimum memory consumption, while considering a system's resources constraints.* In the following, we discuss how we select a plan satisfying such goal.

Resources consumed by stream processing include CPU cycles and memory resource. Considering such two resource constraints, there are four different scenarios shown in Table 2. In Scenario I, the system has sufficient resources to process all the input data. Therefore, as discussed in [14], the input events are fully processed and equivalent plans under such scenario have the same output rate. Scenarios II,

	Memory sufficient	Memory insufficient
CPU sufficient	Scenario I	Scenario II
CPU insufficient	Scenario III	Scenario IV

Table 2. Four scenarios with constraints.

III and IV all face resource constraints of one form or another. Under these scenarios, we reduce streams' input rates by selectively dropping input events to fit the system's resource constraints. This techniques can be referred as *sampling* [15], with which the output is not based on complete input set. However, as we drop events probabilistically, results are not biased.

Given a query, we can have many equivalent plans with different resource requirements. Hence, these equivalent plans may belong to different scenarios for a specific system. Plans under scenario II, III and IV are facing resource shortages. As discussed above, we reduce input streams' rates by sampling input events to move these plans to scenario I, where the system can provide adequate resource to the plan. Then, based on our optimization goal. We first search for plans with maximum output rate, as output rate indicates output completeness [14].

After we get a set of plans with the same maximum output rate, we intuitively want a plan with minimum computation cost and minimum memory consumption from the candidate set. However, in Section 1, we have pointed out that computation cost and memory consumption are independent variables. The ideal plan with minimum values on both metrics may not exist. For such dual-metric selection, we rely on users' preferences, which are specified when users submit queries.

Based on above analysis, our search strategy is summarized below. We first search for plans with maximum output rates. Then we do further selection based on users' preference. If users believe CPU resource is tighter than memory resource for their applications, we first find plans with minimum computation cost, then within the selected plans find the one with minimum memory consumption. Vice versa. Note, if sampling is needed due to resource constraints, all computation is based on decreased input rates caused by events dropping.

3.2 Evaluation Models

Besides the optimization goal and search algorithms, evaluation models are also indispensable for an optimization framework. An evaluation model defines a set of formula to estimate a plan's metrics. As we discussed in Section 1, our framework considers three metrics: output rate, computation cost and memory consumption, therefore we have an evaluation model for each of them.

3.2.1 Evaluation Model for Computation Cost

According to Table 1, c_s, c_p and c_j denote the cost of selection, projection and join operators respectively. In this section, we first show how we compute them based on unit selection and projection cost c_{su} and c_{pu}. We further discuss how we compute a query plan's cost C_q.

We know that there are mainly two ways to combine the basic select conditions: AND($\wedge$) or OR($\vee$). Therefore, we only study the following two cases: ($con1 \wedge con2$) and ($con1 \vee con2$). More complex selection cases can be derived based on these two basic cases. The cost are shown below, where p is the *True Value Probability*, which defines the probability of $con1$ being *true*. We adopt the heuristic rules discussed in [9] to estimate p.

$$\begin{aligned} c_{s1} &= p * (c_{su} + c_{su}) + (1-p) * c_{su} \\ &= (1+p) * c_{su} \\ c_{s2} &= p * c_{su} + (1-p) * (c_{su} + c_{su}) \\ &= (2-p) * c_{su} \end{aligned}$$

As for the projection's implementation, each event comes into the processing engine and the engine copies the selected attributes into the output event. We defines *Unit Projection Cost*, c_{pu}, as the cost to copy one attribute. This cost can be varied for the attributes with different data types. c_{pu} is assumed as the average value. With this assumption, a projection operator with the attribute list $(a_1, a_2 ... a_n)$ has the cost c_p as:

$$c_p = n * c_{pu}$$

Unlike the selection and projection operators, join operator is a binary full-relation operator. That is, the join operation needs the presence of the full relations. However, unlimited stream size makes traditional join algorithms not suitable for stream query processing. Consequently, many stream query processing engines implement join operation over a sliding window, instead of the full stream. Here, we only discuss the join cost c_j of a pair of input events. The complete cost for a join operation implemented on sliding windows can be computed based on c_j and event passing rate discussed in Section 3.2.2.

Given the selectivity of a join operation is σ, $R \bowtie S$ is actually equal to $_\pi(_\sigma(R \times S))$. That is, join operation is implemented based on selection and projection. Therefore, the join operator's *Computation Cost* to process an event pair, c_j, as defined in Section 2 should be:

$$c_j = c_{su} + \sigma * n * c_{pu}$$

Once we have each operator's computation cost for an event or event pair, we can get each operator's cost per time unit by multiply the operator i's cost c_i with its input rate r_i. Further, we can sum these cost together to get a query plan's cost C_q, which reflects how much CPU time consumed by the whole plan in a time unit.

$$C_q = \sum_{i=1}^{k} (c_i * r_i)$$

3.2.2 Evaluation Model for Output Rate

Viglas and Naughton [19] give a series of formulas to estimate the output rates for three basic query operators: selection, projection and join. However, their model requires computing the solution to an integral which is not efficient to evaluate a large number of query plans. Although they use rough heuristics to approximate integrals, this causes inaccurate estimations. Hence, we refine their models with a new estimation techniques on the join operation.

The join operator is a *Binary Full-Relation* operation. Traditional join algorithm does not work under streaming query processing scenario, where different *Sliding Window Based* join algorithms are applied. The *Output Rate* varies with different algorithms. In this paper, we study the join operator that executes natural join over a time-based sliding window [3], where the window size T_w is defined by the time interval. Other join implementations have similar analysis. We plan to do more complete study for various operator implementations in the future, as we will discuss in Section 6.

As shown in [17, 16], a time-based sliding window can be implemented at low cost, and when tied to input stream rate, can be more intuitive for users than a count-based sliding window, where the window size is defined as the number of interested events.

Suppose two streams of events R and S. R is composed of events e_i: $R = \{e_i\}$ and $0 < i < \infty$. i is a monotonically increasing number. Similarly, $S = \{e_j\}$ and $0 < j < \infty$. Further, suppose stream R has an input rate of r_1 and S an input rate of r_2.

There are $r_1 * T_w$ events at any time residing in the sliding window of stream R and $r_2 * T_w$ events residing in the sliding window of stream S. Without loss of generality, we take as the observation period the time interval $[t_0, t_0 + t]$. During this period, $r_1 * t$ events will arrive on stream R, each of which will be joined by Cartesian product to all events in stream S's sliding window. Given the selectivity σ, those joins generate $r_1 * t * r_2 * T_w * \sigma$ output events during the observation interval. An event output from a join is a tuple $< e_i, e_j >$ where $e_i \in S$, $e_i \in R$. Examining this from the side of stream S instead, with similar analysis we obtain $r_2 * t * r_1 * T_w * \sigma$ events output during the observation interval t.

Putting together, there are

$$\begin{aligned} r_1 * t * r_2 * T_w * \sigma + r_2 * t * r_1 * T_w * \sigma \\ = 2 * (r_1 * r_2 * t * T_w * \sigma) \end{aligned}$$

events generated during such an observation period.

Operator	Output Rate
Selection(r_i,σ)	$r_o = \sigma * min(r_i, r_p)$
Projection(r_i)	$r_o = min(r_i, r_p)$
Join($r_{i1}, r_{i2}, T_w, \sigma$)	$r_o = min(2 * r_1 * r_2 * T_w, r_p) * \sigma$

Table 3. Output rate estimations.

Therefore, the output rate is the number of events divided by the observation time t, that is:

$$\begin{aligned} r_o &= 2 * (r_1 * r_2 * t * T_w * \sigma)/t \\ &= 2 * r_1 * r_2 * T_w * \sigma \end{aligned}$$

Table 3 lists output rate estimations for three kinds of basic operators.

3.2.3 Evaluation Model for Memory Consumption

As for memory consumption, we only consider memory used to keep state information. That is, we do not consider the memory consumed for queueing events between operators. Within an SPJ query which is the focus of this paper, only the $JOIN$ operator needs to maintain states to join two input streams. Therefore, we only discuss memory consumption of join operators.

As for join implementation, we still follow the time-based sliding algorithm discussed in Section 3.2.2. Given sliding window size as T_w on both sides, we will need memory $r_1 * es_1 * T_w$ and $r_2 * es_2 * T_w$ on either side respectively, where r_1, r_2 are input rates and es_1, es_2 are event size of two input streams. Put them together, we will get the total memory needs of this join operator:

$$m_j = (r_1 * es_1 * T_w) + (r_2 * es_2 * T_w)$$

4 Experiments

In this section, we first experimentally evaluate our cost model. Then, we show how our multi-model optimization framework differentiate three logically equivalent plans.

4.1 Experiment Setup

All the testing programs are written in C++. To simulate the system's memory constraints, we use a global variable to count the consumed memory. When it reaches the limit, we start dropping input events to simulate real memory constraints. As for the computation cost constraints, we simulate that by adding the delay in the operators. The hardware setup includes a dual processor 2.8GHz workstation with 2GB memory running RedHat Enterprise Linux(RHEL).

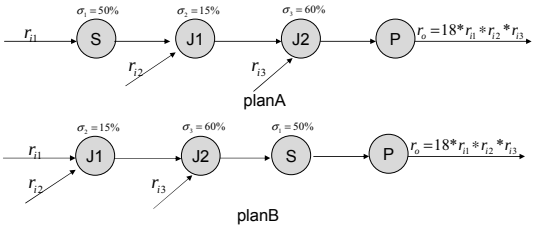


Figure 3. Equivalent SPJ plans.

Plan	Estimated Cost	Real Cost
planA	$9.458ms$	$5.654ms$
planB	$61.4ms$	$56ms$

Table 4. Cost model evaluation.

4.2 Evaluation of Cost Model

As discussed in Section 3, evaluation models are the basis of our optimization framework. Without accurate estimation of plans' metrics, we can not correctly choose an optimal plan. Hence, we first experimentally evaluate our cost model by studying two equivalent SPJ plans, which are shown in Figure 3. Three input streams, stream1, stream2 and stream3, are involved in this query. Their event formats contain 13, 8 and 12 name-value pairs respectively, plus a vector of data. All three input streams are generated as the rate of $1event/sec$ and each input event has the size of $0.5MB$.

We estimate the computation cost of two plans using the cost model introduced in Section 3.2.1. Meanwhile, we measure two plans' computation costs by recording all of their operators' processing time during an observation period of $T = 60sec$. In this experiment, we use a sliding window size of $T_w = 10sec$ for both join operators. All three input streams arrive at the rate of $1event/sec$. The estimated costs and measured costs are compared in Table 4. From the results, we can see that the plan yielding the lowest estimated cost, which is the one that would have been chosen, has the lowest actual cost, too. These results generalize to other SPJ queries which are not presented here.

Our cost model not only correctly orders two plans, but also accurately shows the difference between two plans.

4.3 Multi-Model Framework

In this experiment, we experimentally demonstrate that the three metrics used in the multi-model framework are necessary. Three equivalent multi-join plans are given in Figure 4. The parameters of join operators and input streams are listed in Table 5 and Table 6. These plans run on a system which has $256MB$ memory limitation.

Based on our evaluation models, we estimate each plan's needs of computation cost and memory. The result shows that only planB, which requires $234MB$ memory and $0.42sec\ CPU$ time per second, is under the scenarioI: the

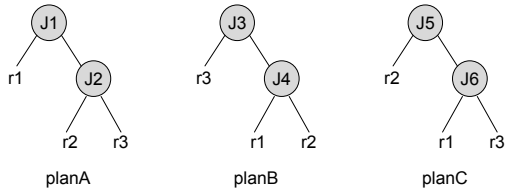


Figure 4. Equivalent multi-join plans.

Operator	Selectivity	Computation Cost
J1	0.3	$0.04ms$
J2	0.3	$0.025ms$
J3	0.45	$0.025ms$
J4	0.2	$0.05ms$
J5	0.9	$0.5ms$
J6	0.1	$0.025ms$

Table 5. Parameters of join operations.

system can provide it adequate computation and memory resources. Both planA and planC, which have resource requirements as ($23MB$, $1.06sec$) and ($210MB$, $4.1sec$) respectively, fail to satisfy the computation cost constraints, therefore, our framework will choose planB based on the search algorithm discussed in Section 3.

We run all three plans in the system with the resource constraints as specified and measure output rates. Figure 5 illustrates the results, from which we can see that planB has higher output rate. Therefore, planB is an optimal plan as our framework expects.

5 Related Work

Literature is rich with work on query optimization in traditional databases [18, 12, 7]. However, traditional optimization models are not appropriate for stream query processing because of the challenges raised by stream processing, as discussed in Section 1.

Several groups have contributed to the literature for query optimization in stream systems. STREAM [2] applies Synopsis Sharing within a single query or among multi-queries. NiagaraCQ [8] exploits incremental group optimization with expression signature. Borealis [1] has designed the QoS based multi-level optimization framework for a distributed stream query processing system. However, all three systems, plus [13, 5, 11], only consider the computation cost while evaluating a query plan. Viglas and

Stream	Input Rate	Event Size
r1	$1event/sec$	$1MB$
r2	$1event/sec$	$20KB$
r3	$10event/sec$	$2KB$

Table 6. Parameters of input streams.

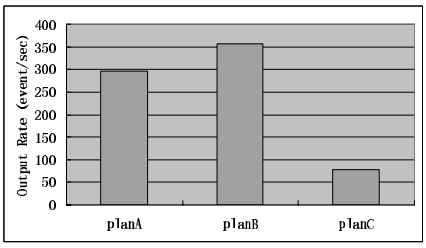


Figure 5. Output rate comparisons.

Naughton [19] consider the output rate in their rate-based optimization model. Ayad and Naughton [4] recognized resource constraints while maximizing the output rate, but do not consider memory consumption which is an important metric for stream processing in the scientific applications. Our multi-model based optimization framework factors in three metrics: output rate, computation cost and memory consumption.

6 Conclusion and Future Work

In this paper, we present our multi-model based optimization framework for stream query processing, which has a combined optimization goal and three evaluation models. With our framework, we can find an optimal plan not only for the the system with adequate resources, but also for the system with resource constraints.

In our study for this multi-model based optimization framework, we also observed some open issues that warrant further research. First, all above work is on a centralized system. However, because streams are distributed, a fact that follows from wide-spread distribution of sensors, a centralized stream data management system is likely to result in limited performance. Therefore, extending such a multi-model query optimization framework to a distributed environment is ongoing work. Second, this paper focuses on the *time-based sliding window* algorithm for join operation. However, there are many other implementations and algorithms [3, 10, 21], e.g. count-based sliding window, pipelined hash join and etc. We plan to conduct more complete study on the rate/cost estimation of the operators with various implementations and algorithms.

References

[1] D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik. The Design of the Borealis Stream Processing Engine. In *CIDR Conference*, 2005.

[2] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom. Stream: The stanford data stream management system. In *Data Stream Management*, 2004.

[3] A. Arasu and J. Widom. Resource sharing in continuous sliding-window aggregates. In *30th VLDB Conference*, 2004.

[4] A. M. Ayad and J. F. Naughton. Static optimization of conjunctive queries with sliding windows over infinite streams. In *ACM SIGMOD international conference on Management of data*, 2004.

[5] S. Babu, R. Motwani, K. Munagala, I. Nishizawa, and J. Widom. Adaptive ordering of pipelined stream filters. In *ACM SIGMOD international conference on Management of data*, 2004.

[6] S. Babu and J. Widom. Continuous queries over data streams. *SIGMOD Record*, 30(3):109–120, 2001.

[7] S. Chaudhuri. An overview of query optimization in relational systems. In *PODS Conference*, 1998.

[8] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. Niagaracq: A scalable continuous query system for internet databases. In *SIGMOD Conference*, pages 379–390, 2000.

[9] H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database System Implementation*. Prentice-Hall, 2000.

[10] L. Golab and M. T. Özsu. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, 2003.

[11] L. Golab and M. T. Özsu. Update-pattern-aware modeling and processing of continuous queries. In *ACM SIGMOD international conference on Management of data*, 2005.

[12] Y. E. Ioannidis. Query optimization. *ACM Comput. Surv.*, 1996.

[13] J. Kang, J. Naughton, and S. Viglas. Evaluating window joins over unbounded streams. In *Int. Conf. on Data Engineering (ICDE)*, 2003.

[14] Y. Liu and B. Plale. Query optimization for distributed data streams. In *Manuscript submitted*, 2006.

[15] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In *CIDR Conference*, 2003.

[16] B. Plale. Leveraging run time knowledge about event rates to improve memory utilization in wide area data stream filtering. In *IEEE HPDC Conference*, 2002.

[17] B. Plale and N. Vijayakumar. Evaluation of rate-based adaptivity in joining asynchronous data streams. In *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2005.

[18] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD Conference*, 1979.

[19] S. D. Viglas and J. F. Naughton. Rate-based query optimization for streaming information sources. In *SIGMOD Conference*, 2002.

[20] N. Vijayakumar, Y. Liu, and B. Plale. Calder query grid service: Insights and experimental evaluations. In *CCGrid Conference*, 2006.

[21] A. N. Wilschut and P. M. G. Apers. Dataflow query execution in a parallel main-memory environment. *Distributed and Parallel Databases*, 1(1):103–128, 1993.

Applying MDA to the Conceptual Design of Data Warehouses

Leopoldo Zepeda
Department of information systems
Instituto Tecnológico de Culiacán, México
leopoldo@correo.ccs.net.mx

Matilde Celma
Department of information systems and computation
Technical University of Valencia, Spain
mcelma@dsic.upv.es

Abstract

In this paper, we present a method based on Model Driven Architecture (MDA) for the conceptual design of Data Warehouses (DWs). The method is made up of a set of transformation rules as a mechanism to extract multidimensional schemas from the conceptual description of the operational database and a Computation Independent Model (CIM) for the definition of user requirements. Next the generation of multidimensional schemas, we use user requirements to guide the selection of the multidimensional schema that is supported by the operational database and most likely satisfy user requirements.

1. Introduction

A DW is a database used for analytical processing whose principal objective is to maintain and analyze historical data [1]. Multidimensional data modeling plays a key role during the design of a DW. Since the introduction of multidimensional modeling, several design techniques have been proposed to capture multidimensional data at the conceptual level. Some of them consist in analyzing the schema of the operational database. But in fact designing a DW requires more knowledge about user requirements. As yet there is not a common strategy for the conceptual design of DWs, current DW methodologies can fall with in two basic groups [2]: a) Data-driven (also called supply-driven) approaches design the DW starting from a detailed analysis of the operational databases. b) Process-driven (requirement driven) approaches start from determining the information requirements of DW user.

On the other hand, MDA is a standard to develop software by transforming an input model into an output model [3]. MDA separates the specification of system in a Platform Independent Model (PIM) from the specific technology in a Platform Specific Model (PSM). Besides these models, a CIM is provided by MDA as means of modeling requirements. In this paper, we have aligned a method for the conceptual design of DWs with the standard MDA. This method is employed within a mixed data and process-driven approach. For this, we define three phases. The first phase is devoted to examining the ER schema of the operational databases, finding out candidate multidimensional schemas for the DW (data-driven), this is achieved through a set of transformation rules between the Entity Relationship (ER) and the On-Line Analytical Processing (OLAP) metamodel. In order to define these transformations, we use the Query View Transformation (QVT) Operational Mapping Language [4]. In the second phase, a CIM is developed (process-driven), for the basis of our elicitation method, we adopt a task model. The final phase integrates these two viewpoints, and thus generates a feasible solution (supported by the existing data sources) that best reflects the user's requirements.

Our main contribution is a new design methodology, which is not only initialized from user's requirements, but also from the operational database schema. The paper is structured as follows: In Section 2, we briefly review previous approaches on DW design. An overview of MDA is presented in section 3. Section 4, introduces our Method. Finally, Section 5 draws some conclusions and future works.

2. Related Work

In this section, some research works in the field of DW conceptual design are classified and summarized.

Data-driven methodologies: Bill Inmon, argues that DW design is a data driven, in comparison to classical systems [5]. Ones of the most important data-driven approaches are presented in [6] [7]. In [6] the authors illustrate a method for developing multidimensional schemas. The design method starts from an existing ER schema, derives a multidimensional schema, and provides implementations in terms of relational tables as well as multidimensional arrays. In [7], the design of a conceptual schema is carried out by producing a fact schema for each fact, which, can be derived from an ER schema using an algorithmic procedure. The procedure transforms the ER schema in a tree. This model represents the fact as the tree's root and the dimension as the tree's descendents.

Process-driven methodologies: In these methodologies, collecting user requirements is given more relevance. In [8] is presented an approach that is based on the SOM (Semantic Object Model) technique. In this approach the business process is analyzed by applying the SOM interaction schema that highlights the customers and their transactions with the process studied. In [9] a goal-oriented approach to requirements analysis based on the Tropos methodology is proposed. In this approach two different perspectives are integrated for requirement analysis: organizational modeling and decisional

modeling. These proposals both data and process-driven have a number of drawbacks namely they are: (a) data-driven without sufficient emphasis or capturing user requirements, (b) the use of proprietary notation is a deficiency, provided that it turns these methodologies in particular and isolates solutions, (c) process-driven requirements are directly used to build a conceptual model without any feedback from the data sources.

3. Basic Concepts in MDA Technology

In this section we summarize MDA, QVT and their main characteristics. MDA is a standard [3] that addresses the cycle of designing, deploying, and managing applications by using models in software development. The key technologies of MDA are: Unified Modeling Language (UML), Meta-Object Facility (MOF), EXtensible Markup Language (XML), and Common Warehouse Metamodel (CWM). MDA separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. Thus, MDA encourages specifying a PIM which contains no information specific to the platform. Then, this PIM can be transformed into a PSM in order to include information about the specific technology. MDA also presents a CIM, this model describes the system within its environment and shows what the system is expected to do. CWM is a PIM definition for DW. CWM provides a set of metamodels to model an entire DW including data sources, MD modeling, OLAP, etc. The basis for these models (CIM, PIM and PSM) is the concept of viewpoint, where a viewpoint on a system is a technique for abstraction using a selected set of concepts and structuring rules. Nowadays the most crucial issue in MDA is the transformation between models. Model transformations are categorized as either vertical or horizontal. Vertical transformations occur when a source model is transformed into a target model at a different level of abstraction (PIM to PSM). A horizontal transformation involves transforming a source model into a target model that is at the same level of abstraction (PIM to PIM). The QVT specification [4] has a hybrid declarative-imperative nature. The declarative part is split into a user-friendly part based on transformations which comprises a rich graphical and textual notation. The declarative notation is used to define the transformations that indicate the relationships between the source and target models, but without specifying how a transformation is actually executed. In this way, QVT also defines operational mappings. This allows defining the transformations which use a complete imperative approach.

4. Our Method

Following the MDA guideline this method has been organized in two views: CIM and PIM (Figure 1).

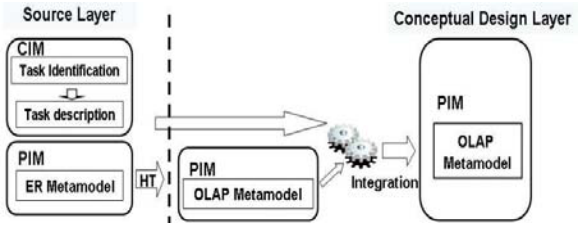


Figure 1. Method organization

In this method, we identify two layers for the definition of the conceptual DW schema:

-Source layer that defines the data sources of the DW. For this layer, we use the CWM ER metamodel [10] as modeling formalisms for the source PIM definition and we specify the early requirements of DW application by means of a task model (CIM).

-Conceptual Design layer, which defines the structure of the conceptual DW schema. CWM OLAP metamodel [11] is used as modeling formalism for the target PIM definition.

Our proposal, as can be seen in Figure 1, introduces a new model for requirements elicitation and a process for the automatic construction of candidate multidimensional (OLAP) schemas from ER schemas (through a set of horizontal transformation rules (HT)).

Finally, an integration process generates the multidimensional schema that best reflect user requirements and is supported for the existing data base. Following, we describe the three phases of our method.

4.1 Phase 1. Obtaining a set of Candidate Multidimensional Schemas from ER Schemas

This phase performs an exhaustive analysis of the ER schema in order to discover the entities that are candidates to become cubes in the multidimensional (OLAP) schema. Then, each candidate cube entity is taken as the center of the candidate OLAP schema, produced by considering all possible dimensions reachable from the center through many-to-one or many-to-many Relationships. This is achieved through a set of horizontal transformation rules between the ER and OLAP metamodels.

In this section, we first describe the metamodels. Next, we show the transformations rules. Finally, each transformation is described by means of one example.

4.1.1 Metamodels description

The CWM ER metamodel (Figure 2), contains all the elements of an ER model (i.e. Entity, RelationShip, etc) In the ER metamodel each Entity may own Attributes A RelationShip represents associations between two or more. Entities. Each end of a RelationShip is a RelationShipEnd, which may optionally have name and multiplicity.

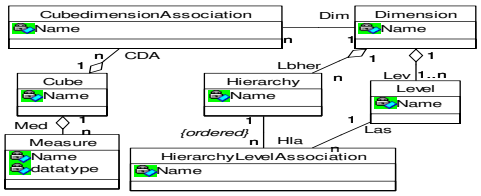


Figure 2. OLAP metamodel

The CWM OLAP metamodel (Figure 3) is structured into a schema class that owns all elements of an OLAP model (i.e. Dimensions and Cubes). In the OLAP metamodel, each Dimension is a collection of Members. Cubes are used to store Measures and they are related to the Dimensions through the CubeDimensionAssociation class.

4.1.2 Transformation rules

One of the advantages of MDA is the development of automatic transformations between models. In this paper, transformations are given using the QVT Operational Mapping Language [4]. This language allows the definition of transformations that are expressed imperatively. Each transformation contains the following elements:

-Mapping: a mapping operation implements a transfotmation

-When: it specifies the condition that must be satisfied by all model elements participating in the transformation.

We have developed each transformation to obtain the OLAP PIM from the ER PIM (PIM-PIM). Due to space constraints we can only describe some of the defined transformations.

4.1.2.1 Ent2Cub

The goals of this rule are identifying the candidate cube entities and transform them to a matching Cube class. We call candidate cube entities the entities in the ER schema on which the decision-making process is focused. A candidate cube entity may be represented on the ER schema as an entity that has references to other entities but no other entities have references to it. According to the transformation rule an Entity with opposite multiplicity equals to many (clause When) gets transformed to a corresponding Cube, having the same name of the Entity but prefixed with a "C". Once this transformation is done, the transformation rules *Atr2Med* and *RSE2CDA* must be done.

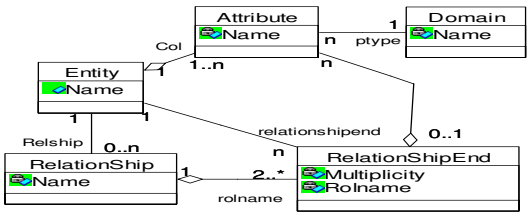


Figure 3. ER metamodel

```
Mapping Entity:: Ent2Cub() : Cube
when (self.oppositeassociationEnd →not empty()
implies self.oppositeAssociationEnd.Multiplicity
is (*))
{Name := "C"+ self.name;
Med := self.Col→Map Atr2Med();
CDA := self.Relship→Map RSE2CDA();}
```

4.1.2.2 Atr2Med

The numeric attributes of the candidate cube Entity, gets transformed to a corresponding Measure of the Cube through the transformation rule *Atr2Med*. The textual representation is as follows:

```
Mapping Attribute::Atr2Med : Measure
When (self.ptype.Name="Integer" or
self.ptype.Name="Float")
{Name := "M"+self.name;
dataType := self.ptype.Name;}
```

4.1.2.3 RSE2CDA

The goal of *RSE2CDA* is to convert each RelationShip of the candidate cube Entity with many-to-one or many-to-many multiplicity to a CubeDimensionsAssociation class. This is textually represented as follows:

```
Mapping RelationShip::RSE2CDA ():
CubeDimensionAssociation
When (self.Association.Multplicity is("*"))
{Name := "CDA"+self.name;
Dim := self.map RSE2Dim();}
```

4.1.2.4 RSE2Dim

The transformation rule *RSE2Dim*, generates a Dimension for each RelationShip of the candidate cube Entity. In this rule, a RelationShip gets converted to a matching Dimension, having the same name as the RelationShip, but prefixed with a "D". Once this transformation is done, the transformation rules *Ent2Lev* and *SecE2Hie* must be done. We wish to point out that the transformation rule *SecE2Hie* depends of special structures. Due to space constrains, we only show the *Ent2Lev* transformation rule.

```
Mapping RelationShip::RSE2Dim () : Dimension
{Name := "D"+self.name;
Lev := self.map Ent2Lev();
Lbher := self.Hierarchies →Map SecE2Hie();}
```

4.1.2.5 Ent2Lev

Each Entity that is directly linked to a candidate cube Entity is matching with a Level. The core of the transformation rule is the recursive call. The goal of the recursive call is look for entities through many-to-one or many-to-many multiplicity, and transform them at dimension's Level. In this rule, an Entity gets converted to a corresponding Level, having the same name as the Entity, but prefixed with a "L".

```
Mapping RelationShip ER::Ent2Lev() : Level
When (self.Association.Multplicity is("*"))
{Name:=´L´+self.oppositeassociationEnd.resolveon
e(#entity).name;
self.oppositeassociationEnd.resolveone(#entity).
RelationShip →Ent2Lev();}
```

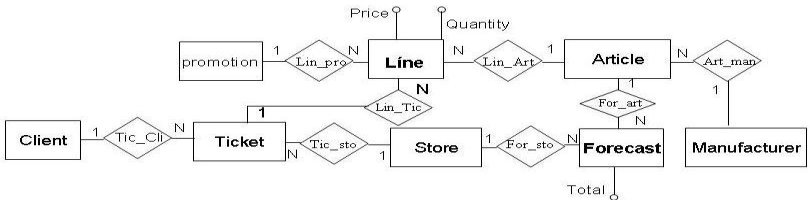


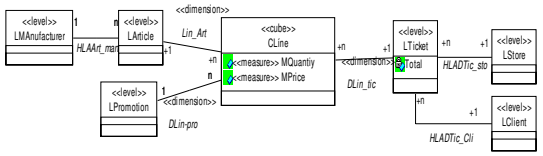
Figure 4. ER schema

4.1.3 An Example

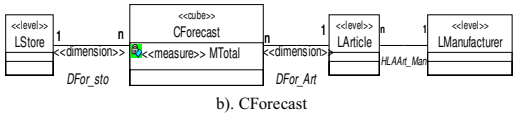
In this section, we provide an example to show how to apply the presented transformation rules to an ER PIM (Figure 4) in order to obtain the corresponding OLAP PIM. The result of the first transformation is given by *Ent2Cub*. This rule identified two candidate cube entities in the ER schema: *Line* and *Forecast*. Using a textual representation a partial transformation for the candidate cube entity *Line* can be tracked from T1 to T9. In T1, the candidate cube entity *Line* is transformed into a Cube class. Once this transformation is executed, the following transformation rules, *Atr2Med* and *RSE2CDA* are executed (T2-T6)).

T1: Entity (Name="Line") →*Ent2Cub*→ Cube (name="CLine").
T2: Attribute (Name="Quantity")→*Atr2Med*→ Measure ("MQuantity")
T3: Attribute (Name="Price")→*Atr2Med*→ Measure ("MPrice")
T4: Relationship (Name="Tic_Lin")→*RSE2CDA*→ CubeDimensionASsociation (Name="CDATic_Lin)
T5: Relationship (Name="Lin-Pro")→*RSE2CDA* → CubeDimensionASsociation(Name="CDALin-Pro)
T6: Relationship (Name="Lin-Art")→*RSE2CDA* → CubeDimensionASsociation (Name="CDALin-Art)
T7:Relationship(Name="Lin_Tic")→*RSE2Dim*→ Dimension (Name="DLin_Tic)
T8: Entity (Name="Ticket")→*Ent2Lev* →Level (Name="LTicket")
T9: Entity (Name="Client")→*Ent2Lev* →Level (Name="LClient")

The transformation rules *RSE2Dim* and *Ent2Lev* can be tracked by T7 to T9. The candidate multidimensional schemas obtained from the ER schema are shown in Figure 5. As we can see the elements of the schemas are stereotyped with the cube, dimension, level and measure keyword to indicate its relationship with the concepts of the OLAP metamodel.



a). CLine



b). CForecast

Figure 5. Candidate multidimensional schemas

4.2 Phase 2. User Requirements (defining the CIM)

The CIM proposed by MDA is built mainly to bridge the existing gap between those that are experts about the domain and those that are experts on how to build the artifacts that satisfy the requirements domain [12]. Then, according to MDA, a CIM must describe the requirements of the system. We specify the early requirements of a DW by means of a task model. This model is built from the tasks that users must be able to achieve when interacting with the DW. The requirements specified when building the task model are used in following stage for select and refine a candidate multidimensional schema. We propose two steps to define the task model: (1) Task identification: we identify the set of tasks that the system together an actor must achieve to accomplish each requirement. The set of identified tasks are organized in task tree. (2) Task description. To accomplish the goal defined by each leaf task included in the task tree, we describe the set of actions that must be performed to succeed in achieving that goal. This description is made by using the UML Activity Diagrams [13].

4.2.1 Task identification

To identify the set of tasks that represent the DW user requirements we must detect, as a first step, which actors can interact with the system. Then, for each detected actor we must define a task tree that represents the tasks that this actor can achieve when interacting with the system. In our example, the actor detected is: *sales manager*. The task tree associated to this actor is shown in Figure 6. For the construction of the task tree, we take as the starting point, a statement of purpose that describes the main goal of the DW application. The statement of purpose is considered most general task of the system. From this task, a progressive refinement is performed, obtaining as a result more specific tasks. Tasks are decomposed into subtasks by following structural refinements. The structural refinement (represented by solid lines in Figure 6) decomposes complex tasks into simpler subtasks.
The statement of purpose of this system is decomposed into two tasks: *Analyze store sales* and *Analyze planned sales*.

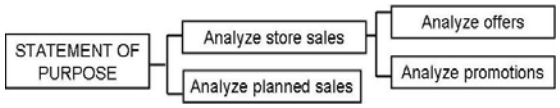


Figure 6. Task tree

At the same time, the task *Analyze store sales* is divided into two tasks: *Analyze offers, Analyze promotions.* In the following section we introduce a strategy to describe each identified tasks. To better understand this strategy we show the description of one task of the presented task tree: *Analyze promotions*.

4.2.2 Task description

In order to describe the set of tasks identified in the previous step we need to indicate the DW actions that are needed to complete each task. For every task we must distinguish the interaction between actors and the DW. To do this, we use UML Activity Diagrams, showing the workflow performed to obtain some task, indicating the roles that are in charge of each activity, and the data required and produced by each activity. Figure 7, shows an activity diagram for the description of the *Analyze promotions task*. There exists a swinlane for every role participating, including the activities performed by that role. The diagram also shows the data needed and produced by each activity. Data appear as objects that flow between activities. We refer to these objects as Data Objects (DO). We distinguish two different types of DOs. (1) Output DO: the system provides actors with information about data. (2) Input DO: the system is waiting for the user to introduce some data; this information is taken by the system to correctly perform a specific activity. The *Analyze promotions task* (see Figure 7) starts with the system activity *Analyze the quantity sold* in promotions by *year, promotion* and *store*. This activity searches information that matches with the information provided by the DW user through an Input DO. Once this action is finished, the task continues with an Output DO, where the DW system provides the DW user with the list of matched promotions. In order to make task descriptions easy, we just indicate the task which the information is related to. This information is specified by means of a technique based on information templates that is next introduced.

4.2.3 Describing the DW data

The information that might be stored in the DW is represented by means of a template technique. We propose the definition of an information template (see Figure 8) for each task identified. In each template we indicate the task name and a specific data section. In this section, we describe the information in detail by means of

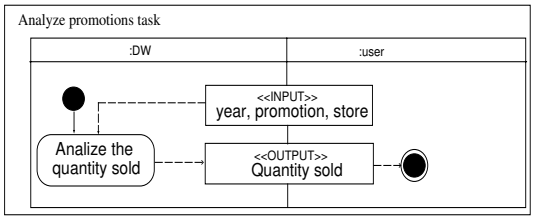


Figure 7. Search process

Task	Analyze promotions			
Data	**Name**	**Description**	**Type**	**DO**
	promotion	promotion identifier	String	InDO
	year	year identifier	Date	InDO
	article	article identifier	String	InDO
	Quantity sold	total quantity sold	Number	OutDO

Figure 8. Task description

a list of specific properties associated to the task. For each property we provide a name, a description and a data type. In addition, we use these templates to indicate the information shown in each DO. For each property we indicate the DOs where it is shown. To identify a DO we use the next notation: OutDO for Output DOs and InDO for Input DOs. According to the template showed in Figure 8, the information that the DW must store about a promotions task is (see the specific data section): *promotions, year, store* and *Quantity sold*.

4.2.4 Template interpretation

The information template can be revisited and reinterpreted to select a candidate multidimensional schema. To do so, the items listed in the *DOs* section are considered as measures and dimensions in the multidimensional schema. Then, the *InDO* defines the variables that may cause changes to measures (dimension analysis) and each *OutDO* contributes a measure. The information template of Figure 8, can be interpreted as follows: the *InDO*s (*promotion, year, and article*) detail the dimensions, while the *OutDO* (*Quantity sold*) detail the measure. Once identified the measures the designer must decide which aggregation operations to use: for instance we can presume that the measure *Quantity sold* it to be aggregated through the *Sum* function.

4.3 Phase 3. Integration

In this section, the requirements identified in phase 2 are mapped with the elements of the candidate multidimensional schemas to generate the conceptual schema for DW. Two steps are involved: (1) *selection*, where a candidate multidimensional schema is selected; and (2) *refinement*, where the selected multidimensional schema is manually modified with the goal of represent the user expectations.

4.3.1 Selection

Given user requirements and candidate multidimensional schemas, the problem is to find semantic mappings between their elements. For this, we use a *structure-level* matching technique [14]. This technique only considers the properties of the elements, such as name, description, data type and not instances data. For example, the Level *LPromotion* of the multidimensional schema *CLine* (Figure 5a), matches with the dimension *promotion* identified in phase *2* and the measure *MQuantity* matches

the measure *Quantity sold*. Table 1, resume the number of properties of each candidate multidimensional schema (for the *Analyze promotions* task). From this information we can select the multidimensional schema *CLine*, because it captures better the user requirements and is supported by the operational database.

Table 1. Schema matching

Property\Schema	CLíne	CForecast
Measures	1	0
Dimensions	2	1

4.3.2 Refinement

This step is aimed at reorganizing the selected multidimensional schema in order to better fit them to the users´ requirements. During this process the operations that can be carried out are: remove unnecessary attributes, remove dimension's levels, add measures, add summarization constraints and define time granularity (Due space constraints, we only describe the process to remove a dimension level).

-Remove a dimension level: probably, not all of the levels represented in the selected multidimensional schema, are interesting for DW. Thus, the level must be removed. In general, removing a level correspond a change in the granularity of cube instances, if the level eliminated has two or more descendents levels, leads to increasing the number of dimensions in the multidimensional schema [7].

For instance, in the final multidimensional schema (Figure 9), we assume that the user want to classify the information directly by *LClient* and *LStore* levels without considering the *LTicket* level generating two new dimensions: *DLin_Cli and DLin_Sto*. For this example, we also assume that the user is interested in day granularity (Time dimension) and that the measure *Quantity sold* is computed with the aggregation operator *Sum.*

5. Conclusions and future works

In this paper, we have introduced our MDA method for DW conceptual design. The choice of MDA as a guiding methodology for the first phase is justified by the need of exploiting an ER schema using standardized means. In conclusion, the experience we have gathered by applying the proposed method in our case study is encouraging.

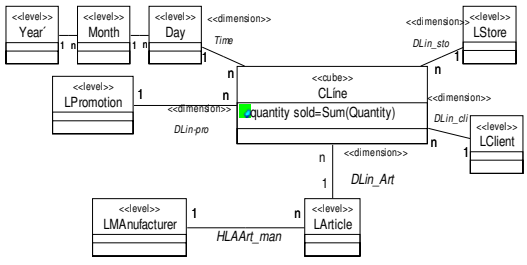


Figure 9. Final multidimensional schema

The method can be essential to direct the designer toward a solution that is both efficient to implement and consistent with users' requirements. We think that the description of each metamodel and its transformations is a good example about the use of MDA in conceptual design of DWs, at the same time the final DW schema is strongly rooted to the operational database which makes the design of ETL simpler. The transformation rules were programmed with the Atlas transformation language [15] btaining the expected results. We plan for future work to extend the approach presented in this paper by considering richer criteria for the third phase.

References

1. Kimball, R. Ross, M.: The Data Warehouse Toolkit, 2ed edition, John Wiley & Sons. (2002).
2. Winter, R. and Strauch, B.: A Method for Demand-Driven Information Requirements Analysis in Data Warehousing projects. HICSS, Hawaii USA (2003) 231.
3. Kleppe, A. Warmer, J. Bast, W.: MDA Explained. The Practice and Promise of the Model Driven Architecture. Addison Wesley. (2003).
4. OMG 2nd Revised Submission: MOF 2.0 Query/Views/Transformations. Internet: http://www.omg.org/cgi-bin/doc?ad/05-03-02 (2005).
5. Inmon, W.H.: Building the Data Warehouse. 3rd edition, John Wiley & Sons. (2002).
6. Cabibbo, L. and Torlone, R.: A Logical Approach to Multidimensional Databases. In Proceedings of the International Conference on Extending Data Base Technology (EDBT '98), Valencia, Spain (1998) 183–197.
7. Golfarelli, M. Rizzi, S.: A methodological Framework for Data Warehouse Design. In: Proc. of the ACM 1st Intl. Workshop on Data warehousing and OLAP (DOLAP'98), Washington D.C., USA (1998) 3-9.
8. Boehnlein, M. Ulbrich, vom Ende, A.: Business Process Oriented Development of Data Warehouse Structures. In: Proceedings of Data Warehousing (2000) 3-22.
9. Giorgini, P. Rizzi, S., Garzetti, M.: Goal-oriented requirement analysis for data warehouse design ACM 8th International Workshop on Data Warehousing and OLAP (DOLAP'05), Bremen, Germany (2005) 47-56.
10. Object Management Group (OMG), Common Warehouse Metamodel specification (Extensions). Internet: http://www.omg.org/cgi-bin/doc?ad/2001-02-02.
11. OMG, Common Warehouse Metamodel (CWM) specification 1.0.1. Internet: http://www.omg.org/cgi-bin/doc?formal/03-03-02
12. Wirfs-Brock, B. Wilkerson, and L. Wiener,: "Designing Object–Oriented Software.", Prentice–Hall, New ´Jersey, USA (1990).
13. OMG. Unified Modeling Language (UML) Specification Version 2.0 Final Adopted Specification. www.omg.org, (2003).
14. Rahm, E. Bernstein, P.: A survey of approaches to automatic schema matching. Very Larges Databases (2001) 334-350.
15. The Atlas Transformation Language, Internet: http://www.sciences.univ-nantes.fr/lina/atl/, (2005).

A Data Warehouse Architecture in Layers for Science and Technology

André Luís Andrade Menolli[1], and Maria Madalena Dias[2]

[1] *Departamento de Ciência da Computação, Faculdades Luiz Meneghel,*
Rod. BR 369, Km 54, Caixa Postal 261
86360-000 Bandeirantes, Paraná, Brasil
a_menolli@hotmail.com

[2] *Departamento de Informática, Universidade Estadual de Maringá,*
Avenida Colombo 5790
870200-900 Maringá, Paraná, Brasil
mmdias@din.uem.br
http://www.din.uem.br/~mmdias

Abstract

The data warehousing technology has been widely used in companies with the aim of offering organization, management and data integration. In the knowledge discovery process in database, the first step is the data preparation, where the data must be organized and stored in the data warehouse (DW). The construction of a DW is a complex and laborious task because it demands a high knowledge of the involved company business, and also of the contents of data sources and of the technologies, such as database, data integration and data warehousing. The choice of the architecture is an important decision to be making to build a DW. This paper presents a DW layered architecture of integrated and increased data marts for management in Science and Technology (S&T). This architecture is applied to build a DW that integrates the institutional bases of CNPq and CAPES.

Keywords. *Data Warehouse, Science and Technology, Data Integration.*

1. Introduction

Companies of different activities and government organizations have data in several data sources that contain important information for the support to decisions making. Thus, to make these decisions are necessary the understanding and preparation of these data.

Data Warehousing provides tools to support decisions making through the integration of all company data in a repository. The data can be accessed easily and analyzed without the lost time involved in the manipulation and processing. The decisions can be made quickly and with larger safety, because the data are always available and with the necessary consistence. "DW is an integrated, non-volatile, time-variant set of data based on subject matter, supporting management decisions" [11].

The organization can to build a DW or integrated Data Marts (DM) that facilitates the process and does not require great initial investment. DM is a departmental DW, built for an organization's particular area [11].

The Brazilian Science & Technology (S&T) environment have important databases that are consolidated and trustworthy.

To have a good planning in S&T is necessary that the manager has safe information for the decision making. For this, a DW is built, and a DW architecture would have to be chosen or defined.

The remainder of this paper is organized as it follows: Section 2, architectures proposed by other authors are related; Section 3, the proposed architecture is described; Section 4, the steps to build a DW for management in S&T are described; Section 5, some results obtained through queries to the DW are showed. Finally, in the Section 6 the conclusions of this paper are presented.

2. Related Works

In the project of a DW, the choice of the architecture and development methodology should be the first decisions to be making, because the organization of the components depends of these choices.

There are a lot of works that presents DW architecture, like [11] and [10].

Kimball et al. [10] proposed the Data Warehouse Bus Architecture in that the storage area is composed by several data marts projected to maintain the data integration. This Architecture is divided in two great components, the Back Room and the Front Room. According to [10], in spite of the Bus Architecture make possible to search the first results faster and with less cost, it has complex metadata to manage data distribution and integration.

3. Proposed Data Warehouse Architecture

In this paper is proposed a layered architecture of integrated and incremental data marts. With incremental DM, new data marts can be implemented or data marts can be modified allowing inclusion of new requirements. The main characteristic of this architecture is the replace of data sources of several formats for standardized databases, facilitating the data integration.

This architecture is constituted of five layers, represented in Figure 1 and described in the following sections. The thick arrows represent the load of data, and the fine arrows represent the access to the data.

3.1. Data Source Layer

In any DW environment the extraction of data from some data source is necessary, because DW is not a conventional database. The Data Source layer represents all data sources that are used as information suppliers to DW. These data sources can be in the reference format or in others formats. The reference format is chosen by the DW planner according with the format in that the data will be stored in DW.

3.2. ETL Layer

In this layer the data are extracted of the data sources and they are stored in the staging area. In this layer there are the largest problems in the development of a DW. According to [3], estimate that more than 1/3 of the cost and time are spent to ETL and problems like extraction, transformation and cleaning, which can take up 80% of the time spent in the development of a DW [6].

The ETL layer is divided in two modules, a migration module and another of extraction, transformation and load. The migration module transfers data in several formats for a reference format chosen by DW planner. The module of extraction, transformation and load collects relevant data, cleans, integrates, transforms and stores these data in the staging area.

3.3. Staging Area Layer

The staging area is an intermediate step between the extraction of the data source and the load in DW. In the staging area the data receive the first cleanings, transformations and integrations.

According to [9], a fundamental concept that greatly simplifies DW projects and facilitates maintenance is the use of data staging areas. Starting from the logical project of the staging area, it is possible to have a good idea of the attributes and sources tables necessary to populate the DW. The data staging should just contain necessary information to populate the warehouse.

The staging area layer and the ETL layer include all databases and tools that are necessary to transform, integrate, load, control the quality and clean the data [2].

The staging area, defined in this work, uses a normalized data model to allow a best data consistence and to facilitate the integration process among different databases.

3.4. Data Warehouse Layer

The DW layer represents integrated data marts. Dimensional model was used in the data model of the data marts and a bus matrix was elaborated to identify the intersection between the data marts.

The dimensional model was used because, according to [12], there are two main advantages of using a dimensional model in data warehouse environments. At first, a dimensional model provides a multidimensional analysis space in relational database environments. At second, a typical and not normalized dimensional model has a simple schema structure, which simplifies the processing of query and improves performance.

In the definition of the DW data model, the first step is the definition of the data marts that are in a single data source initially. In the second step, the dimensions are identified for data marts defined in the first step and the intersection between the data marts and their dimensions are marked. In the third step, is elaborated the bus matrix. In the sequence, the granularity of the fact tables of the data marts would have to be declared. Also is verified which dimensions are related to these tables. Finally, are defined the facts that compose the fact tables.

3.5. Analysis Area Layer

In the analysis area layer are defined the responsible components for the search of information and interesting knowledge in DW. These components can be classified as: OLAP tools, data mining tools and materialized views. OLAP tools can search information directly from DW. Materialized views can be used to facilitate the access to data. Data mining tools can need that the data are reorganized and formatted.

3.6. Metadata

The metadata contains information about data used by several components of the architecture. The information contained in this repository is essential for the complete operation and maintenance of the components.

It is necessary to know the information on the data that compose the DW, as the data were integrated and transformed, and as information and knowledge can be extracted from DW.

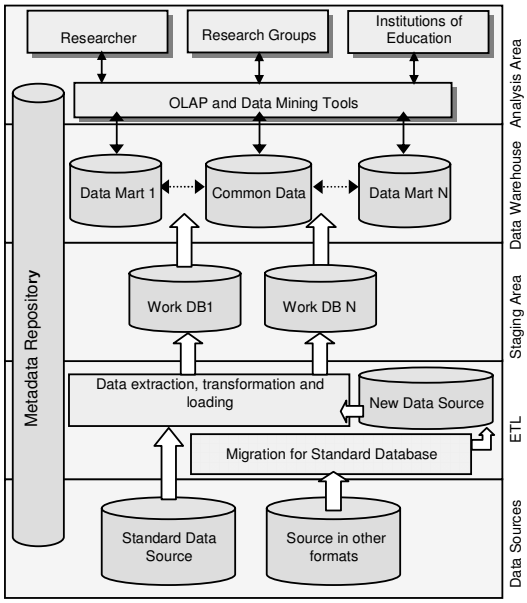


Figure 1. Proposed Architecture

4. DW for Management in S&T

A DW for management in S&T was built following the defined steps for [10].

The DW was projected according to the star schema. This modeling has been chosen for the already presented advantages in this research work and because dimensional modeling is widely accepted as the viable technique for delivering data to final users in a DW [1], [4], [8], [10].

A Bus Matrix was used to define data integration rules. This matrix represents the fact and dimensions tables previously structured from way to maintain integration between the data marts, as shown in Figure 2.

The ETL, Staging Area and DW layers were projected and implemented using resources as triggers and stored procedures of SGBD Oracle, besides tools as ERWin for data modeling, Rational Rose [5] for the project of the layers and Delphi for creation of the migration module to ETL layer.

The implementation of the migration module have been developed in Delphi 7.0 to make possible migration and integration of the data, because for each institution there was a different database of CAPES.

The extraction, transformation and load module of the ETL layer was implemented through stored-procedures built with the PL-SQL language of Oracle, solution that was shown quite efficient in performance and flexibility.

Some case studies had been made with the objective of analysis of the data stored in the DW. The results of these studies are described in the next section.

5 Case Studies

This section presents the results of some of the case studies made.

5.1. Study on Intellectual Property

The Brazil has some problems because does not have the culture to make patents and registers. An example of this is that Japanese companies patented the process of *cupuaçu* (fruit from Brazil north region) derivatives [7].

The application of DW demonstrated that amongst all the processes and products created by the researchers since 1990, only 0.08% patents were generated.

5.3. Published Books

The Figure 3 shows comparison in the number of published books or books chapters between men and women in diverse range of age.

The difference in the number of publications until 35 years between men and women is very small. In the range of the 35 until 45 years this difference increase, and in range of the 45 until 55 years it starts to decrease.

Process of the Business	Fact Table	Granularity	Data	Researcher Autor	Agency Financier	Graduation	Knowledge Area of Graduation	Institution of the Graduation	Institution	Person who orientates	Person who orientates in Graduation	Paper	Periodic	Thesis	Others Scientific production
	Publication of Papers	1 line for Publication by Author	X	X	X	X	X	X	X		X	X	X		
Management	Publication of Books	1 line for Book or Chapter by Author	X	X	X	X	X	X	X		X				
Works and	Scientific Productions	1 line for Production by Author	X	X	X	X	X	X	X	X	X	X		X	X
Scientific production	Publication of Thesis	1 line for Thesis by Author	X	X	X				X	X				X	

Figura 2. Bus Matrix of DW

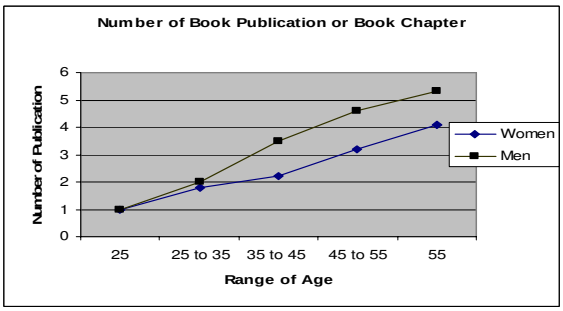


Figure 3. Number of book publication

6. Conclusions and Future Works

The success of a DW is directly related to the definition of implementation architecture and to the choice of a development methodology.

The proposed architecture was developed based on the BUS Architecture and in the methodology of integrated and incremental data marts. Each layer of the architecture was projected from way to get the biggest possible independence, and the superior layer uses parameters of the layer immediately lower to it.

The definition of the architecture worried with the integration of different data sources. To facilitate this integration, a reference format was specified. Whenever any data source is not in according to this format, the same is migrated for it.

The methodology of integrated and incremental data marts makes possible the integration of different data sources bit by bit. This characteristic facilitated the implementation of DW with S&T databases, considering the existence of different data sources. This also turns possible the inclusion of new data sources in the future.

In the development of the DW some problems have been found. The main problem was relative to the source databases. There are many laws that restrict the use of these databases.

As future works the following topics can be suggested:

- Incorporation of other data sources;
- Development of OLAP tools for search of knowledge in the DW;
- Definition of a results analysis method that verifies questions relative to the specific domain of S&T.

References

[1] B. Dinter, C. Sapia, G. Hofling, and M. Blaschka, "The Olap Market: State of the Art and Research Issues", In: *Proceedings of the ACM 1st International Workshop on Data Warehousing and Olap (DOLAP'98)*, Washington, United States of America, 1998, pp. 22-27.

[2] C. Häberli, and D. Tombros, "A Data Warehouse Architecture for MeteoSwiss: An Experience Report", In: *International Workshop on Design and Management of Data Warehouses (DMDW'01)*, Zurich, Switzerland, 2001.

[3] C. Shilakes, and J. Tylman, "Enterprise Information Portals. Enterprise Software Team", Retrieved from *http://www.sagemaker.com/company/downloads/eip/indepth.pdf*, May, 2003.

[4] D. Meyere, and C. Cannon, *Building a Better Data Warehouse*, Prentice Hall, Inc., Upper Saddle River, New Jersey, United States of America, 1998.

[5] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, New York, United States of America, 1999.

[6] I. Millet, D. H. Parente, and J. L. Fizel, "Data Warehouse Design for Ease of Data Transformation", In: *Proceedings of the 4th International Workshop on Design and Management of Data Warehouses*, Toronto, Canada, 2002, pp. 82-89.

[7] L. Indriunas, "Biopirataria - O Brasil se defende", In: *Revista Super Interessante*, Edição 193, Nº. 24, 2003.

[8] M. Axel, and Y. Song, "Data Warehouse Design for Pharmaceutical Drug Discovery Research", In: *Proceedings of the 8th International Conference and Workshop on Database and Expert Systems Applications (DEXA'97)*, Toulouse, France, 1997, pp. 644-650.

[9] R. Dumoulin, "Architecting Data Warehouses for Flexibility, Maintainability, and Performance", Retrieved from *http://www.olap.it/Articoli/architectingdwh.pdf*, October, 2005.

[10] R. Kimball, L. Reeves, M. Ross, and W. Thornthwaite, *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing, and Deploying Data Warehouses*, John Wiley & Sons Inc., New York, United States of America, 1998.

[11] W. H. Inmon, *Como Construir o Data Warehouse*, 2ª. Edição, Campus, Rio de Janeiro, Brasil, 1997.

[12] Y. Song, W. Rowen, C. Medsker and E. Ewen, "An Analysis of Many-to-Many Relatioships Between Fact and Dimension Tabels in Dimensional Modeling", In: *Proceedings of International Workshop on Design and Management of Data Warehouses (DMDW'01)*, Zurich, Switzerland, 2001.

Querying ontology based databases. The OntoQL proposal

Stéphane Jean, Yamine Aït-Ameur, Guy Pierra
LISI/ENSMA BP 40109, 86961 Futuroscope Cedex, France
E-mail: {jean,yamine,pierra}@ensma.fr

Abstract

Several approaches for storing ontologies and their instances in databases have been proposed. As a consequence, the need of defining languages to query such ontology based databases (OBDB) appeared. In this paper, we present OntoQL, an ontology query language designed for OBDB databases. This paper particulary emphasizes on the language constructs. Several query examples showing the interest of this language compared to traditional database query languages are given. Finally, we overview the proposed OntoQL implementation on a prototype of OBDB.

1 Introduction

Nowadays, ontologies are used in a lot of diverse research fields. They provide with the capability to represent a huge set of information contents. Several data models recommending the use of ontologies to describe data and their semantics have been developed. These models consider that a user or a system is able to retrieve the definition, meaning, translation and/or identifier of a given data item corresponding to a given data concept stored in an ontology.

Therefore, the idea of describing the ontology as well as the database model in a single model emerged in several work [18, 2, 16, 10, 5]. We call Ontology Based Database (OBDB) the database models allowing to store the ontology and the data in a common and single data model.

The goal of this paper is to describe OntoQL, an ontology query language for an OBDB model, called OntoDB, designed according to a database approach. In this paper, we present the basic constructs of this language and an overview of an implementation of this language on a prototype of OBDB.

This paper is structured as follows. Sections 2 gives a brief state of the art of OBDB models. Section 3 presents the OBDB model addressed in this paper. Section 4 introduces the OntoQL language by a set of queries on an example of ontology based database. Section 5 overviews the development conducted around the OntoQL language. Section 6 discusses related work. Section 7 concludes the paper by summarizing the main results and suggesting future work.

2 OBDB models: state of the art

Storing ontologies and their instances in databases has been the subject of several research studies and proposals. In the context of the Semantic Web, several OBDB models [2, 5, 16, 10] have been proposed to manage data described by ontologies represented in the standard ontology models RDFS [14] or OWL [3]. In these approaches, an instance, often called an individual, has its own property and class structure. Therefore, to manage instances, a generic database schema, not meaningful to a user and not customizable, is used. The simplest and more general one uses an unique table of triples [10] for storing both the ontology and its instances. Other approaches [2, 5] separate the representation of the ontology and its instances in two parts. The most common practice for storing instance data is to used the so-called vertical model [1] where information is stored in triples `(subject, property, value)` a variant of which, called the binary model is to have one table per property that contains only pairs of the form `(subject, value)`.

Our approach differs from the ones listed previously. A table is associated to each class of the ontology. The columns of these tables correspond to the properties of these classes. Before presenting the OntoQL query language exploiting this OBDB proposal for storing ontologies and their instances, let us describe the OntoDB OBDB model.

3 The OntoDB ontology based database model

The OntoDB model is composed of two main parts. The *ontology* part storing ontology definitions and the *content* part storing the instances whose descriptions and semantics are described by the stored ontologies. In order to illustrate the OntoQL proposal, a practical toy example is shared by all the sections of this paper.

3.1 The ontology part

The *ontology* part gathers the basic shared constructions of the PLIB [17], RDFS [14] and OWL [3] ontology mod-

els. Some of these constructions are issued from object oriented modelling.

- Each ontology concept has an unique identifier (*oid*);
- Each ontology concept is an instance of the *concept* entity of the ontology model.
- Each ontology concept is described by attributes values whose types may be either primitive types, association or collection types.
- The entities of the ontology model may be linked by an inheritance (multiple) relationship.

The *ontology* part may be described by the simplified UML class diagram of figure 1. This figure shows that an ontology contains a set of concept definitions gathered by classes and properties. An ontology defines a mechanism for uniquely identifying concepts (`name spaces`). Each class is described by a set of multi-lingual attributes (`name`, `definition` ...). Classes are linked by an inheritance relationship (`superClasses`). A set of properties may be used to characterize instances of classes (`properties`). Each property is defined in the scope of a class for which it is applicable (or of a superclass of this class) (`scope`).

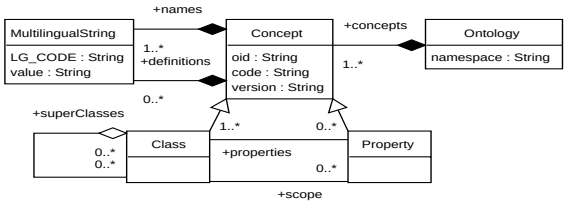


Figure 1. An UML model for the "ontology" part of an OBDB

Figure 1 represents the kernel of the ontology model addressed by the OntoQL language presented in this paper. This kernel can be extended by specific features related to any specific ontology model. For example, in the PLIB ontology model, classes are also described by *illustrations*, *external files*, *notes*, *remarks*, ... which are added to this kernel.

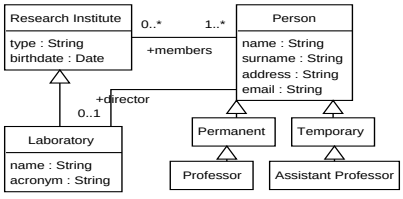


Figure 2. An example of ontology

In order to illustrate our model, let us consider a simple example describing data related to research institutions (figure 2). Each `research institute` is composed of several `members` (`permanent` or `temporary`). These persons have different statuses (`professor`, `assistant professor`, `associate professor`, ...). Among the research institutes, we distinguish `laboratories` that are characterized by an `acronym` and headed by a `director`.

3.2 The content part

The *content* part contains data and their logical schemas. However, contrary to classical databases, the logical schemas are linked to the ontology. Several works have studied this link [19]. In our approach, the logical schemas are built from a subset of the ontology. Therefore, different logical schemas can be derived from a single ontology.

A user determines which classes must be represented in the content part (extension of the class), and for each class the user determines which available properties (thanks to the scope attribute) are needed to describe the instances of a class. A table is derived to represent these instances. The link between this table and the ontology is automatically built and stored in the database.

Figure 3 presents a simple logical schema of a database that can be built from the defined ontology. In this example, the chosen properties are `name`, `director` and `members` for the `Laboratory` class and `name`, `surname` and `email` for the `Person` class. The property `members` being $n - n$, this schema contains the intermediate table `Laboratory_Members`.

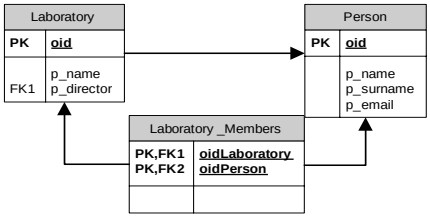


Figure 3. An example of logical database model for content part

3.3 Yet another database language ?

The answer to the question *Why another database language ?* is found in the specificities of the ontology based database model chosen to encode ontologies and their instances. Indeed, not all the properties of an ontology class are valuated in the content part. Therefore, it is necessary to store on the one hand the ontology in order to get the class and properties descriptions and on the other hand the logical schema (may be more than one logical schemas) storing the content and instances of classes and values of properties.

Having defined the ontology and their content, we are able to present the OntoQL language we have developed.

4 The OntoQL language

The OntoQL language is defined from the traditional components available in database exploitation languages. This section presents the language in a syntactical way and gives relevant queries on our example of ontology.

4.1 The data definition language

4.1.1 Definition of the ontology part

OntoQL provides with the resources to create, update and delete concepts of an ontology (classes, properties, ...) and of attribute values (names, definitions, ...). The following instruction creates a class with an English name `Laboratory`. It extends the class `Research Institute` and gives `definitions` and `names` in other languages (French and Spanish). Four properties are defined and created: the `name` of the laboratory, its `director`, its `acronym` and its `members`.

```
CREATE CLASS Laboratory  EXTENDS "Research Institute"(
  DESCRIPTOR (
    #name[fr,es] = ('Laboratoire','Laboratorio'),
    #definition = 'workplace for conducting
                                research activities',
    #definition[fr] = 'lieu pour mener des recherches')
  PROPERTIES (
    name String, director Person,
    acronym String, members SET OF Person));
```

Two specific clauses are introduced: **DESCRIPTOR** for introducing class attributes and descriptions (prefixed by #) and **PROPERTIES** to describe the relevant characterization properties of the concept. The valuations of these properties define the instances of concepts.

A property may be modified. The following clause changes the English `name` and adds an `illustration` to the `director` property.

```
ALTER PROPERTY director ADD DESCRIPTOR (
  #name[en] = 'headmaster',
  #illustration='headmaster.jpg');
```

4.1.2 Definition of the content part

The extent of a class is defined from the ontology by choosing which properties are valued for a given class.

```
CREATE EXTENT OF Laboratory (name, members, director);
```

This clause creates a container (table) of instances of the class `Laboratory` with the properties `name`, `members` and `director`. The link with the ontology and its concept definitions is also kept in the database.

4.2 The data manipulation language

When the extent of a class is defined, like in SQL3 [7], class instances can be inserted, deleted and updated. The following clause creates a new instance of the `Laboratory` class.

```
INSERT INTO Laboratory (name, acronym)
     VALUES ('Laboratoire d'Informatique Scientifique
                      et Industrielle', 'LISI');
```

The properties valued in an **INSERT** clause may be not described in the extent of a class (the `acronym` property in the previous clause). In this case, OntoQL offers three options: 1) either a *NULL* value is inserted or 2) an error is returned and the clause is rejected or 3) the extent of the class is completed by a new property and all the values of this property are completed with *NULL* values for the other instances.

To manipulate a property which type is another class, there is need to use nested clauses. For example,

```
UPDATE ontology_lisi:Laboratory
   SET director =
     (SELECT p FROM p in ontology_lisi:Person
      WHERE p.name='Dupont' and p.surname='Jack')
 WHERE acronym = 'LISI';
```

modifies the `director` of the `laboratory` which `acronym` is `LISI` by retrieving the `director` in a class `Person` of the ontology `ontology_lisi`.

4.3 The data query language

This section reviews the most important and novel aspects of the querying part of the OntoQL language. We focus on specific constructions that have emerged from the ontology based database model we have implemented. The language offers the possibility to query ontologies, contents (instances) and both ontology and content thanks to the implementation of the link between these two parts.

4.3.1 Querying the ontology

The ontology model of figure 1 contains some object oriented constructions. Therefore, and in order to keep aligned with relational object database languages, OntoQL proposes a syntax close to the one of languages like OQL [6] or SQL3[7].

Object oriented database constructions of OntoQL

Path expressions. OntoQL allows the use of *path expressions* in a **SELECT** clause. The following query is used to retrieve the identifier of the class domain of a property which identifier is `7B13543`.

```
SELECT #scope.#oid FROM #property WHERE #code='7B13543'
```

Manipulation of collections. To traverse collections, OntoQL expresses *dependent collections* in the **FROM** clause. Next query retrieves the code and version of classes defining a property named `acronym`.

```
SELECT c.#code, c.#version
  FROM c in #class, p in c.#properties
 WHERE p.#name = 'acronym'
```

In our example, the previous query returns the `code` and the `version` of the class `Laboratory` and its potential subclasses.

Moreover, queries may be *nested* in the clauses **SELECT**, **FROM** or **WHERE**. The following query returns the `name` and the `properties` of all classes which name starts by the letter `"A"`.

```
SELECT c.#name, (SELECT p.#name FROM p in #property
                 WHERE p.#scope=c)
  FROM c in #class
 WHERE c.#name like 'A%'
```

OntoQL provides with quantification operators **ALL** and **EXISTS** like in the clause

```
SELECT c.#name FROM c in #class
 WHERE '001' < ALL (SELECT p.#version
                    FROM p in c.#properties)
```

which returns the name of classes having all applicable properties with a version greater than `001`.

Finally, OntoQL is equipped with aggregate operators (**count, sum, avg, min, max**), access to the i-th element of an indexed collection, sorting (**ORDER BY**) and set operations (**UNION, INTERSECT, EXCEPT, GROUP BY**).

Specific OntoQL constructions.

Significant differences between the ontology model and object oriented models are available in the ontology model we defined. These differences led to specific constructions of the OntoQL language.

The multi-lingual aspect is one of these differences. Translations of the attributes may be defined. For example, the following query returns the names in French and in English of all the available classes.

```
SELECT #name[FR,EN] FROM #class
```

A second difference is the availability of internal (known by database implementors) and external (known by users, e.g., URI) identifiers and their names for all the concepts. Each of these identifiers may be used in an OntoQL query. For example, the names can be used to retrieve other elements of the ontology. The following query uses the name of a class to retrieve the French names of all its properties.

```
SELECT #name[FR] FROM "Research Institute".#properties
```

Notice that, for efficiency, the OntoQL engine uses the internal identifiers to run this query.

Moreover, name spaces may be used to refer to elements of an ontology. The **USING** clause is used for this purpose, it refers to a given name space outside the current one. The previous query, run on the `lisi_ontology` name space, can be written as:

```
SELECT p.#name[FR]
FROM ns_lisi:Laboratory.#properties
USING NAME_SPACE ns_lisi AS lisi_ontology
```

We have shown how OntoQL manages the *ontology* part and allows a user to retrieve the relevant descriptive information available in an ontology. Let us describe now, how *contents* may be queried.

4.3.2 Querying the content

OntoQL allows the query of contents. Moreover, since the content is linked to the ontology, querying the content does not rely on any specific logical database model. Therefore, two applications sharing a common ontology will have the right to run a common query even if the underlying logical database models are different.

Before giving the intuition of the content querying, let us recall some basic characteristics of instances.

- Each instance has an unique identifier (*oid*).
- Each instance has a basic class in the ontology.
- Each instance is described by the values of the properties defined in the extent of the class.
- Ontology classes may be linked by an inheritance relationship.

Querying the content will be similar to querying the ontology, except that that properties will not be prefixed by the # symbol.

Next query returns the `laboratory names` whose members do not have homonymous `name` with the `director` of this `laboratory`.

```
SELECT l.name
  FROM l in Laboratory
 WHERE l.director.name <> ALL (SELECT member.name
                               FROM member in l.members)
```

Notice that the same query written in SQL on the logical model presented on figure 3 is much more complicated. It requires to retrieve all the tables associated to the class `Laboratory`.

A polymorphic search operator * allows a query to retrieve the instances of a class and of all its subclasses. As illustrated below, the first query retrieves the names of instances which basic class is `Research Institute` while the second one returns the names for the `Laboratory` class as well.

```
SELECT name FROM "Research Institute"

SELECT nom FROM "Institut de Recherche"*
```

Notice that the second query of the previous example is written in French. This possibility is provided thanks to the capability of the ontology to support multilingual attribute names and concept language translations.

4.3.3 Querying both ontology and content

Since the links between ontologies and their contents are kept in the Ontology Based Database model, OntoQL has exploited this capability to allow querying both ontology and content in the same query.

From ontology to content

To query contents, OntoQL suggests an iterator `i` on instances of a class `C` by writing `i in C` or `C as i`. Next query returns the identifiers of instances belonging to the polymorphic extent of a class which French `name` begins with the string `"Per"`.

```
SELECT i.oid
  FROM C in #class, i in C*
 WHERE C.#name[fr] like 'Per%'
```

Moreover, it permits to retrieve and/or use the values of a property discovered by the query itself on the ontology part. The following query allows the retrieval of the values of the properties of the instances obtained in the previous query.

```
SELECT i.oid, p.#name[fr], i.p
  FROM C in #class, p in C.#properties, i in C*
 WHERE C.#name[fr] like 'Per%'
```

The previous query returns a tuple per property, but to obtain a single tuple per instance, one can write:

```
SELECT i.oid, (SELECT i.p, p.#name[fr]
               FROM p in C.#properties)
  FROM C in #class, i in C*
 WHERE C.#name[fr] like 'Per%'
```

From content to ontology

OntoQL proposes the use of the **`typeOf`** operator to make distinction between properties and retrieve information from the ontology part starting from the content part. This **`typeOf`** operator is implemented thanks to the link between ontology and content stored in the OBDB database. For example, the following query

```
SELECT i.name, i.surname, typeOf(i).#name[fr]
  FROM i in Person*
```

returns the French `name` of the basic class of the polymorphic instances of the class `Person`.

This kind of queries is particularly useful in a system where instances are originated from different sources and specializing a shared ontology. It allows the implementation of several automatic integration strategies [4].

5 The OntoQL toolkit

Around the OntoQL language, we have designed and implemented[1] the tools that we describe in this section.

OntoQL engine. The OntoQL engine implements the algebra, called *OntoAlgebra* [11], defining the semantics of the OntoQL language and the resulting optimizations on an OntoDB prototype.

OntoQL*Plus. OntoQL*Plus is a command line OntoQL language interface. It provides a syntax highlighting and a history of the executed commands.

[1] demonstrations are available at:
http://www.plib.ensma.fr/plib/demos/ontodb/index.html

OntoQBE. OntoQBE is a graphical OntoQL language interface. Figure 4 shows an interactively constructed OntoQL query. This interface allows to express object-oriented constructors of OntoQL such as path expressions (`TAG1`) or polymorphism (`TAG2`). Moreover, this interface presents the ontological definitions (illustration, code, version ...) of the concepts involved in the query (`TAG3`).

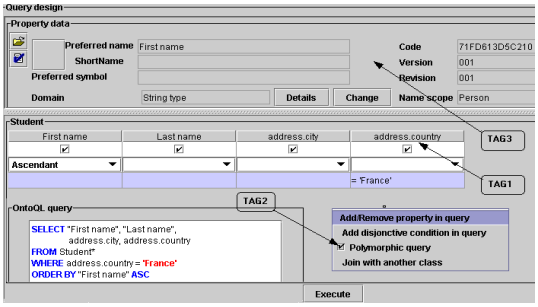


Figure 4. OntoQBE: A graphical OntoQL language interface.

OntoAPI. OntoAPI is a JAVA representation of the ontology model presented in section 3.1. This API (Application Programming Interface) allows to load classes and properties of an ontology from an OBDB without knowing the OntoQL language.

JOBDBC. JOBDBC API provides an access to an OBDB from the JAVA programming language. It extends the JDBC API providing methods to retrieve instances of OntoAPI interfaces as the result of a query and giving ontological definitions as metadata of a query.

6 Comparison with Related Work

OntoQL has been defined as an extension of SQL to exploit an OBDB model defined for semantic integration. With respect to this origin, related languages are multidatabase languages like SchemaSQL [13] or MSQL [15]. OntoQL shares with these languages the capability to express queries on data independently of their schemas. However, whereas these languages use the system catalog as an abstraction from database schemas, OntoQL uses ontologies. Consequently, OntoQL presents many differences with these languages such as its object-oriented nature or its independency of the model (relational, object-relational, object) used to represent the schemas of the data.

Query languages for the semantic web like RQL [12] or OWL-QL [8] have been defined on OBDB models presented in section 2. Like OntoQL, these languages offer the possibility to query ontologies, instances and both ontology and instances. However, in these approaches, an instance is an URI independently of its values of properties. It can belong to different classes not related by the subsumption

relationship. Consequently, OntoQL presents the following differences with these languages.

- *Object orientation*. The result of an OntoQL query searching instances of a class/entity is an object (internal identifier and state) and not an URI (external identifier).
- *Manipulation of schema*. OntoQL allows to create, alter and drop the schema of the data and offers the possibility to query data from these schemas (SQL upward compatibility).
- *Type checking*. OntoQL checks whether properties/attributes used in the **SELECT** or **WHERE** clauses are defined on some classes/entities in the current scope.
- *Usual syntax*. OntoQL provides usual constructors of traditional languages. For example, **SELECT** * can be used to retrieve the state of an instance.

Moreover, OntoQL presents the following features not yet provided by semantic web query languages [9] :

- *Exploitation of multi-lingual definitions*. An OntoQL query on content may be expressed in different natural languages. Moreover, attributes may have values for different natural languages.
- *Useful operators*. OntoQL provides grouping (**GROUP BY**), sorting (**ORDER BY**) and collection manipulation operators.

7 Conclusion

The contribution of this paper is two-fold. On the one hand, we have discussed and shown the differences existing between classical database models and ontology based database models. The need of new tools to manage the latter was a result of this study. On the other hand, we proposed a database language for managing the ontology based databases.

This paper showed a new language allowing the exploitation of ontology based databases. The need of such a language was motivated by the fact that both ontologies and their contents are stored in a database. This language is fully implemented on an OBDB prototype on top of PostgreSQL and actually runs on several applications. A QBE like interface is also available and a demo of the usage of this language can be found at:
`http://www.plib.ensma.fr/plib/demos/ontodb/`.

For the future it is planned to study the link existing between database based approaches for ontologies and the logic based approaches for ontologies. Our claim is that it is possible to offer an efficient storage capability for the instances described in the logic based approaches for ontologies like in OWL. As a benefit, we would be able to provide a database management to the instance together with a reasoning engine.

References

[1] R. Agrawal, A. Somani, and Y. Xu. Storage and querying of e-commerce data. In *VLDB*, pages 149–158. Morgan Kaufmann Publishers Inc., 2001.

[2] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, and K. Tolle. The ics-forth rdfsuite: Managing voluminous rdf description bases. In *SemWeb*, 2001.

[3] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, and P. F. P.-S. ad Lynn Andrea Stein. *OWL Web Ontology Language Reference*. World Wide Web Consortium, Feb. 2004.

[4] L. Bellatreche, G. Pierra, D. N. Xuan, H. Dehainsala, and Y. Aït-Ameur. An a priori approach for automatic integration of heterogeneous and autonomous databases. In *DEXA*, pages 475–485, 2004.

[5] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *SemWeb*, pages 54–68, 2002.

[6] R. G. G. Cattell. *The Object Database Standard: ODMG-93*. Morgan Kaufmann, 1993.

[7] A. Eisenberg and J. Melton. Sql: 1999, formerly known as sql 3. *SIGMOD Record*, 28(1):131–138, 1999.

[8] R. Fikes, P. J. Hayes, and I. Horrocks. Owl-ql - a language for deductive query answering on the semantic web. *Journal of Web Semantics*, 2(1):19–29, 2004.

[9] P. Haase, J. Broekstra, A. Eberhart, and R. Volz. A comparison of rdf query languages. In *SemWeb*, November 2004.

[10] S. Harris and N. Gibbins. 3store: Efficient bulk rdf storage. In *PSSS*, 2003.

[11] S. Jean, Y. Aït-Ameur, and G. Pierra. An object-oriented based algebra for ontologies and their instances. Technical report, LISI/ENSMA, january 2006.

[12] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Scholl. Rql: a declarative query language for rdf. In *WWW*, pages 592–603, 2002.

[13] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. Schemasql - a language for interoperability in relational multi-database systems. In *VLDB*, pages 239–250, 1996.

[14] O. Lassila and R. Swick. Resource description framework (rdf) model and syntax specification, 1999.

[15] W. Litwin, A. Abdellatif, A. Zeroual, B. Nicolas, and P. Vigier. Msql: A multidatabase language. *Inf. Sci.*, 49(1-3):59–101, 1989.

[16] Z. Pan and J. Heflin. Dldb: Extending relational databases to support semantic web queries. In *PSSS*, 2003.

[17] G. Pierra. Context-explication in conceptual ontologies: Plib ontologies and their use for industrial data. *Journal of Advanced Manufacturing Systems*, 2004.

[18] G. Pierra, H. Dehainsala, Y. Aït-Ameur, and L. Bellatreche. Base de données à base ontologique : principes et mise en œuvre. *Ingénierie des Systèmes d'Information*, 10(2):91–115, 2005.

[19] H. Wache, T. Vgele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, and S. Hübner. Ontology-based integration of information — a survey of existing approaches. In *IJCAI-01 Workshop: Ontologies and Information Sharing*, pages 108–117, 2001.

Towards a Conceptual Framework to Classify Ubiquitous Software Projects

Rodrigo O. Spínola Jobson L. M. da Silva Guilherme H. Travassos
{ros, jobson, ght}@cos.ufrj.br

COPPE / UFRJ – System Engineering and Computer Science Department
Caixa Postal: 68511 – CEP: 21945-970 – Rio de Janeiro – Brasil

Abstract

OBJECTIVE: To propose a conceptual framework that supports the identification of the level of adherence of software projects regarding ubiquitous computing.

METHOD: To run systematic reviews to characterize 1) ubiquitous computing, 2) its main features and 3) its specific requirements.

RESULTS: Fifty seven papers were analyzed regarding: (1) the ubiquitous computing definition, (2) ubiquitous applications, and, (3) 10 dimensions characterizing ubiquitous applications. To complement these results, other systematic review was accomplished. In this case, fifty nine papers were analyzed identifying a set of 123 functional and 45 non-functional requirements associated to the ubiquitous dimensions.

CONCLUSIONS: It is possible to have a common perspective to work with the ubiquitous software engineering area including the definition for ubiquitous computing and ubiquitous dimensions, and functional and non-functional requirements associated to each ubiquitous dimension. Therefore, a conceptual framework to classify software projects according to the adherence to the ubiquity perspective has been organized and discussed in this work.

I. INTRODUCTION

Many recent software projects have been characterized by a great number of requirements directly related to ubiquitous computing. These requirements usually are concerned with device independence, omnipresence, alternative interfaces that changes the usual computer screen and mouse paradigm, adaptation to the surrounding environment and so on. In a broader scenario, ubiquitous computing can be seen as a new information access paradigm where computers are embedded into the environment in such way that their use becomes natural [1].

This prospective scenario can represent new research challenges on the development of methodologies, software processes, testing approaches, and quality assurance techniques for the software engineers. That is what we have observed when dealing with some innovative software projects regarding e-science in Brazil, where one of the requirements explicitly mentioned the characteristic of ubiquity.

From that moment, we have identified the need of understanding the impact of the ubiquitous characteristic in the software project development planning. However, no information can be easily found regarding this new category of software.

This lack of knowledge raises some challenges on how to deal with the planning, designing, implementing, testing, and management of such kind of software project.

Therefore, the identification of the impact of ubiquitous characteristics on software projects can represent important information to reduce the risks concerned with the development of ubiquitous software projects. So, it is relevant to characterize the level of adherence of a software project according to a classification criteria supported by ubiquitous features.

Based on this perspective, software engineers should be able to define what should be the features to identify a software project as ubiquitous and how to classify a software project as ubiquitous according to these features.

As an initial tentative to address these issues, we propose a conceptual framework to classify software projects according to their adherence level regarding ubiquitous computing. We expect this characterization scheme will make software engineers able to identify those software project characteristics that should be impacted by these features of ubiquity.

To develop such approach, a fundamental task is related to the definition of ubiquitous computing and its main characteristics. This definition is particularly important because all the software projects should be classified according to the same perspective.

The methodology adopted to build the classification framework is based on the scientific method. In order to reach a solid scientific level, we decided to execute systematic reviews [2] rather than ad-hoc literature reviews. Its results tend to be more reliable because it makes use of a rigorous methodology that is susceptible to auditing and replication [3].

We have conducted two systematic reviews. The first one aimed at defining ubiquitous computing, identifying where it is currently being used and which are its main characteristics. The results described that, besides its definition, ubiquitous computing can be represented by 10 different dimensions. However, these dimensions are still described in so high abstraction level that was not possible to use them directly for classifying the software projects. Therefore, a second systematic review has been accomplished aiming at the identification of functional and non functional requirements associated to these ubiquitous dimensions. These requirements have been distributed among the ubiquitous dimensions, improving the classification criteria used by the conceptual framework. To exemplify, we have applied such framework to classify 8 different ubiquitous software applications, what can give some directions on how to use it for a larger population of software projects.

We have organized this paper in seven sections, including this Introduction. In section II, the definition and dimensions of ubiquity are presented. In section III, the results of a detailed analysis for each ubiquitous dimension are discussed. Next, in Section IV, we propose an approach to classify software projects considering their adherence level to ubiquity, with some results described in section V of applying such approach. In section VI, some identified open issues regarding ubiquitous computing and software engineering have been presented. Finally, in section VII we present the main contributions of this paper and future perspectives of this research project.

II. Ubiquitous Computing: Definition and Dimensions

Since the initial ideas presented by Weiser [1] and considering that ubiquitous computing is a broad term, software engineers have not worked enough to evolve its definition into a less ambiguous one through the definition of its intrinsic characteristics. In this context, this section presents the results obtained from the first systematic review execution. The goals of this review were to answer the following questions [4]: P0) What is ubiquitous computing?; P1) How ubiquitous computing is currently being presented? and; P2) What characteristics do define applications for ubiquitous computing?

To answer these questions, the following key words were considered: ubiquitous computing, pervasive computing, ubiquitous application, ubiquitous system, ubiquitous software, pervasive application, pervasive system, pervasive software, feature, requirement, characteristic, definition, characterization, and concept.

The papers sources considered during the study execution were: IEEE Portal and ACM Digital Library.

During this review, 751 scientific papers were identified and after its analysis, only 57 were selected to extract information. Within these 57 papers, eight describe concrete ubiquitous applications. The definition and dimensions of ubiquity obtained as result of the systematic review execution are presented below.

Ubiquitous computing definition: ubiquitous computing is present when computational services or facilities become available to the people in such a way that computer is no longer a visible or essential tool to access these services or facilities. That is, these services or facilities can materialize at any time or place, transparently, through the use of common devices. To make it happen it is necessary that software project take into consideration the dimensions presented below.

Ubiquitous systems definition: ubiquitous computing defines the ideal conditions where we can access computational resources in a ubiquitous way. On the other hand, ubiquitous system definition has a well-defined scope and it is strongly related to different characteristics that compose ubiquitous computing. It happens because, as ubiquity can be a property of a system, it can be achieved completely or partially. This variation is related to the fact that a particular system can implement or not the functions representing ubiquitous computing dimensions.

Dimensions: after the identification and analysis of the applications' characteristics [3], we could observe that many concepts explored by these applications have the same meaning but with different names or approaches. This way, the results obtained pointed out in the direction of a set of dimensions that can be interpreted as common ubiquitous applications dimensions. They are:

- Service omnipresence (SO): it allows users to move around with the sensation of carrying computing services with them;
- Invisibility (IN): ability of being present in objects of daily use, weakening, from user's point of view, the sensation of explicit use of a computer and enhancing the perception that objects or devices provide services or some kind of "intelligence";
- Context sensitivity (CS): ability to collect information from the environment where it is being used;
- Adaptable behavior (AB): ability of, dynamically, to adapt offered services according to the environment where it is being used, respecting its limitations;
- Experience capture (EC): ability of capturing and registering experiences for later use;
- Service discovery (SD): pro-active discovery of services according to the environment where it is being used;
- Function composition (FC): ability of, based on basic services, to create a service required by the user;
- Spontaneous interoperability (SI): ability to change partners during its operation and according to its movement;
- Heterogeneity of devices (HD): provides application mobility among heterogeneous devices;
- Fault tolerance (FT): ability to adapt itself when facing environment's faults.

The characteristics identified are not restricted to the list above. Other examples are: environment monitoring, application reconfiguration, and devices discovery. However, they do not define dimensions by themselves, but are associated with one or more dimensions listed above.

Although it is possible to capture a clear-cut set of dimensions, these dimensions still represent a high level of abstraction. This way, we need to go down into a set of more concrete requirements associated with each dimension. That is the goal of the secondary study presented in the next section.

III. Functional and non-Functional Characteristics of the Ubiquitous Dimensions

This section presents the results obtained with the execution of the second systematic review. The goal of this review was to answer the question [4]: what are the functional and non-functional requirements that characterize each ubiquitous dimension?

To answer this question, the following key words were considered: ubiquitous computing, pervasive computing, functional requirement, functionality, feature, characteristic, non-functional requirement, quality requirement, and the dimensions of the ubiquity.

For this systematic review, it has been decided to enlarge the search and the papers sources considered during the study execution were: IEEE Portal, ACM Digital, INSPEC, and EI COMPENDEX.

During this review, 599 scientific technical papers were identified and after its analysis, only 59 were selected for information extraction. It was possible to identify 168 (123 functional and 45 non-functional) requirements.

Moreover, it was possible to group most of the requirements according to its characteristics, associating each formed group to one and only one dimension instead of having each requirement individually. This grouping made easier the analysis process due to the great number of requirements found by the systematic review.

The analysis was made up in three steps: 1) identifying the presence of the ubiquitous dimension; 2) identifying the requirements of each dimension, and; 3) grouping corresponding requirements in each dimension.

The results of the first and second steps are presented in Table 1. The first column shows the dimensions obtained from the first systematic review. The second and third columns show how many studies are concerned with each dimension, in absolute values and percentage. The fourth and fifth columns show how many requirements were found for each dimension. Finally, the sixth column shows the percentage distribution of requirements per dimension.

According to the third step activity, an example of a requirements group for the context sensitive dimension is presented below:

Dimension: Context sensitive
  Group: Information capture
    Requirement: To capture the user identity, location, effect or activity.
    Requirement: To consider the time variable.

TABLE 1
DIMENSIONS PRESENCE AND REQUIREMENTS

Ubiquitous Dimension	Presence	% of 59	Functional	Non-Functional	% of 168
SO	28	47,5	9	1	6,0
IN	26	44,0	8	2	6,0
CS	56	94,9	22	8	17,9
AB	52	88,1	24	8	19,0
EC	11	18,6	7	0	4,2
SD	28	47,5	13	13	15,5
FC	19	32,2	18	5	13,7
SI	21	35,6	10	2	7,1
HD	18	30,5	9	3	7,1
FT	11	18,6	3	3	3,6
Total of requirements			**123**	**45**	

All 59 selected projects were fully qualified using the 10 proposed dimensions. From Table 1 it is possible to notice that for all ubiquitous dimensions, except service discovery and fault tolerance, the focus is concentrated on exploring functional requirements. Besides, the third and sixth columns allowed extracting the data to produce the graph on Fig. 1, representing the distribution of requirements per dimension and the presence of each dimension regarding the selected papers. It is possible to observe that the curves present similar trends. It seems to indicate that the: 1) ubiquitous dimensions make sense; 2) requirements distribution is fair, and; 3) relative importance of each dimension in the current ubiquitous computing scenario does exist.

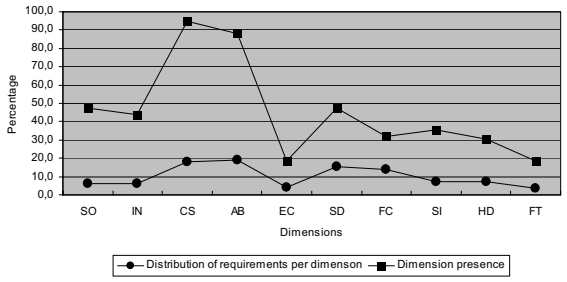


Fig. 1. Distribution of requirements and dimensions presence

IV. APPROACH TO CLASSIFY APPLICATIONS ACCORDING TO PERSPECTIVE OF UBIQUITY

Ubiquitous computing comes up when its dimensions are fully implemented in software projects. Thus, to be considered totally ubiquitous, a software project has to contemplate the different requirements of each ubiquitous dimension. However, we can have software projects with different levels of adherence to the ubiquitous dimensions.

From the concepts described in section III, it was elaborated a classification approach comprised of three steps to: 1) verify the presence of the functional and non-functional requirements of each dimension; 2) consolidate the software project adherence level of each dimension based on the presence/absence of each functional and non-functional requirement, and; 3) generate the graph that will represent the application adherence level considering the ubiquitous dimensions using the values calculated on the previous step.

To automate steps 2 and 3 it has been elaborated a spreadsheet based form to evaluate the adherence level results. Further details about this form can be found at the URL http://www.cos.ufrj.br/~ros/ubforms.html.

Fig. 2 represents the reached result when applying this approach to ubiquitous application presented in [5].

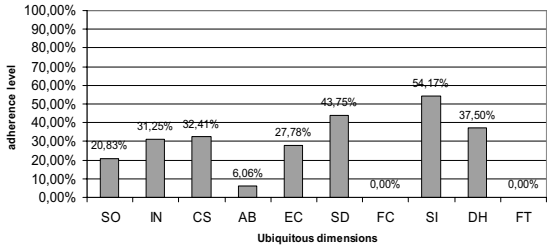


Fig. 2. Example of dimensions and their adherence levels

In the next section will be discussed the results of applying this conceptual framework to classify the 8 projects found by the first systematic review (section II).

V. USING THE PROPOSED CLASSIFICATION APPROACH

In this section, the proposed classification approach has been applied on a set of eight applications [5, 6, 7, 8, 9, 10, 11, 12] characterized as ubiquitous by the first systematic review [4]. These applications were found only in the first systematic review. For each one of the software projects, an evaluation like that exemplified in the previous section (Fig. 2) has been made. For the sake of space, just the general results will be presented. Further details can be found at the URL http://www.cos.ufrj.br/~ros/ubresults.html.

The accomplishment of the first step can be visualized on Table 4. Based on these data, it was generated the graph on Fig. 3. It represents the ubiquitous adherence level of each investigated application. It shows the number of attended requirements per application and presents the number of requirements for each dimension.

Analyzing these results and considering the set of applications selected in the systematic review, we can observe that exist more concern with the invisibility, context sensitive and adaptable behavior dimensions. The other dimensions appear just as isolated initiatives.

Analyzing the graph on Fig. 3, it is possible to observe an expected trend: if the number of requirements identified for each dimension increases or decreases (dashed horizontal lines), this behavior is followed by the number of requirements implemented in the applications (vertical bars).

The exception to this behavior is the function composition dimension. This result was not expected by the fact that this dimension was considered necessary. However, this could be explained by the difficulty to deal with the inherent complexity of the function composition dimension requirements.

As a final result, we can say that this behavior resembles that one described at the end of Section III (concerned with the trends observed on Fig. 1).

VI. OPEN ISSUES

The obtained results with the execution of the classification approach presented in Section V allow us to create a baseline that reflects some trends on ubiquitous software project construction. This is important because it makes possible to perceive how far the states of practice and art could be and brings some important insights related to the software development process and product. In this context, several questions arise: 1) what are the risks associated with each ubiquitous dimension and its respective requirements?; 2) what testing approaches may be applied according to the identified dimensions?; 3) what activities should be accomplished on the software development process to deal with each dimension?; 4) what quality characteristics software engineers should have

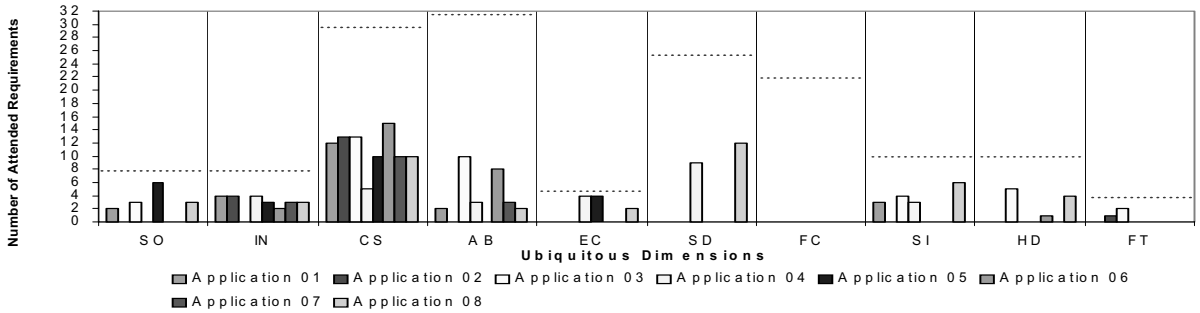


Fig. 3. Attended Requirements

in mind for each dimension?; 5) do developers need specific software engineering techniques to deal with these dimensions? If so, which are they?, and; 6) what is the influence of each dimension and its respective requirements on the design and architecture of the software?

TABLE 4
UBIQUITOUS REQUIREMENTS PER APPLICATION

Ubiquitous Dimension	Number of Identified Requirements	Attended Requirements per Application							
		A1	A2	A3	A4	A5	A6	A7	A8
SO	10	2	0	3	0	6	0	0	3
IN	10	4	4	0	4	3	2	3	3
CS	30	12	13	13	5	10	15	10	10
AB	32	2	0	10	3	0	8	3	2
EC	7	0	0	0	4	4	0	0	2
SD	26	0	0	0	9	0	0	0	12
FC	23	0	0	0	0	0	0	0	0
SI	12	3	0	4	3	0	0	0	6
HD	12	0	0	5	0	0	1	0	4
FT	6	0	1	2	0	0	0	0	0

VII. FINAL CONSIDERATIONS

From the identification of opened issues, this conceptual framework brings together some contributions: 1) a definition of ubiquitous computing and ubiquitous systems; 2) a proposal for the necessary dimensions to achieve ubiquitous computing; 3) identification of functional and non-functional requirements for each ubiquitous dimension, and; 4) proposal of a classification approach of ubiquitous software projects considering the ubiquitous dimension adherence level.

From the point of view of ubiquitous software projects, the selected technical papers on the first systematic review allowed us to conclude that ubiquitous software projects are too much restricted on their scope; they are still limited to research centers, and; they present solutions that take into consideration just a small number of ubiquitous dimensions and its respectively requirements.

It is important to observe that the dimensions and the classification approach represent just starting points to new research activities regarding ubiquitous computing. Future works should explore issues like: 1) establishing the dependencies and complementarities among functional and non-functional requirements of the different dimensions; 2) improving the adherence level evaluation by setting different weights for each requirements group; 3) improving the conceptual framework by setting different weights for each ubiquitous dimension according to the application domains; 4) evaluating the impact of each ubiquitous dimension and its requirements on the software architecture and design, and; 5) investigating new software engineering areas to support the construction of ubiquitous software projects.

This way, we believe we would be supporting the building of a body of knowledge concerned with the development of ubiquitous software projects.

ACKNOWLEDGMENTS

The authors would like to thank CAPES and CNPq for the financial support to this work. This work has been partially developed in collaboration with HP Brazil R&D.

REFERENCES

[1] Weiser M., 1991. The Computer for the 21st Century. Scientific American, pp. 94-104.

[2] Kitchenham, B., 2004. Procedures for Performing Systematic Reviews. Technical report, Keele University, Australia.

[3] Biolchini, J., Mian, P.G., Natali, A.C.C., Travassos, G.H., 2005. Systematic Review in Software Engineering. Technical Report ES 679/05. COPPE/UFRJ.

[4] Spínola, R.O., Silva, J.L.M., Travassos, G.H., 2006. Towards a Conceptual Framework to Classify Ubiquitous Software Projects: Further Details. Submitted to the Brazilian Symposiums on Software Engineering.

[5] Tahti, M., Rauto, V., Arhippainen, L., 2004. Utilizing context-awareness in office-type working life. Proceedings of the 3rd international conference on Mobile and ubiquitous multimedia.

[6] Kindberg, T., Barton, J., Becker, G., Caswell, D., Debaty, P., Gopal, G., Frig, M., Krishnan, V., Morris, H., Schettino, J., Serra, B., Spasojevic, M., 2000. People, places, things: Web presence for the real world. Third IEEE Workshop on Mobile Computing Systems and Applications, pp.: 19 – 28.

[7] Ali, J.A., Won-Sik, Y., Jai-Hoon, K., We-Duke, C., 2004. U-kitchen: application scenario. WSTFEUS'04, pp.: 169 – 171.

[8] Bossen, C., Jorgensen, J.B., 2004. Context-descriptive prototypes and their application to medicine administration. DIS2004. Pages: 297 – 306.

[9] Hatala, M., Wakkary, R., Kalantari, L., 2005. Rules and Ontologies in Support of Real-time Ubiquitous Application. Journal of Web Semantics.

[10] Lee, S.H., Chung, T.C., 2004. System Architecture for Context-Aware Home Application. WSTFEUS 2004.

[11] Zhou, P., Nadeem, T., Kang, P., Borcea, C., Iftode, L., 2005. EZCab: A Cab Booking Application Using Short-Range Wireless Communication. PerCom 2005: 27-38.

[12] Joel, S., Arnott, J.L., Hine, N.A., Ingvarsson, H., Rentoul, R., Schofield, S., 2004. A framework for analyzing interactivity in a remote access field exploration system. SMC(3) 2004: 2669-2674.

Open Source Development Process: a Review

Marco Scotto
Free University of Bolzano
Piazza Domenicani, 3
I-39100 Bolzano-Bozen
Marco.Scotto@unibz.it

Alberto Sillitti
Free University of Bolzano
Piazza Domenicani, 3
I-39100 Bolzano-Bozen
Alberto.Sillitti@unibz.it

Giancarlo Succi
Free University of Bolzano
Piazza Domenicani, 3
I-39100 Bolzano-Bozen
Giancarlo.Succi@unibz.it

Abstract

This paper presents a review of the Open Source development process from the point of view of the involvement of the developers in the production process. The study focuses on how developers contribute to projects in terms of involvement, size and kind of their contribution. Data have been collected from different kinds of open source projects and the results show that the general behavior is the same, even if projects and people are different.

Keywords
Open Source Software, Development Process, Empirical Studies

1. Introduction

Nowadays, there is an increasing interest in Open Source Software (OSS), not only from the point of view of the final user, but also from the point of view of the development process. The success of some Open Source projects has promoted the interest regarding how OSS is developed and what are the approaches used. However, there are a few evidences regarding the quality and the effectiveness of the open source development process and this lack of knowledge is creating several myths [6, 7, 13].

The aim of this paper is to investigate the open source development through the comparison of different projects in terms of involvement of developers and code produced. The projects considered in the study are medium or large products developed by a team with, at least, 10 developers. We have excluded from the study small projects carried out by single developers or very small teams. Even if the considered projects are heterogeneous, the developers and the projects themselves behave in very similar ways. Even if every open source project is unique in terms of rules, people, and management, the behavior of developers do not change too much across different projects. There is a common approach to software development going beyond rules, people, and management.

This paper is organized as follows: section 2 introduces the background of the study; section 3 describes the sample considered in the study; section 4 presents the analysis; section 5 summarizes the results; finally, section 6 draws the conclusions and presents future work.

2. Background

The open source community includes very different actors, such as single developers, universities, non-profit organizations (e.g., Free Software Foundation, Apache Software Foundation, Debian, etc.), and commercial organizations (e.g., IBM, Sun Microsystems, Novell, RedHat, etc.) [4, 15]. Due to this heterogeneity, contributions to open source projects come from two kinds of people: volunteers and corporate developers.

Volunteers are developers that spend part of their spare time writing code for the Community. For these people, developing open source code in not the main activity, it is a hobby [16]. On the other hand, corporate developers are regular employee of companies promoting or participating to open source projects. In this case, developing open source software is the main activity for which they are paid for. Usually, only corporate developers follow a process. Therefore, only a subset of the total contributions comes from a structured development process. Since contributions come from developers working in different environments and without any superimposed process, we expect that development patters vary across projects. However, successful projects are not based on unstructured development: there is a small set of rules that every developer has to follow. Such rules include a restricted access to the source code repositories, only some developers can actually change the code [12]. The others have to send their modifications to the authorized developers that evaluate the contributions and accept or reject them.

Usually, such loose control of the process includes only coding standards for names and formats. Therefore, there are no constraints for the personal coding style and

behavior of the developers. Such unstructured process should generate software with undefined quality. As instance, developers are not forced to fix bugs before adding new functionalities; they can implement nearly every kind of feature they like without considering the needs of the users and their priorities, etc. All these behaviors are also dependant on the project and the developer. Every project can define its own policy and, inside a single project, developers can behave differently, since their freedom is usually very high. These heterogeneous approaches are likely to cause the production of software with undefined level of quality. However, this is the approach adopted in nearly all the open source projects.

Most of the open source projects are just toy projects [2, 15]. Such projects are usually small, relay on a couple of very good developers, and become obsolete very fast. Usually, these projects are started to solve a specific problem for which no tools are available. After the problem is solved, the developers are rarely interested in maintaining the tool, but they allow anyone to use it and modify it. In this way, useful tools become outdated soon, if there are no developers interested in the project with the skills required to carry on the development. However, this does not happen for all the projects. Among such toy projects there are a few that are more ambitious and are able to attract a larger number of developers for several reasons [15]. In this case, there are a quite small number of core developers and a larger number of contributors [12]. The core developers are the architects of the project, provide the majority of the code, and decide how the project evolves. For these reasons, the number of the core developers is critical. If such group is too small, the project is highly dependant on the gurus that started the project. On the contrary, if the group is large enough, the project is able to survive and evolve even if the developers that started it decide to leave. In this last case, the number of core developers is enough to carry on the development of the project keeping it up-to-date. According to these considerations, open source software is affected by a set of critiques regarding its undefined process and quality; dependence on gurus; lack of updates; and many others.

The aim of this research is to verify whether some of these critiques are valid or not.

3. The sample

The study has been carried out through the analysis of 13 source code repositories listed in Table 1. Since the tool used is able to analyze only C/C++ and Java code, all the code written with different languages has been ignored. Some of the projects considered include a small amount of code written with different programming languages (i.e. Ant scripts), however, we think that such contributions are small and do not affect the results of the analysis since the amount of code is less than 5% of the entire project. In this preliminary study we are not considering the potential effects of such extra code that should be investigated in the future.

Table 1: Descriptive statistics of the product under investigations

Product	Files	People	Commits	ΔLOC	ΔC
Apache 1.2	92	16	2,031	11,216	866
Apache 1.3	222	50	8,969	40,252	3,210
Apache 2.0	298	52	15,192	21,748	3,105
Tomcat 3.0	349	34	3,405	8,954	1,467
Tomcat 4.0	799	31	4,745	3,283	2,769
PostgreSQL	177	13	1,205	12,640	748
KDE	2,670	379	76,273	343,495	22,995
XFree86 3.0	971	17	2,468	53,663	3,819
XFree86 4.0	5,279	19	12,518	229,378	24,723
OpenBSD	303	66	3,266	4,759	689
Xerces	1,938	34	16,639	23,516	4,795
Xalan	1,905	31	19,288	33,795	4,577
JBoss	551	87	8,827	66,124	4,153

Table 1 shows basic characteristics of the projects included in the study:

- The number of source code files analyzed (C/C++ and Java files only excluding any other languages), including C/C++ headers.
- The number of unique contributors.
- The number of commits. This is the number of atomic modifications made to the source code at file level.
- **ΔLOC:** number of lines of code (LOC) added since the beginning of the project. It is calculated as follows:

  $$\sum_{i=1}^{N} (\Delta LOC)_i$$

  where $(\Delta LOC)_i = LOC_{i,end} - LOC_{i,start}$ and N is the number of files of the project.
- **ΔC:** cyclomatic complexity (C) [11] added since the beginning of the project, calculated as follows:

  $$\sum_{i=1}^{N} (\Delta C)_i$$

  where $(\Delta C)_i = C_{i,end} - C_{i,start}$ and N is the number of files of the project.

Table 1 does not include the absolute values of LOC and C. This is because some projects started from an

empty repository (e.g., Xalan and Xerces), while others started importing a previous version (e.g., Apache and Tomcat). This is not a problem for the analysis, since the study focus on the evolution of the source code, therefore only the variations are considered and showed in the table.

Table 2 summarizes the contributions to the projects by different developers and points out that the Pareto rule is valid in most of the cases. A small number of the developers (about 20%) provide most of the contributions (about 80%). This is true from the point of view of both the total number of commits and the total number of lines of code (added, removed, or modified).

Table 2: Number of contributions

Product	People contributing to 80% of the commits	People contributing to 80% of the LOC
Apache 1.2	8 (50%)	8 (50%)
Apache 1.3	11 (22%)	11 (22%)
Apache 2.0	12 (23%)	10 (19%)
Tomcat 3.0	5 (15%)	4 (12%)
Tomcat 4.0	6 (19%)	5 (16%)
PostgreSQL	2 (15%)	1 (8%)
KDE	39 (10%)	38 (10%)
XFree86 3.0	1 (6%)	1 (6%)
XFree86 4.0	1 (5%)	1 (5%)
OpenBSD	8 (12%)	5 (8%)
Xerces	9 (26%)	9 (26%)
Xalan	9 (29%)	8 (26%)
JBoss	9 (10%)	10 (11%)

In some cases (e.g., XFree86 3.0 and 4.0), most of the commits are made using a shared login in the CVS system. For this reason, it is not possible to identify the single developers but there is only a virtual main contributor who has provided more than 95% of the code. Therefore, in such cases the Pareto rule cannot be verified.

According to Table 2, only about 20% of the developers are the people who drive the development. This means that if the team is small the development is guided by a 2-3 gurus and the project is highly dependant on them. On the other hand, if the project is large enough, the dependency on gurus is reduced and the project is likely to survive even if some of the main developers decide to leave. Based on several studies on code evolution [1, 3, 8, 9, 10] and on refactoring [5], we have developed the classification showed in Table 3.

Moreover, for each type of modification, we identify whether they are additions, removals, or modifications of code.

Table 3: Classification of the modifications

Type	Code identifier
Structural	Any changes to the source code that affect the structure of the product. It includes the following: class and function definitions, decision statements, and loops.
Non-structural	Any changes to the source code that do not affect the structure of the product. It includes the following: variable definitions, inclusions, assignments, name change, and any other statement not included in the structural definition.
Comment	Any changes to comments.

The sample includes 13 products: 6 written in C/C++, 5 in Java, and 2 in C/C++ and Java, with a number of files between 92 (Apache 1.2) and 5,279 (XFree86 4.0). The considered products belong to very different application domains: web and application servers, database, operating system, and windows manager. Major versions of the projects Apache, Tomcat, and XFree86 are considered as separate projects, since they are very long lasting projects and stored in different repositories.

4. Data Analysis

The analysis of the data is made through the different techniques described here below.

4.1 Contribution analysis

To identify patterns in the behavior of the developers, a time series analysis technique has been used. A sequence analysis identifies a set of phases of a model and evaluates their evolution in time. This kind of analysis is based on categorical data and its aim is to identify patterns. The considered technique is a sequence analysis that comes from the social sciences called gamma analysis [14]. We classify source code modifications into three categories (Table 3). Hence, the phases considered for the analysis are listed in Table 4.

Table 4: Phases for the gamma analysis

Phase	Structural modifications	Non-structural modifications	Comment modifications
000	0	0	0
100	$\neq 0$	0	0
010	0	$\neq 0$	0
001	0	0	$\neq 0$
110	$\neq 0$	$\neq 0$	0
101	$\neq 0$	0	$\neq 0$
011	0	$\neq 0$	$\neq 0$
111	$\neq 0$	$\neq 0$	$\neq 0$

The gamma analysis is able to describe the order of the phases in a sequence and provide a measure of the overlapping of the phases. It is based on the *gamma score*, which is a non-parametric statistic based on the ordinal relationship of Goodman and Kruskal [14] defined as follows:

$$\gamma_{(A,B)} = \frac{P - Q}{P + Q}$$

where P is the number of A-phases preceding the B-phases and Q is the number of A-phases following the B-phases (A and B phases represent two symbols or sequences of symbols). The γ calculated in this way is symmetric and varies between -1 and +1. Its meaning is summarized in Table 5.

Table 5: Meaning of γ

	Meaning
$\gamma_{(A,B)} < 0$	A-phases follow B-phases
$\gamma_{(A,B)} = 0$	A-phases and B-phases are independent
$\gamma_{(A,B)} > 0$	A-phases precede B-phases

The *gamma score* is used to calculate the *precedence score* and the *separation score*. The *precedence score* measures how frequently one symbol precede the other. It is positive if A-phases precede B-phases, negative if B-phases precede A-phases and zero if the phases are mixed. The *separation score* measures how much separated the phases are. It is positive if it is more frequent that A-phases precede or follow B-phases and zero if the phases are mixed.

The *precedence score* is the mean of the gamma scores:

$$\gamma_A = \frac{1}{N} \sum_i \gamma_{(A,i)}$$

where N is the number of phases and $\gamma_{(A,i)}$ is the gamma score calculated between the phases A and i. The precedence score varies between -1 and +1. The *separation score* is the mean of the absolute value of the gamma scores:

$$s_A = \frac{1}{N} \sum_i \left| \gamma_{(A,i)} \right|$$

It varies between 0 and +1. A separation score of 0 means that the phases are independent, while +1 means that there is a separation among the phases. The values of γ and s are calculated for each file in the project. Then, their values for the whole project are calculated as a weighted average as follows:

$$\gamma = \frac{\sum_i v_i \gamma_i}{\sum_i v_i} \qquad s = \frac{\sum_i v_i s_i}{\sum_i v_i}$$

where γ_i, s_i, v_i are the precedence score, the separation score, and the number of versions of the file i.

Table 6 evidences that most of the modifications made to the source code (more than 80%) belong to the phases: 000, 010, 110, 111.

Table 6: Main modifications

Product	**000, 010, 110, 111**	**Other**
Apache 1.2	86% (24%, 27%, 21%, 14%)	14%
Apache 1.3	87% (25%, 31%, 18%, 13%)	13%
Apache 2.0	89% (25%, 30%, 20%, 14%)	11%
Tomcat 3.0	86% (9%, 31%, 21%, 25%)	14%
Tomcat 4.0	87% (11%, 31%, 21%, 24%)	13%
PostgreSQL	94% (25%, 25%, 31%, 13%)	6%
KDE	83% (16%, 24%, 26% 17%)	17%
XFree86 3.0	80% (19%, 20%, 16%, 25%)	20%
XFree86 4.0	80% (28%, 13%, 18%, 21%)	20%
OpenBSD	88% (8%, 37%, 32%, 11%)	12%
Xerces	85% (19%, 31%, 18%, 17%)	15%
Xalan	87% (27%, 28%, 14%, 18%)	13%
JBoss	88% (10%, 19%, 29%, 30%)	12%

This means that most of the contributions are:

- Cosmetic changes of the code and refactoring without affecting structural and non-structural instructions
- Non-structural modifications
- Structural and non-structural modifications at the same time
- Structural, non-structural, and comment modifications at the same time

All the other kinds of modifications are less than 20% of all the contributions. Therefore, these are the phases that mainly affect how software development is carried out. The precedence scores calculated for all the projects varies in the whole range of values, but most of the values are in the range [-0.2, 0.2]. Only 14% of the values are not in this range and only 7% is not in the range [-0.25, 0.25]. Therefore, it is not possible to identify a precedence pattern in such data. On the other hand, the behavior of the separation scores is more meaningful. Considering the main phases identified in Table 6, 86% of the values are above 0.30. This means that each kind of these contributions tend to occur before or after all the

others. Therefore, when developers write these kinds of contributions focus on them and tend to implement them separately from all the others. This implementation pattern is common in all the projects considered in the study.

4.2 Correlation analysis

The separation scores of the most relevant kind of contributions identified in section 4.1 are negatively correlated with the number of versions of the files belonging to the projects (Table 7). All the values are significant at the 0.01 level.

Table 7: Spearman correlation coefficient among number of versions and separation scores

Product	N, 000	N, 010	N, 110	N, 111
Apache 1.2	-0.631	-0.541	-0.502	-0.636
Apache 1.3	-0.695	-0.620	-0.719	-0.649
Apache 2.0	-0.314	-0.363	-0.593	-0.408
Tomcat 3.0	-0.524	-0.731	-0.688	-0.608
Tomcat 4.0	-0.613	-0.554	-0.702	-0.689
PostgreSQL	-0.511	-0.505	-0.642	-0.622
KDE	-0.662	-0.686	-0.663	-0.636
XFree86 3.0	-0.422	-0.523	-0.475	-0.512
XFree86 4.0	-0.475	-0.481	-0.448	-0.482
OpenBSD	-0.586	-0.462	-0.458	-0.368
Xerces	-0.625	-0.633	-0.668	-0.636
Xalan	-0.715	-0.741	-0.580	-0.666
JBoss	-0.463	-0.541	-0.599	-0.609

This means that in every project the separation among phases decreases with the increasing of the number of versions of the files of the project. On the other hand, there are no significant correlations among the evolution of the number of lines of code, the McCabe Cyclomatic complexity, the number of modifications (structural, non-structural, and comment), and the separations scores.

This means that nearly in all the projects the separations scores are independent from:

- the number of lines of code modified;
- the McCabe cyclomatic complexity modified;
- the number of modifications (structural, non-structural, and comment) introduced.

Therefore, the pattern of the modifications is independent from the amount of code modified.

5. Results

According to the data collected, even if the 13 projects considered are very different, their behavior is very similar. In nearly all the cases, there is a set of core developers (about 20% of the total) that develop most of the code (about 80%). This means that the final product is highly dependant on such a small group and on its quality. In many cases, such group includes 8 - 12 people, not just a couple. Therefore, the project is not dependant on a couple of gurus but on a team. According to the classification of the modifications presented, about 80% of the modifications in a project belong to a subset of the possible modifications. In particular:

- Cosmetic changes of the code and refactoring without affecting structural and non-structural instructions
- Non-structural modifications
- Structural and non-structural modifications at the same time
- Structural, non-structural, and comment modifications at the same time

It is interesting that the set and the percentage are the same in all the projects, even if their application domains are very different (operating systems, web and application servers, etc.).

Moreover, the development pattern is the same. Modifications are not performed in an ordered way but developers tend not to mix different kinds of modifications. They focus on a modification per time. As expected, this separation is negatively correlated with the number of versions of a file. This could mean that if there are many changes, the developers have to implement many kinds of modifications at the same time and they cannot focus on a single modification per time.

However, the patterns of modifications are independent from the number of lines of code modified, from the algorithmic complexity of the code, and from the number of modifications introduced.

6. Conclusions and future work

This paper has presented an empirical review of the open source software development through the analysis of 13 projects belonging to very different application domains. The results show that even if the projects are very different and developed by different people; the average behavior is the same in all the cases. The validity of the study is limited due to the small number of projects considered; therefore we will continue to analyze further Open Source repositories in order to verify whether they behave in the way.

7. References

.

[1] Baxter I.D., Pidgeon C., Mehlich M., "DMS: Program Transformations for Practical Scalable Software Evolution", *26th International Conference on Software Engineering*, Edinburgh, UK, May 2004.

[2] Cusumano M.A., “Reflections on Free an Open Software”, *Communications of the ACM*, Vol. 47, No. 10, pp. 25-27, October 2004.

[3] Eisenbarth T., Koschke R., Simon D., “Aiding program comprehension by static and dynamic feature analysis”, *IEEE International Conference on Software Maintenance*, Stuttgart, Germany, November 2001.

[4] Feller J., Fitzgerald B., *Understanding Open Source Software Development*, Addison-Wesley, 2002.

[5] Fowler M., Beck K., Brant J., Opdyke W., Roberts D., *Refactoring: Improving the Desing of Existing Code*, Addison-Wesley, 1999.

[6] Fuggetta A., “Open source software – an evaluation”, *Journal of Systems and Software*, Vol. 66, No. 1, April 2003.

[7] Healy K., Schussman A., “The Ecology of Open-Source Software Development”, 2003. Available online at: http://opensource.mit.edu/papers/healyschussman.pdf

[8] Klint P., “How Understanding and Restructuring Differ from Compiling - A Rewriting Perspective”, *11th IEEE International Workshop on Program Comprehension*, Portland, OR, USA, May 2003.

[9] Lin Y., Holt R.C., “Software Factbase Extraction as Algebraic Transformations: FEAT”, *1st International Workshop on Software Evolution Transformations*, Delft, The Netherlands, November 2004.

[10] Maletic J.I., Collard M.L., Marcus A., “Source Code Files as Structured Documents", 10th *IEEE International Workshop on Program Comprehension*, Paris, France, June 2002.

[11] McCabe T., “A software complexity measure”, *IEEE Transactions on Software Engineering*, Vol. 2, No. 4, pp. 308-320, 1976.

[12] Mokus A., Fielding R.T., Herbsleb J., “A Case Study of Open Source Development: The Apache Server”, *International Conference on Software Engineering*, Limerick, Ireland, May 2000.

[13] Paulson J.W., Succi G., Eberlein A., “An Empirical Study of Open Source and Closed Source Software Products”, *IEEE Transactions on Software Engineering*, Vol. 30, No. 4, pp. 246-256, April 2004.

[14] Pelz D.C., “Innovation Complexity and Sequence of Innovating Strategies”, *Knowledge: Creation Diffusion, Utilization*, Vol. 6, pp. 261-291, 1985.

[15] Raymond E.S., *The Cathedral and the Bazaar*, 2000, available at: http://www.catb.org/~esr/writings/

[16] Torvalds L., Diamond D., *Just for Fun: The Story of an Accidental Revolution*, HarperCollins, 2001.

Organizational Programming: Hierarchy Software Construction

Zhuo Yin
Institute of Information System & Engineering,
Tsinghua University.
Beijing, P. R. China. 100084
yanzhuo@tsinghua.org.cn

JianMin Wang
School of Software,
Tsinghua University.
Beijing, P. R. China. 100084
jimwang@tsinghua.edu.cn

Abstract

Software's complexity refers to the difficulty to control the consistency between its implementation and design goal. The unrestricted interrelationships of software components are the essential reason, which cause software systems much more complicated. In this paper, we propose "organizational programming", a new technique-oriented methodology, to restrict the interrelationships of software components and make system itself more understandable. Organizational programming principles provide some applicable guidelines to direct software implementation: adopting role-base modeling, we distinguish the process-dependent from the process-independent classes and organize them into different (sub) systems respectively; using "holonomic abstraction", we construct the overall hierarchical architecture of system, which facilitates to achieve component's shareability and reusability simultaneously. Our methodology is intended to information systems or application systems in general and it has been applied to our software practice successfully currently.

Keywords: *organizational programming principles, software engineering methodology, hierarchy*

1. Introduction

As Brooks' well-stated observation [1], people like to see complexity is the essential property of software system. Most prevailing SE methodologies focus on the management aspects of software development and less concern with the concrete software techniques. It's time to turn our attention to the technical aspects. In this paper, we take the perspective that the solution of software engineering relying on the constraints of the interrelationships of software components and propose "organizational programming", a new technique-oriented methodology, to make software system itself more understandable. Our methodology is intended to information systems or application systems in general and its core idea relies on that the hierarchy is the most important mechanism for people to cope with complex problem and role-based assignment is the essence of effective collaboration.

The paper is organized as follows: Section 2 talks about the complexity of software and briefly reviews the current software engineering technologies. Section 3 explains the foundation of our methodology and integrates ideas of system, cognition and organization theories. Section 4 introduces five programming principles to construct the hierarchical software architecture and restrict the interrelationships of software components. Finally, Section 5 provides some practical examples and Section 6 presents conclusions.

2. Software and Complexity

2.1. Software's Complexity

Software helps us to solve the complex problems. Normally, its complexity should be related to the complexity of the problem that it wants to solve. Why software seems to be "more" complicated? Let's take some deep insight: First, programming is a design activity. The delivery of software is its static design, but programmer must ensure that its dynamic behavior will satisfy the desired specifications. Second, software components are logic entities and do not have physical limitation amongst their interactions. It is possible that there are more complex interrelationships between them than other engineering entities. Finally, people like to see programming as an art and usually loose to (or have no good idea to) control its complexity. Overall, software's complexity refers to the difficulty to control the consistency between its implementation and design goal, and mainly represents its readability,

that is, whether the program is easy to be understood. The unrestricted interrelationships of components are the essential reason, which cause software much more complicated.

2.2. Software Complexity Control

Structured programming is one step toward the simplification. GOTO-less programming restricts the using of "Go To" statements and maintains the "independent coordinates" between static-code form and their dynamic-execution process [2]. "Go To" is no harmful for computer, but for programmer. The restriction of control paths improves the system's understandability at the statement level. Moreover, top-down design concerns with the relationship of software components: the layered structure implements "abstract resources", which facilitates to share and reuse the modules of lower levels. The hierarchical structure constraints the scope of effect of modules and facilitates to analyze and construct software step-wisely [3], [4]. Unfortunately, there are some conflicts between them: the physical structure of the layered system is a graph; the hierarchical structure is a tree.

Object encapsulates states and behaviors and helps us to achieve "information hiding". Object-Oriented Design focuses on inherited refinement and domain-orientation abstraction, which classifies the classes firstly and adds the relationships amongst them later and arbitrarily. As a result, the overall structure in object-oriented systems is often much complicated than in procedural systems [5]. Design Patterns abstract best solutions and reuse the local structures to solve certain problems [6]. Software Architecture classifies and studies the typical application skeletons for certain well-defined domains [7]. All of them are experience-based and can't cover with the entire system. Meanwhile, the choice itself could become a problem. Service Oriented Architecture (SOA) models enterprise solutions as federations of services, which can be invoked through published and discoverable interfaces [8]. Component-Based Development (CBD) constructs the explicit context dependent components to achieve coarse-gained reuse [5]. The service model relies on the intrinsic process-independent properties of the problem domain and the reusable components are always common functions. SOA and CBD are not applied to all applications. Moreover, the components and services themselves are usually complicated; we need some good ideas to implement them firstly.

On the inspiration of structured programming, we need some guidance to restrict the relationships of objects. Booch thinks that the hierarchy is the third mechanism of human to cope with complex problems [9]. Hierarchical Object-Oriented Design (HOOD) wants to combine the advantages of the top-down decomposition and OOD, which adopts a project-oriented method to build object's hierarchies [10]. Maciaszek et al. emphasize the structural benefits of "part-of" relationship and propose using "holon" to minimize dependence between software objects [11]. Fischer suggests using the reference factor (RF) and the hierarchy factor (HF) to distinguish project-specific from project-independent components, which belong to application and library domains respectively [12].

3. System, Cognition and Organization

3.1. Complex System

In some sense, everything is a system, which is a collection of subsystems integrated to accomplish the common goal. As a result, the whole is more than the sum of its parts. Simon figured that the complex systems frequently have the similar attributes: [13]

- "Complexity frequently takes the form of hierarchy…"
- "Complex systems will evolve from simple systems much more rapidly if there are stable intermediate forms than if there are not."
- "…hierarchies have the property of near decomposability. Intracomponent linkages are generally stronger than intercomponent linkages."

Hierarchy, stable intermediate forms, and nearly decomposability facilitate us to understand complex systems.

3.2. Human Cognition

Traditional analysis focuses on decomposition, which separates the pieces of what is being studied, and uses abstraction to generalize common features of difference objects, which forms inherited structure of classes. Adopting the systems thinking, we need some counterpart of decomposition. For convenience, we introduce some conceptions in this paper: one is "*composite abstraction*", which represents a "part-of" relationship and characterizes the synergic effect of related objects. The other is "*holonomic abstraction*" which is a *composite abstraction* at the (sub)system level and generalizes the overall behavior of all components within the (sub)system. For example: Consider a family, which has a father, a mother, and a son. Outcomes of the *composite abstraction* could be a couple, a father-son or a mother-son, while that of the *holonomic abstraction* only could be a family.

Holonomic abstraction is very important in human cognition and problem solving, which helps us to analyze the system with the blocks of whole subsystems. In a sense, it constructs the stable intermediate forms of our cognition.

3.3. Organization and Market

Organization is the most important mechanism of human dealing with the complex social problem. In organizations, each person is assigned to a specific role and associated with some well-defined functionalities and responsibilities. Moreover, partition groups of individuals into departments and a new managerial role, was evolved to handle the problems of coordination. In a sense, a manager is the *holonomic abstraction* of his department and helps to form the hierarchical structure of organization. It's hard to find the strict hierarchy in actual organization. A supporting role, staff, which provides the advice and service, has been introduced in Line-Staff Organization. Depending on staff, organization achieves "resource sharing" without disturbs the overall hierarchy [13], [14], [15]. Although modern organization theories like to concern with the adaptivity and openness, the hierarchical control structure and the definitely managerial, operational and supporting roles assignment are the essential mechanisms of social complexity control.

Market is another important mechanism to coordinate social behavior, which relies on that sometimes we lack the power for the request of the hierarchic structure [13]. Normally, market copes with the relationships amongst organizations (individuals).

4. Organizational Programming

In the Multi-Agent Systems (MASs) community, a broad spectrum of social patterns has been introduced to cope with the behavior of agents: single, market, virtual enterprise, alliance, strategic network, group, corporation and even self-organization [16], [17]. The agent, which is per se a complex system, is suitable for modeling, but not for construction. Organizational programming uses organization metaphor to depict the hierarchical software architecture and emphases the constraints of the relationships of system components. Here, we use principles to provide some applicable guidelines of programming implementation and use methods as the refinements, enhancements and supplements of the principles.

4.1. Boundary Principle

Boundary principle divides the inner and outer environments of software system and enables us to adopt different mechanisms of the complexity control.

Boundary Principle bounds software systems based on the entry points. The components below the same entry point usually have the strong linkages and could be developed under the same project. We have the chance to restrict the interrelationships amongst them and construct their hierarchical structure. For desktop and console applications, the active class is the sole entry point and can be the control root of the entire system. Multi-entry points system can be viewed as an associated-organization, while each entry point is a sub-organization and could have its own hierarchical structure. Furthermore, if each entry point provides independent service, like most web service applications, the whole system can be seen as a collection of independent sub-organizations; if entry points are process-dependent, like web application (or the server system of the C/S application), the web return pages (or the client system) coordinate the behavior of each sub-organization. Library is a specific multi-entry points system, which each class can be seen as an entry point, and we will talk about it later.

Method 1-1: The multi-entry points system can be viewed as an associated-organization, while each service could has its own hierarchical structure.

Method 1-2: Market mechanism is applied to coordinate the interactions amongst organizations.

4.2. Division Principle

Object-Oriented Design adopts the idea of entity-based modeling. A class only reflects its domain nature and not related to the role that it plays in system. Conventionally, role-based modeling uses role to abstract the "narrow" behaviors interface of class [18]. In organizational theory and MASs, role is more used to analyze the behavior and interaction characteristics of autonomous individuals. Here, we use role to distinguish the process-dependent from the process-independent classes and more concern about its structural aspects: each class has a well-defined role type, which is at a definite position of the system and has a definite set of relationships.

Division Principle classifies the classes into three categories: *management* class, *task* class, and *support* class. The *management* and the *task* classes are process-dependent which associated with business processes. The *support* classes provide the process-independent services and could be out of the system.

Method 2-1: The *management* class should only contain "policy method", which makes context-

dependent decisions. The *task* class could contain "implementation method", which is the execution of specified algorithms.

Method 2-2: The *support* class provides sharable services for both *management* and *task* classes, and can be further divided into *tool* class, s*taff* class and *product* class. *Staff* class states the common rule and usually provides "static" services. *Product* class is serializable and is transferrable between task classes.

4.3. Partition Principle

The existence of high reusable components does not guarantee the reuse of them. The developers maybe do not know what reusable components exist and how to access them [19]. Fischer use RF and HF to distinguish project-specific from project-independent components [12]. But, the project-specific components might have different RF and HF characteristics also. We need a way to group software artifacts according to their intrinsic properties.

Partition Principle is to classify the subsystems within a (sub)system into two categories: *business* subsystem and *support* subsystem. The classes of different categories have the different functional and structural properties. According to the Line-Staff organization, we groups the classes through their role types: the *business* (sub)systems are process-dependent and organized according to process or product, which facilitate to achieve the hierarchical decomposition. The *support* (sub)systems are process-independent and organized according to function, which facilitate to achieve the layered decomposition.

Method 3-1: *Business* (sub)system consists of *management* and *task* classes and is partitioned by either process or product.

Method 3-2: *Support* (sub)system contains *support* classes and is partitioned by function.

Partition principle provides a coincident manner to deal with the components belong to application and library domain: the Standard Class Library is a common *support* system, which is out of the system and contains project-independent classes. The project-specific *support* classes are organized into project library (out of the system) or *support* subsystem. Moreover, it also facilitates us to achieve software reuse: the classes in *support* (sub)system are potential reusable components and *business* subsystems are more domain-specific, which could be the source of frameworks (patterns).

4.4. Hierarchy Principle

Hierarchic structure is a major facilitating factor enabling us to understand and describe the complex systems and their parts. But layered structure also facilitates us to construct software system. We are in a dilemma whether to choose hierarchy or layer. Booch likes hierarchical object structure but does not tell us how to achieve it [9]. HOOD has to use the uncle and environment objects, which are out of the hierarchy [10]. The PCMEF model of Maciaszek is a pattern-based layered architecture finally [11]. Organizational programming distinguishes the *business* from *support* (sub)systems definitely and uses *support* (sub)systems to encapsulate the sharable components, which enables us to construct the hierarchy of *business* (sub)systems.

Hierarchy Principle restricts that each *business* subsystem has only one contract *management* class.

Method 4-1: *Management* class is the *holonomic abstraction* of its (sub)system and acts as "abstract" *task* class at the directly supersystem.

Method 4-2: *Business* (sub)system could contain *business* and *support* subsystems and could be partitioned in turn.

The *management* class is the sole interface of the *business* (sub)system, which divides the system into independent pieces that can be analyzed, implemented and maintained respectively. In a sense, the *business* subsystem becomes a potential stable intermediate form of the entire system. Meanwhile, organizational programming also constrains the scope of effect of *support* classes, which facilitates us to maintain the sharable components at different levels. Finally, unlike parent object in HOOD [10] and the "Façade object" in Façade Pattern [6] are almost empty shells. The *management* class fulfills all decision and coordination functions of the (sub)system and achieves the "unity of command", which facilitates to clarify the system's decision process.

4.5. Cooperation Principle

In OOD, the relationships can be added to classes arbitrarily. Due to polymorphism, inheritance and dynamic binding, the program's runtime structure often bears little resemblance to its code structure. Trying to understand one from the other is very difficult [7]. We need a way to regulate the relationships of classes and establish the "independent coordinates [2]" of the system's static architecture to its dynamic structure. **Cooperation Principle defines precisely relationships amongst classes.**

Method 5-1: *Management* class has *composition* relationships with their direct subordinate (abstract)

task classes. The *task* classes, within the same subsystem, might have *cooperation* relationship.

Method 5-2: *Management* and *task* class may have *dependency* relationship with *support* classes, which in the same level or the superior level *support* systems.

Design Patterns favor object composition over class inheritance [6]. Organizational programming does not proposal to use inheritance in *business* (sub)system but only use it (or interface implementation) to uniform the interfaces of *task* classes. Contrarily, the *composition* relationship is trivial, which facilitates us to achieve the "centralized management". Finally, the system's class structure and its object structure are isomorphic: the *composition* relationship implies construction, initiation, command and report relations amongst relevant *management* and *task* objects; the *cooperation* relationship implies service access relation amongst relevant *task* objects.

5. Programming Practice

Briefly, we provide a practical application of organizational programming, which has implemented a TPC-H Benchmark testing driver using C#. The Benchmarks is issued by Transaction Processing Performance Council (TPC) and used to evaluate the performance of decision support systems. The business model of TPC-H is an international wholesaler, which has a database to store business data and provides 22-query (business report) and 2-refresh operations [20].

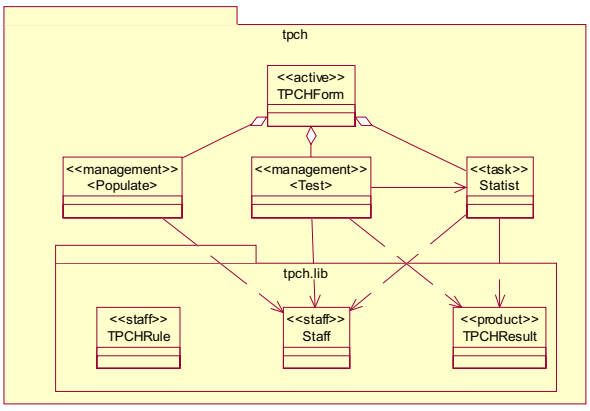


Figure 1. The structure of TPC-H system top level

The top-level of the TPC-H system is shown in Fig. 1. The boundary of the system is its top directory (naming space) "tpch", which consists of an active *management* class "TPCHForm", a *task* class "Statist", a *support* subsystem "tpch.lib", and two *business* subsystems "tpch.populate" and "tpch.test", whose *management* classes shown in the top-level structure as "abstract" *task* classes, <Populate> and <Test>. The <Test> has *cooperation* relationship with the "Statist" which statistics the test result. The *management* class "TPCHForm" is a GUI class and has *composition* relationship with above three (abstract) *task* classes.

The *support* subsystem "tpch.lib" has three *support* classes; all of them can be shared within the system. In details, the "Staff" *staff* class manages database connections and conducts log etc.; the "TPCHRule" *staff* class defines the common system rules and generates the common populate and query data of TPC-H test; the "TPCHResult" *product* class records the performance data. The <Test> and the "Statist" have *dependency* relationships with the "TPCHResult" and the "Staff". The <Populate> has *dependency* relationship with the "Staff" also.

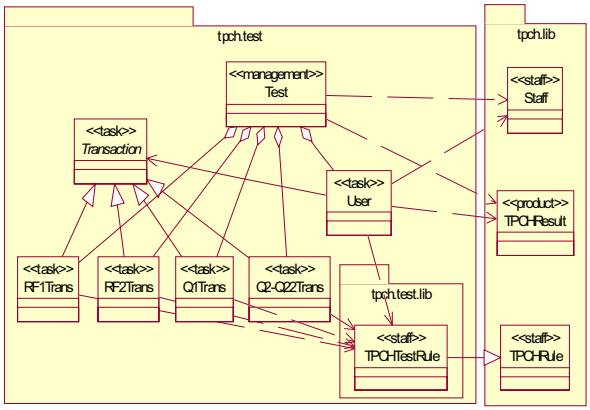


Figure 2. The structure of tpch.test subsystem

For brief, we only show the "tpch.test" subsystem structure as Fig. 2. Here, the *management* class is "Test". The *task* classes can be classified into two categories: one is dispatcher class "User", which do dispatching and timing in independent thread mode; others are 24 operator classes, which include two refresh classes ("RF1Trans" and "RF2Trans") and 22 query classes (for brief, query classes Q2 to Q22 shortening as "Q2—Q22Trans"). All operator classes are the subclasses of the abstract *task* class "*Transaction*". Since the "User" class uses the "*Transaction*" array to dispatch operator classes, we only label the *cooperation* relationship from the "User" to the "*Transaction*".

The "Test" and the "User" have *dependency* relationships with the "Staff" and the "TPCHResult", which in the higher level *support* system "tpch.lib". The *support* subsystem "tpch.test.lib" has only one *staff* class "TPCHTestRule". It defines specific rules of "tpch.test" subsystem and is a subclass of the "TPCHRule", which inherits the common methods of

system. All *task* classes have *dependency* relationships with it.

Ignoring *inheritance* and *dependency*, the *management* and *task* classes, which really involved in business process, only have *composition* and *cooperation* relationships. Because of the *composition* relationship in organizational programming is trivial. The relationship that we should pay more attention is only *cooperation* relationship. Meanwhile, it is easy to understand the interactions of objects from their code structure: the *management* object constructs, initializes, dispatches and monitors its subordinate *task* objects; the *task* objects report their states to their superior *management* object; the *task* objects access the service of their collaborative *task* objects. Second, the *management* class is the "*holonomic abstraction*" of its subsystem, we can model the subsystem as an "abstract" *task* class directly at the direct higher level, which reduces the number of classes that we have to deal with simultaneously. Finally, the developing for reuse and the developing with reuse are more smoothly: the sharability of classes in "tpch.lib" is trivial and we only need to take a little more effort to make some of them project-independent; the hierarchical architecture of "tpch" is intrinsically simple and clarifying and need only a little refactoring to become a information system general performance testing frame. Actually, we have used the general frame to finish a TPC-C project using C#, a TPC-W project using Java and some other commercial testing projects successfully. In each project, it is easy to find and verbatim reuse the "Staff" and rewriting reuse the other *support* classes and even *management* or *task* classes.

6. Conclusion

Organization programming concentrates on the technique aspects of software implementation, which provides some applicable guidelines to control the complexity of software system and makes it more understandable. Meanwhile, it helps to improve the modularity, reusability and maintainability of the software system and achieves notable improvement of software quality and productivity.

Some future works below:

1. Organization programming principles concentrate on the software design and implementation phases. As a complement, organizational software engineering process, which covers other topics of the software development process, is under developing.

2. Market mechanism is applied to cope with inter-system coordination problems. An in-depth study of the applicable techniques will get under way soon.

3. Our methodology has not used in large projects currently, we look forward to its increasing application, popularization and improvement in the future.

References

[1] Frederick P. Brooks, The mythical man-month : essays on software engineering. Addison-Wesley, 1975.

[2] E. W. Dijkstra, Goto Statement Considered Harmful. Comm. ACM, 1968, 11(3), pp. 147-148.

[3] E. W. Dijkstra, Structured Programming. Classics in Software Engineering (edited by Yourdon E. N.). Yourdan Press, 1979, pp. 43-50.

[4] E. F. Miller, G. D.Lindamood, Structured Programming: Top-Down Approach. Classics in Software Engineering (edited by Yourdon E. N.). Yourdan Press, 1979, pp.187-194.

[5] Clemens Szyperski, Dominik Gruntz, Stephan Murer, Component software: beyond object-oriented programming, 2nd ed. Addison-Wesley, 2002.

[6] Erich Gamma etc. Design Pattrens: Elements of reusable Object-Oriented Software. Addison-Wesley, 1997.

[7] L. Bass, P. Clements, R. Kazman, Software Architecture in Practice. Addison-Wesley, 1998.

[8] A. W. Brown, Model driven architecture: Principles and practice. Published online, Springer-Verlag, 2004

[9] G. Booch, Object-Oriented Analysis and Design with Applications, 2nd edition. Benjamin/Cummings Pub. Co., 1994.

[10] Peter J. Robinson, Hierarchical Object-Oriented Design. Prentice Hall, 1992.

[11] L. A. Maciaszek, B. L. Liong, Practical software engineering: a case study approach. Pearson/Addison-Wesley, 2005.

[12] G. Fischer, D. Redmiles, L. Williams etc. Beyond Object-Oriented Technology: Where Current Approaches Fall Short. Human-Computer Interaction, Vol.10, 1995, pp. 79-119.

[13] H. A. Simon, The Sciences of the Artificial, 3rd. The MIT Press, 1996.

[14] J. R. Galbraith, Organization Design. Addison-Wesley, 1977.

[15] J. D. Mooney, The principles of organization. Harper & Brothers, 1947.

[16] M. Schillo, K. Fischer, B. Fley, etc. FORM - A Sociologically Founded Framework for Designing Self-Organization of Multiagent System. Regulated Agent-Based Social System. LNAI 2934. Springer, 2004.

[17] F. Zambonelli, N. R. Jennings, M. Wooldridge, Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems. International Journal of Software Engineering and Knowledge Engineering, 2001, 11 (3), pp. 303-328.

[18] C. Richter, Designing Flexible Object-Oriented System with UML. Macmillan Technical Publishing, USA, 2001

[19] V. Basili, G. Caldiera, D. Rombach, The Experience Factory. Encyclopedia of Software Engineering, Wiley, 1994

[20] Transaction Processing Performance Council (TPC), TPC Benchmark H (Decision Support), Standard Specification Revision. 2.1.0. http://www.tpc.org.

Towards a Methodology for Hybrid Systems Software Development

Isabel María del Águila[1], Joaquín Cañadas[1], José Palma[2], and Samuel Túnez[1]

[1]Dept. of Languages and Computation. University of Almeria. Spain
E-mail: {imaguila,jjcanada,stunez}@ual.es
[2]Dept. of Information and Communications Engineering. University of Murcia. Spain
E-mail: jpalma@dif.um.es

Abstract

On numerous occasions the complexity of the domains forces the software development process to use both Software Engineering and Knowledge Engineering methodological approaches. This work presents specific solutions to problems raised in the development of software performing knowledge intensive tasks, related to the decision support process, and non-knowledge intensive tasks, related to transactions processing and information management. After identifying these problems, we propose using the Requirements Engineering as the way to classify system functionalities as knowledge based and non-knowledge based ones, and the Model Driven Architecture (MDA) as an approach to obtain design and implementation models from conceptual models in a systematic and controllable way.

1. Introduction

Nowadays computer applications are integrated in multiple domains and they are the key element that distinguishes modern products and services. Usually, people work, wellbeing, security, entertainment, decisions and their own lives depend on computer software. Any computer application must be developed as an industrial product, applying engineering methods to ensure the resulting product quality, that is, using Software Engineering (SE). But in many situations, the complexity of the domains entails the necessity of using techniques that exceed the SE, in order to develop software that solves the necessities raised in those domains. This kind of software applications requires the utilization of non-transactional or heuristic techniques, that is, knowledge based solutions, and therefore, a methodological approximation typical of Knowledge Engineering (KE).

In this context, *Hybrid Systems* [12] are software systems composed by several components or subsystems, some of them performing knowledge intensive tasks related to the decision support process, and others performing tasks related to transactions processing and information management in databases. The former are knowledge based components and the second are non-knowledge based components.

Knowledge based software and non-knowledge based software have different nature. This affirmation is sustained by the differences in the activities, models, languages and tools used for the development of both kinds of software. Focusing on knowledge based software, its development involves the specification of the *knowledge model* [15, 21], a declarative and explicit representation of the knowledge that users and experts apply in their reasoning. Knowledge engineers build up these models in order to develop intelligent software in different domains and environments [23]. Moreover, implementation tools for knowledge based software provide features which are not included in traditional software, as knowledge representation techniques (rules, decision trees, ...), uncertainty representation (fuzzy logic, bayesian networks, ...), ontologies, reasoning methods, learning, and so on.

The experience of our research group developing hybrid systems in different application domains, as medicine [20] and agriculture [2], has led us to conclude that the application of existing methodologies presents some problems. Firstly, when SE methodologies are used, the knowledge model can not be specified adequately: it must be modelled separately, producing coherence and integrity problems, or it must be mixed with non-knowledge conceptual models, distorting the applicability of the knowledge model.

Secondly, if a KE methodology is used then integration problems with non-knowledge based functionalities arise. These methodologies do not consider this kind of functions, which can not be modeled adequately. Even if the modeling of non-knowledge based functionalities is forced, the effort would be out of proportion because algorithmic tasks would be considered as knowledge intensive tasks.

So developing hybrid systems requires to integrate SE and KE techniques and activities. It is necessary the intertwining between SE and KE, in order to obtain the best result and a successful final product [1].

Nowadays there is no an integrated method for hybrid system software development. Several solutions that partially solve the problem have been proposed, working on management models or live cycles [4, 5], proposing alternative process models [1], or a layered definition [12], or the division between a system definition at contents level, bound to knowledge, and a container level bound to software implementation [10], or using languages and models integrating knowledge based and non-knowledge based software [3]. But all these are descriptive proposals that should be developed in detail, providing models and languages, activities, artifacts and specifying documental structure; in brief, the complete definition of a methodology for hybrid system software development.

Another problem, mainly presented in knowledge intensive software, is the traceability from conceptual models to computational models, that is, how to obtain the system design and implementation from conceptual models. The principle of structure preserving design advocates that elements of design models must be directly related to elements of conceptual models [21]. In the SE community, the problem of defining design models from conceptual models is solved and there are several techniques and approaches for system design sufficiently tested and validated. However, KE methodologies, and CommonKADS [21] in particular, lack of a rigorous discipline to direct the transition from conceptual models to system design: the relationship between conceptual model elements (task, problem solving methods, ontologies,...) and components in the design model is not well established.

Among the SE approaches for the definition of design models stands out the Object Management Group (OMG) initiative called Model Driven Architecture (MDA) [13, 16]. MDA approaches the software process through the definition of different models, allowing the specification and execution of transformations from one model to another in an automated way.

This work presents solutions for specific problems in hybrid systems software development. Firstly, we propose using Requirements Engineering (RE) [22] as the way to classify system functionalities in knowledge-based and non-knowledge based. This classification allows performing the specification of the functionalities using the most adequate conceptual modeling approach.

Secondly, we propose using MDA in the development of the knowledge based components of hybrid systems, in order to guide the transition from conceptual models at knowledge level, written in the Conceptual Modeling Language (CML)[6] proposed in the CommonKADS methodology [21], to design models of the system under development, and finally to code on a specific platform, usually a development tool for knowledge based systems as *JESS*[1], *ARTEnterprise*[2], or *G2*[3].

These proposals are the first steps in the definition of a Software and Knowledge Engineering Methodology, specifying in an integrated method the process, activities, languages and tools used for hybrid systems software development.

The rest of this work is structured as follows: section 2 describes the modeling activities in hybrid software development. Then, section 3 introduces the application of RE in hybrid systems. Section 4 presents how the MDA is used in hybrid systems development. Finally, the main conclusions and future work are summarized.

2. Hybrid software modeling

In general, the main modeling activities in software development are conceptual modeling and computational modeling [7]. Conceptual modeling is independent of any implementation technology. Developers must transform conceptual models into computational models in a way that, depending of the development approach, a conceptual model turns into one or several computational models, such as object oriented, structured or following a knowledge-based representation.

In hybrid systems software development, there are two kinds of conceptual models: conceptual models for knowledge based components and conceptual models for non-knowledge based components. These models are defined from two different axes, as Figure 1 shows. On the *Z* axis, the conceptual model at knowledge level is a declarative and explicit representation of the knowledge that experts apply in their reasoning processes. On the *X* axis, conceptual models are obtained from the organization strategic objectives and users needs, representing non-knowledge based functionalities.

The process of obtaining computational models from conceptual models at knowledge level is the aim of KE (plane *ZY* in Figure 1). Similarly, SE is represented in the plane *XY* in Figure. Hybrid software development involves

[1] Java Expert System Shell. http://herzberg.ca.sandia.gov/jess
[2] http://www.mindbox.com
[3] http://www.gensym.com

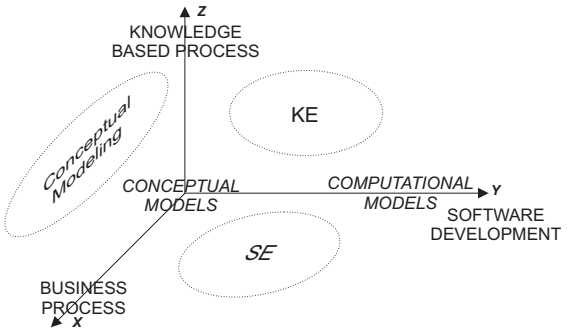


Figure 1. A 3-D representation of hybrid software modeling

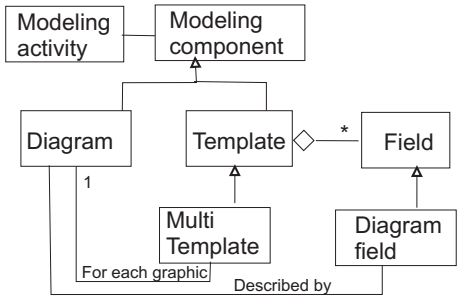


Figure 2. RE metamodel

the existence of a third plane, *XZ*, representing the necessity of conceptual modeling from both engineering points of view.

3. Requirements Engineering for hybrid systems

In hybrid system development process, an early classification of system functionalities in knowledge based and non-knowledge based ones can arise problems in later steps of the development process. In order to illustrate this affirmation, we describe the experience reached in the development of the system SAEPI, a hybrid system developed as part of the research project "An Intelligent Decision Support System for the South-East Spanish Agricultural Environment", funded by the European Community and the Spanish Ministry of Science and Technology. This system provides functionalities developed using information management techniques, and it includes a decision support subsystem for growers that advises them about pest control decisions in typical crops of the South-East of Spain [8].

After an initial study of the problem, the partition in subsystems providing the main functionalities was performed: each subsystem was classified to be developed using a SE methodology, in particular Rational Unified Process (RUP) [11], or a KE methodology, in this case the CommonKADS methodology [21]. However, when the project development progressed, we confirmed the wrong initial classification of some functionalities as knowledge based, because they were more transactional than it was expected initially. Therefore, in these cases a SE method was a better approach than a KE one. Similarly, some objectives planned as transactional were finally developed using KE techniques due to their nature and complexity.

In order to solve this problem, a harder work in problem requirements is proposed, applying RE techniques before classifying system functionalities in knowledge or non-knowledge intensive ones.

RE allows software specification by means of two kinds of requirements: *goals* and *services*. *Goals* are requirements at organizational level bound to business processes, and *services* are requirements at software level becoming goal refinements and allowing the specification of the conceptual model.

Figure 2 describes the proposed metamodel used in the specification of hybrid systems requirements -goals and services. Two modeling elements are used: *diagrams* and *templates*. Each template is composed by a set of *fields*, and some fields are described by means of a diagram (*diagram-field*). Also, each diagram is described by means of a *multi-template*.

Following this metamodel, two modeling activities are defined: the first one works with goals at business level, and the second one works with services at software level.

In the first requirement activity the business level is specified using a *Project* template, which defines the project organizational context. Business processes are specified by means of one or several diagrams. Figure 3 describes an example using use cases in business processes modeling. Every use case must be described by means of a *Goal* template. As result of this activity, main goals are obtained allowing a first non-disjoined classification of system functionalities into knowledge based and non-knowledge based ones.

From business level on, the development with a KE methodology as CommonKADS begins in parallel, focusing on functionalities identified as knowledge based in the previous requirement activity. A knowledge model in a language as CML is specified. KE must be applied in parallel to RE because the former analyzes the system goals from the expert point of view, whereas RE does from the user point of view.

Second activity in RE refines the business level into the service level. So every goal is revised and described identifying its workflow. As we said before, KE is being applied in parallel, so a knowledge model and a service model are available. Both models are matched in order to classify functionalities in knowledge based and non-knowledge

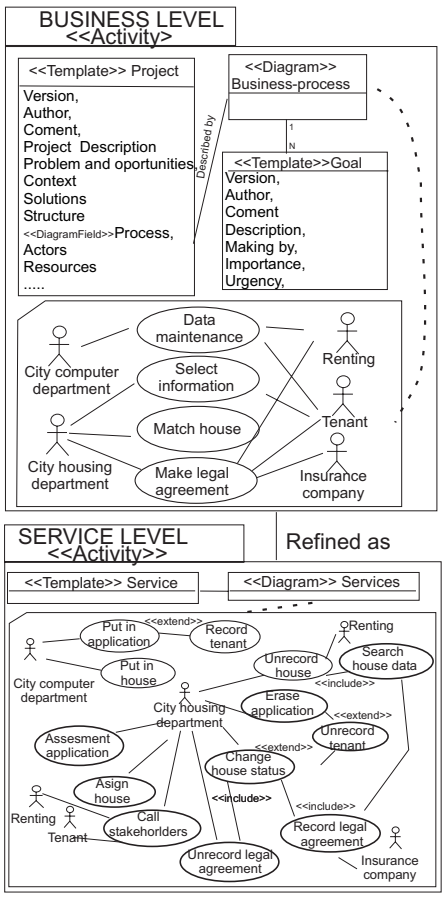


Figure 3. Business and Service specification

based. The developer must search in the knowledge model for a description of knowledge based services, and the non-knowledge ones are described by means a specification using a SE language for conceptual modeling as UML (Unified Modeling Language).

4. Hybrid software development with MDA

As we described in the introduction, we propose using the Model Driven Architecture (MDA) [16] as the approach to guide the transition from conceptual models to computational models.

MDA [4] is a framework for software development which is driven by the activity of modeling the system [13, 16]. The first model that MDA defines is a model with a high level of abstraction that is independent of any implementation technology. This is called a Platform Independent Model (PIM). Next step in the development consists on transforming the PIM into one or more Platform Specific Models (PSMs). A PSM specifies the system in terms of

[4] http://www.omg.org/mda

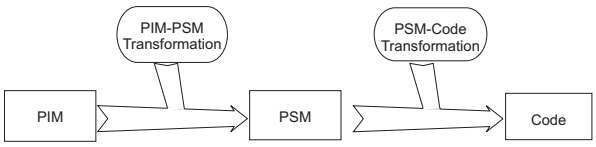


Figure 4. The basic steps in MDA

the implementation constructs that are available in one specific implementation technology. For example, a relational database PSM is a model of the system in terms of "table", "column", "foreign key", and so on. The final step in the development is the transformation from PSM to code. This transformation is relatively straightforward because a PSM is a model directly related to its platform technology.

Model transformation consists on the process of converting one model to another model of the same system in a different abstraction level: from PIM to PSM and from PSM to code. Also a transformation from PIM to PIM is possible, in order to represent the same PIM in a different modeling language. MDA tools allow these transformations to be automated and executed automatically. Figure 4 show the MDA approach to software development.

The main elements of the MDA framework are a set of standards: UML [18], is the modeling language; the Meta Object Facility (MOF) [17] defines a common, abstract language for the specification of metamodels; and Query/Views/Transformations (QVT) [19] is the upcoming standard for transformations specification. In general, the term Model Driven Development (MDD) [14] is used for this approach in software development process that do not follow these standards.

Table 1. MOF four-level metamodel hierarchy

Level	*Description*	*UML Modeling*	*CommonKADS Modeling*
M3	**MOF**	MOF Class	MOF Class
M2	**Metamodels**, instances of MOF elements	Class, Attribute, Association ...	Task, Task-Method, Inference, Dynamic Role ...
M1	**Models**, instances of M2 elements	Client, Name, Invoice ...	Diagnosis, Causal-covering, select, hypothesis ...
M0	System runtime **instances**, objects and data	C1: Client name = "John" address = "Main St." ...	aDiagnos: Diagnosis complaint = "High temperature" hypothesis = "Viral Infection"

As well as these standards, OMG defines a four-level metamodel hierarchy [18] in software systems modeling. Table 1 describes the levels in the modeling hierarchy and an example of the elements that take part in each layer in

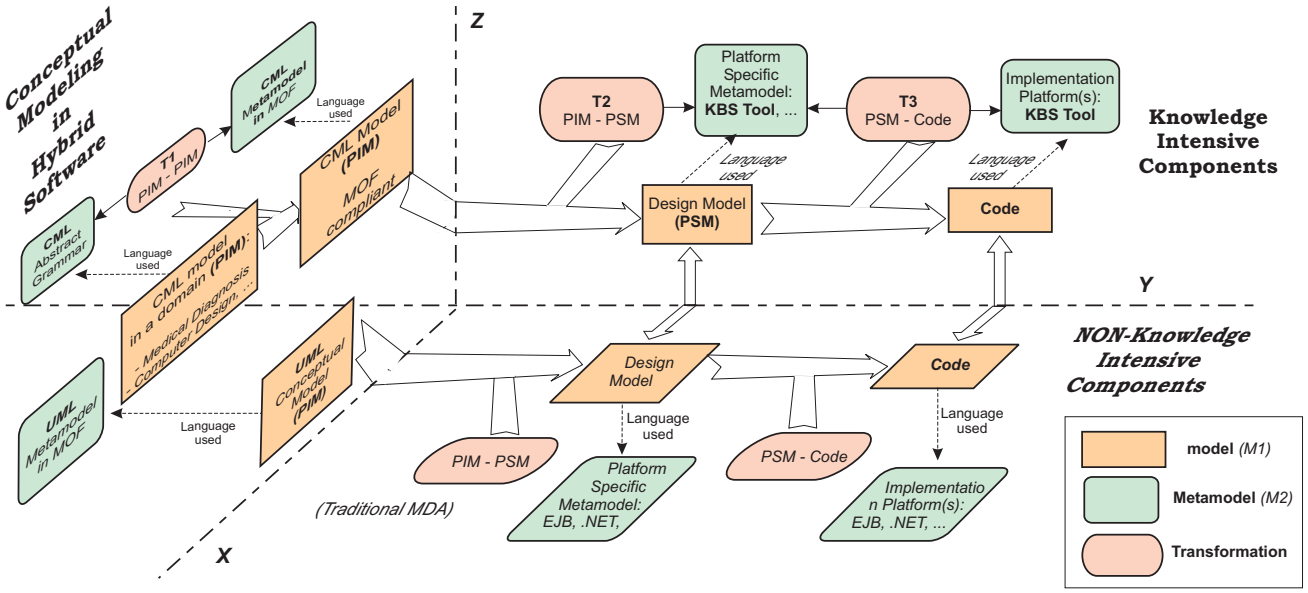


Figure 5. MDA applied to Hybrid Software development

two modeling languages: UML and CommonKADS.

Figure 5 describes the general framework of the application of the MDA approach to hybrid software development, following the same metaphor of three dimensions presented in Figure 1. The workflow covering the bottom part of plane XZ and the whole plane XY presents the traditional MDA approach for non-knowledge based software development, using standard languages and platforms. This process is not the aim of this work but it is a research topic in the SE community. The contribution of this work is the workflow described at the top part of plane XZ and the whole plane ZY of Figure 5. It is focused on the development of knowledge-based software using the MDA approach. Combination of both perspectives allows the development of hybrid software.

Focusing on knowledge intensive components, the process is divided into three main steps. The development starts with the specification of the conceptual model of the system under study. This conceptual model is expressed in a KE language, as CML (source model). Transformation *T1* deals with the first step of converting the source CML model into an equivalent model written in a MDA compliant notation, following the CML metamodel defined in MOF.

Once the PIM is expressed in a language defined by a MOF metamodel, from this point forward the process is a typical MDA framework. In the second step, the *T2* transformation takes care of converting the PIM obtained in the previous step into one (or several) PSM. This PSM is a design model that takes into account the characteristics of the implementation platform used for the system.

We consider that knowledge based components being part of hybrid systems should be designed using a traditional three layered architecture (persistence, business logic, and user interface), the same non-knowledge based components are designed following this architecture too [11]. This approach allows a better integration of the whole system. Therefore, three PSMs should be obtained from the PIM, one for each layer in the architecture. Whereas PSMs for persistence and user interface layers can be obtained by means of MDA transformations already defined by de SE community, the business logic layer of a knowledge based component must be implemented in a knowledge based tool. *T2* is focused on the transformation from a CML model to a PSM for a KBS Tool.

The last step in the development using MDA is the PSM to code transformation. This transformation, called *T3* in the proposal, is relatively direct because a PSM is a model very close to its implementation platform [13].

A detailed description of the transformations proposed, as well as a prototype of tool that allows partially performing the process, are described in [9].

5. Conclusions and future work

In this work we propose solutions to some problems found when developing software composed by knowledge based and non-knowledge based subsystems. The lack of a integrated method to develop this kind of software systems requires applying software and knowledge engineering methodologies interlaced.

Requirements Engineering methods let us to study the problem requirements in order to split the system in dif-

ferent subsystems -knowledge intensive or non-knowledge intensive-, assigning them their functionalities. This reduces later development problems.

Model Driven Architecture facilitates the transition from conceptual models to computational ones. Application of MDA approach constitutes an advantage over traditional KE methodologies, because it fixes the correspondence between the conceptual models and the design and implementation ones, using transformation definition between models. MDA tools can be used for automating this process.

This work proposals are the first step in order to define a new Software and Knowledge Engineering Methodology, oriented to the development of hybrid software from an integrated point of view. At the moment, our research group is working on completing the definition of this methodology and validating these proposals in the development of a system in agricultural domain.

6. Acknowledgements

This work is part of the Research Project "A Methodology for hybrid software analysis and design: integration of software and knowledge modeling techniques" (TIN2004-05694), funded by the Spanish Ministry of Education and Science.

References

[1] S. T. Acuña, M. López, N. Juristo, and A. M. Moreno. Process model applicable to software engineering and knowledge engineering. *International Journal of Software Engineering and Knowledge Engineering*, 9(5):663–687, 1999.

[2] I. M. Águila, J. Cañadas, A. Bosch, S. Túnez, and R. Marín. Knowledge Model of Therapy Administration Task Applied to an Agricultural Domain. *Lecture Notes in Artificial Intelligence*, 2774:1277–1283, 2003.

[3] I. M. Águila, J. Cañadas, S. Túnez, and J. Palma. Integration of Development Methodologies for the Building of Knowledge Intensive Multiagent Systems. In *In Proceedings of International Conference on Integration of Knowledge Intensive Multi-Agent Systems (KIMAS'03). IEEE Press*, pages 203–208, Boston, MA., 2003.

[4] I. M. Águila, S. Túnez, J. Cañadas, A. Bosch, and R. Marín. A Proposal for Project Management Using CommonKADS. *Lecture Notes in Computer Science*, 2178:160–171, 2001.

[5] F. Alonso, J. L. Fuertes, L. Martnez, and C. Montes. An incremental solution for developing knowledge-based software: its application to an expert system for isokinetics interpretation. *Expert Systems with Applications*, 18(3):165–184, 2000.

[6] A. Anjewierden. CML2. Technical Report 11, University of Amsterdam.

[7] B. I. Blum. *Beyond Programming. To a New Era of Design*. Oxford University Press, 1996.

[8] J. Cañadas, I. M. Águila, A. Bosch, and S. Túnez. An intelligent system for therapy control in a distributed organization. *Lecture Notes in Computer Science*, 2510:19–26, 2002.

[9] J. Cañadas, J. Palma, and S. Túnez. From Knowledge Level to Symbolic Level. A Metamodeling Approach. In *Proceedings of Knowledge Engineering and Software Engineering Workshop (KESE2005)*, pages 13–23, Koblenz, Germany, 2005.

[10] A. Gachet and P. Haettenschwiler. Developing Intelligent Decision Support Systems: A Bipartite Approach. *Lecture Notes in Artificial Intelligence*, 2774:87–93, 2003.

[11] I. Jacobson, B. G., and R. J. *The Unified Software Development Process*. Addison Wesley Longman Inc., 1999.

[12] S. Kendal and X. Chen. Towards Hybrid Knowledge and Software Engineering. In *Proceedings of International Conference on Computing and Information Technologies*, 2001.

[13] A. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison Wesley, 2003.

[14] S. Mellor, A. Clark, and T. Futagami. Model-Driven Development - Guest editors introduction. *IEEE Software*, 20(5):14–18, Sep-Oct 2003.

[15] A. Newell. The knowledge level. *Artificial Intelligence*, 18(1):87–127, 1982.

[16] Object Management Group. MDA Guide Version 1.0.1. OMG document: omg/2003-06-01, 2003.

[17] Object Management Group. Meta Object Facility (MOF) 2.0 Core Specification. OMG document: ptc/03-10-04, 2003.

[18] Object Management Group. UML 2.0 Infrastructure Specification. OMG document: ptc/03-09-15, 2003.

[19] Object Management Group. Revised submission for MOF 2.0 Query/View/Transformation RFP. OMG document: ad/2005-03-02, 2005.

[20] J. Palma, R. Marín, M. Campos, and A. Cárceles. ACUDES: Architecture for Intensive Care Units DEcision Support. In *2nd Joint Meeting of the IEEE Engineering in Medicine and Biology Society and the Biomedical Engineering Society*, Houston, 2002.

[21] A. T. Schreiber, J. M. Akkermans, A. Anjewierden, R. de Hoog, N. R. Shadbolt, W. Van del Velde, and B. J. Wielinga. *Knowledge Engineering and Management. The CommonKADS Methodology*. MIT Press, Massachusetts, 2000.

[22] I. Sommerville and P. Sawyer. *Requirements Engineering: A Good Practice Guide*. John Wiley & Sons, Inc. New York, USA, 1997.

[23] M. Taboada, J. Des, J. Mira, and R. Marin. Diagnosis Systems in Medicine with Reusable Knowledge Components. *IEEE Intelligent Systems*, 16(6):68–73, 2001.

After the Scrum: Twenty Years of Working without Documentation

Sukanya Ratanotayanon
Department of Informatics
University of California, Irvine
sratanot@uci.edu

Jigar Kotak
Department of Informatics
University of California, Irvine
jkotak@uci.edu

Susan Elliott Sim
Department of Informatics
University of California, Irvine
ses@ics.uci.edu

ABSTRACT

Agile processes enable software development projects to react to rapid changes in the development environment. However, they are often criticized for not creating and maintaining standard documentation such as requirements and design documentation. The lack of documentation can be detrimental for maintaining knowledge, especially in the long run, because there is no explicit medium for communication to new people and among existing developers. This poses an important question: whether the use of agile processes in long run is feasible. In this paper, we presented a field study of an organization that has been using an agile process for more than 20 years and has been successful in maintaining knowledge over that period. Instead of written documentation, they use living documents, well-connected communication, and working software are prioritized as mediums for maintaining knowledge. However, success is not easily achieved. There are important factors that enable the organization to use the current practices successfully. These factors are: shared values, overlapping knowledge among team members, low turnover rate, and well-understood requirements.

1. Introduction

Processes in the agile family have gained increasing popularity recently due to the competitive environment and rapid changes in both technologies and requirements. The lightweight characteristic of agile processes enables fast-paced development and rapid reaction to change. However, much skepticism toward agile processes has been shown, especially by the proponents of plan-based software processes, due to the lack of standard documentation, which can be detrimental for maintaining and distributing knowledge among project members throughout a project's life cycle.

Agile software processes do not produce and maintain any high-level documents other than source code and comments. Instead, it recommends communication and collaboration among people in the project as a means of maintaining knowledge rather than using documentation. The process suggests that documentation be created and maintained only when necessary and only to facilitate communication. For example, the process advocates documenting important knowledge that will help others understand the source code as comments. The facts that there is no explicit medium for knowledge transferring, and most important knowledge resides only within team members' heads raise an important question. Is long term use of the agile approach feasible? This concern is especially valid when the software project reaches its maintenance phase and the original developers of the system are leaving or are no longer available.

Most field studies of agile projects have not investigated this issue. They have studied projects that only recently adopted agile processes and have focused on how the agile processes were adopted [1], [2]. We have conducted a field study within a small organization that has been using an agile process for more than 20 years. In this study, we interviewed all staff members who were involved in two particular software projects. The first software project is being implemented as a computer system for the first time and is in the development phase. The second project is one of the organization's critical applications and is the third generation of computer systems performing the same task. This second application is in its maintenance phase and has already lost some of its original developers. However, even without documentation, the remaining developers are able to sustain the knowledge required to maintain their software successfully.

At first blush, one would expect that scrums, or frequent, informal face-to-face communication would be insufficient to sustain knowledge over decades. However, this was not the case. We found that mediums through which knowledge is mainly retained and distributed are: living documents, well-connected communication, and exemplar software systems. However, there are a number of factors that enable this organization to use these communication mediums effectively, and these have led to successfully sustaining and distributing the knowledge. These factors are: shared values, overlapping knowledge among team members, a low turnover rate, and well-understood requirements.

In the next section, we review related studies.. The method employed in our study and the characteristics of the organization and projects under study are presented in

Section 3. Section 4 describes in detail the mediums used to maintain and distribute knowledge in this organization. Section 5 analyzes the factors that enable them to successfully use these mediums effectively. Section 6 presents concerns that the organization has about maintaining its knowledge in the future. We conclude our paper in Section 7.

2. Related Work

Agile processes are a family of lightweight processes that share a common framework called "Agile Manifesto" established in early 2001 by the Agile Alliance [3]. The Agile Manifesto declares that agile processes value:

- Individuals and interactions over processes and tools,
- Working software over comprehensive documentation,
- Customer collaboration over contract negotiation,
- Responding to change over following a plan.

In addition, a true agile process will have the following characteristics: iterative, incremental, self-organized by team member, and emergent [4]. Examples of famous agile processes are Extreme Programming (XP), Scrum, Crystal, Adaptive Software Development (ASD), Feature-Driven Development (FDD), and Lean Development (LD).

Agile processes are popularly adopted by small teams building software that requires fast development and fast reaction to changes. Agile processes normally prioritize communication and shared tacit knowledge as a means to maintain and sustain knowledge in software projects. Documents are created only to facilitate the communication and are updated only when necessary [5]. This raises skepticism about the ability of agile processes to maintain and distribute knowledge among team members throughout the development life cycle.

Maintaining and distributing knowledge is crucial in software development processes. This is especially true in the case of maintenance, when sometimes a person who is responsible for maintaining the system is not involved in its development. Information has to be stored in some form in order to help the maintainer gain an understanding of the system. Failure to address this issue could lead to a "thin spread of application domain knowledge," and "communication and coordination breakdown" [6], which eventually lead to the failure of the project.

In traditional plan-based processes [7-9], a standard set of documents associated with software development phases is prescribed as the medium for communication to new people on the project, existing developers, users, and other related software projects. These documents are used to enforce conformity of coding standards, design documents, and requirements, as well as to provide mechanisms for tracking project features, status, and bugs.

The cost of producing and maintaining these documents, however, poses a problem, especially in small teams such as those practicing agile processes. Although there are many advocates for using documentation as a means to transfer and sustain knowledge [10-12], there is also evidence against the effectiveness of documents as a means to maintain and distribute knowledge. Studies show that, in practice, developers do not do a good job of documenting [13]. In addition, developers don't feel that it's worthwhile to update certain types of documents. Therefore, documents are not created and maintained a timely manner, and they become outdated [14, 15]. This inconsistency undermines the value of documentation as a means for maintaining knowledge in an organization.

3. Empirical Method and Field Site

In order to better understand how knowledge is maintained and transferred without the benefit of documentation, we conducted a field study of an organization that has been using an agile development process for more than 20 years. This section presents the method employed in our study and the characteristics of the organization and projects studied.

3.1 Method

We conducted interviews with 9 technical and non-technical staff members who were involved in two specific software development projects. A detailed description of the projects is presented in section 3.2. We employed semi-structured interviewing techniques, which involve a set of questions designed to cover the ranges of topics. These questions are open-ended in order to encourage subjects to talk at length about their experiences and also help us learn about other related issues. The questions, selected to promote conversation, fall into three categories: (i) their background and their roles in the project, (ii) documents and information required for their tasks, and (iii) the means used to obtain required information and to record information obtained during their tasks.

Each subject was interviewed individually for 30 to 60 minutes. For each interview, there were two interviewers: the first interviewer took the role of ensuring that all the topics in the protocol were covered, and the second interviewer, freed from focusing on the protocol, focused on following up on interesting but unanticipated remarks. In addition to the notes taken during the interview, interviews were tape recorded.

3.2 Field Site and Project Characteristics

The interviews were conducted at a small administrative office at a university, which we will call "IStar." IStar has a flat hierarchy and employs 14 individuals: 6 nontechnical staff and 8 technical staff. Non-technical staff carry out IStar's business. Technical

staff implement software to support IStar's works. Each division has its own manager, who reports to the head of the organization. At the time of the interviews, there were 7 full-time developers and 1 student programmer in the technical division.

Recently, the trend of software development in IStar has been to "move everything to the web". All paper-based and client-server systems were being migrated to web applications in order to: (i) provide better service to the students and staff, and (ii) eliminate additional costs attached to paperwork. We studied two specific web application projects, WA1 and WA2. WA1 is a web application for administrators to view and perform job-related functions concerning a student's enrollment. WA2 is a web application that allows students and counselors to process requests for graduation. WA1 is a version of an existing client-server system developed in IStar that has been ported to a web application and is already in the maintenance phase. WA2, on the other hand, is a newly created application that replaces a paper-based system and recently went into production.

Although we interviewed only staff members who were involved in WA1 and WA2, the process employed by both projects is the same for all software development in IStar, and it has been used for more than 20 years. The agile processes employed are not one of the famous agile processes, but IStar's own specific process. We perceived this process as an agile process because, in addition to being a lightweight process, it presents the required agile attributes [4]:

Incremental: Developers do not elicit all the requirements up front and implement the whole system at once. They start with core requirements and then implement these first. Additional features are added later, one at a time.

Iterative: An evolutionary prototype [16] is implemented and used to clarify the requirements of the system.

Self-organizing: Developers are allowed to make their own estimates and to determine how to handle the assigned work.

Emergence: There is no predetermined plan created for the projects. Only a rough deadline is estimated for each feature. The development is carried out by the technical staff, and management tracks the status of the project via frequent informal communication.

4. Maintaining Knowledge without Paper

The common opinion about documentation shared in IStar, including the management team, is that the cost of creating and maintaining the documents is not worth its usefulness. Therefore, rather than using documentation, IStar chose alternative mediums for communicating and sustaining knowledge throughout its software life-cycle. In this section we will describe the mediums utilized by IStar.

4.1 Living Documentation

Rather than focusing on creating documentation, IStar believes in using "living documents" to maintain and transfer knowledge in software projects. The living document refers to the staff members, especially "*pioneer employees*," who are technical and nontechnical staff members who have been employed for more than 20 years.

Pioneer employees play an important role in maintaining knowledge throughout the software life cycle in IStar due to their deep understanding of domain knowledge assimilated during the years. We learned of their importance for sustaining and passing along important knowledge through interviews with WA1's developers. Since WA1 is a re-implementation of an existing system with web application technology, it shares similar characteristics with its predecessor system. Some pioneer people were involved in the development and maintenance of WA1's predecessor system. They are able to provide the developers of WA1 valuable advice on requirements, data description, design decision and solution to some known issues, which were obtained from developing the predecessor system.

There is no documentation created during the development phases, so how do existing developers transfer their knowledge to newcomers when they leave the project? This situation arose with WA1, which had some student developers who left before the project stabilized. The common belief in agile projects is that comments in the source code provide enough information for new programmers to understand the application. New developers in WA1 disagree, however. What has allowed other developers to obtain the required knowledge to continue the task lies in IStar's processes for managing the departure and arrival of employees.

Typically, a developer who is leaving IStar will suspend other work and reserve the last few days only for "explaining" and "talking" to other developers whose work is related about various aspects of the tasks. Since this office community is tightly knit, developers are aware of each other's work. This lay a foundation that allows the remaining staff to understand the information left by the departing staff. This practice is so engrained among the people who work at IStar that when people leave the office on a good note they are still considered to be part of the office. They are willing to answer telephone queries pertaining to their applications, and if need be even to come back to solve some of the problems.

IStar has a philosophy that "more familiarity leads to more knowledge." When a developer first arrives in IStar, he is usually given a task to perform low priority

enhancement to the existing systems. The task allows the new employee to assimilate knowledge about the application and IStar's process. Other programmers who are aware of the assigned task "fill in" the new hire with information that is vital for the task through "informal talking" or "walk-through" discussions and meetings. This type of activity usually lasts a couple of months.

4.2 Well-Connected Communication

Like other agile projects, IStar uses communication as the primary medium for transferring and maintaining knowledge. However, what we find distinctive in the participating organization are its highly cooperative environment and the frequency of ad hoc communication among the staff. The organization's environment cultivates informal communication and a willingness to provide information and help to others. Technical and non-technical staff are on a good terms and have strong connection with each other. Due to its flat hierarchy, there are very few communication obstacles between people in different teams or even with the management team members. Each individual in the organization is approachable. Ad hoc conversation is welcomed by individuals and encouraged within the organization. Staff members are welcome to walk up to others and initiate informal communication, even with those in managerial positions.

By having frequent informal communication with non-technical staff members who are clients of the application and domain experts, developers do not have to rely on having system requirements documented. They are able to collect requirements, assimilate domain knowledge, and obtain quick feedback about the implemented system through frequent informal communication.

In addition, this connected communication facilitates knowledge transferring for various purposes, such as following up an idea presented in a meeting, distributing tasks, tracking the status of projects, and getting advice from fellow developers. It allows one staff member to be aware of and to understand the responsibilities of some of other staff, which results in overlapping knowledge, as presented in Section 5.2.

4.3 Reference Implementations and Prototyping

IStar's developers also utilize exemplar software systems as sources of information for their implementations, especially when re-implementing a system that has predecessor software. The effort spent in developing the predecessor is therefore not lost, but used in the current project. For an example, WA1's developers, with the help of pioneers, are able to use WA1's predecessors as examples for user interfaces, implementation of data processing, and some computation processes. Seeing running software also helps them to have a better understanding of business rules and descriptions of the data being processed.

Since IStar develops its software in an incremental and iterative manner, evolutionary prototypes [17] are also used to aid in communication among staff. Typically, programmers start by developing a prototype, providing core functionality of the system as a proof of concepts. Interfaces of the prototype or its screen shots are then shown to non-technical staff and fellow developers to clarify the requirement and gain design suggestions about its usability. The prototype itself is used to gain feedback and a clear requirement misunderstanding. As an example, sometimes non-technical staff members provide feedback to programmers by running the program to determine any incorrect behaviors. The management team also tracks project status by observing the current working system.

5. Success Factors

In the previous section, we described how IStar maintains and transfers knowledge without documentation. This section presents factors that enabled this organization to use these practices successfully. The reasons center around creating a positive work environment with shared values, ensuring that developers have overlapping knowledge areas, which in turn results in a low turnover rate.

5.1 Shared Values

At IStar, informal communication among peers is valued over standard documentation as a medium. All members share this value. When we asked about documentation, we were presented with the graph in Figure 1.

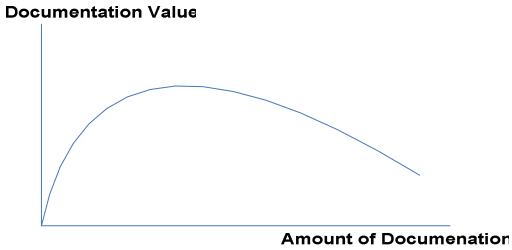


Figure 1: Amount vs. Value of Documentation

The graph shows the relationship between the value of documentation to the quantity of documentation according to IStar's staff. After a certain amount of documentation, its value decreases. The reason given was that the more documentation generated, the more time spent by existing developers to create it, and the more time needed by a new developer to read it. Another reason, given by the head of technical division, is that "a lot of people talk of creating documentation, but not many people do a good job at it." Their shared value and viewpoint on documentation cultivates informal communication and a willingness to provide information to other developers.

5.2 Overlapping Knowledge

Overlapping knowledge is key to filling the void left by the lack of documentation, for example, when a staff member leaves. Due to the closeness among the staff during their time of employment, employees discuss, explain, argue, and communicate via other informal means. In addition, various developers are involved in the same project, although at different times. As a result, overlapping knowledge areas emerge over time.

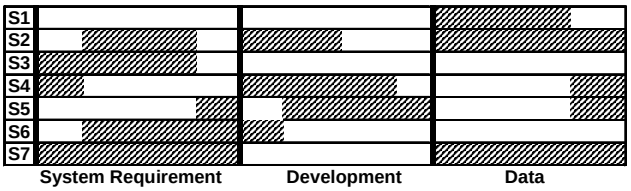


Figure 2: Overlapping Knowledge

To illustrate the overlapping knowledge areas, Figure 2 displays knowledge overlap among employees with respect to WA1. The figure shows that if any one of the employees has to leave the department permanently, people who share that employee's knowledge can combine their respective knowledge to fulfill that individual's duties. Moreover, because the employees maintain close ties with one another, if any one of the employees is leaving, the others are usually aware of this, and the last few days are spent "telling" people how they are doing and what they are doing. Their overlapping knowledge aids them in assimilating the information being passed on.

5.3 Turnover Rate

This organization has a trait that any organization would envy: a low turnover rate. Figure 3 shows that most of the employees have been working in the office for more than 10 years and some are still working even after 30 years of service to the organization.

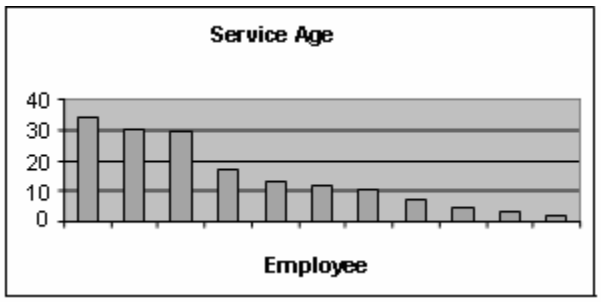


Figure 3: Service Age of Subjects

This low turnover rate aids in creating knowledge overlap among employees. Developers assimilate domain knowledge by interacting with non-technical staff members over many years. The pioneer employees can pass on their knowledge first-hand to the recent hires. As one of the non-technical staff members points out, "[she] would love to impart her knowledge if someone was ready to pick her brain."

The low turnover rate also helps to prevent the organization from losing information the staff members have. No matter how well aware the staff is toward others' responsibilities, some specific information is always lost when an existing staff member leaves the project, and it will take some time for a newcomer to pick up the knowledge. As mentioned above, the arrival and training process might take up to two months. Having a low turnover rate helps prevent the project from losing this time.

5.4 Well-Understood Requirements

Because developers are co-located with the business office and the goal of the software is to support IStar's work, developers are exposed to business logic, which helps them to grasp the domain knowledge. The non-technical team members and pioneer people who understand the business process are also there to give feedback and clarify requirements.

In addition, many critical applications have predecessors. A system is usually re-implemented to keep up with the technology; the requirements of the system behavior and workflow usually stay the same. The new system only has to mimic its predecessor behavior and computation. The system requirements were already gathered and analyzed when the predecessor system was implemented, so pioneer people who implemented it can provide this valuable knowledge of the requirements to current developers. If pioneer people forget the requirements, the working software can be run to obtain the information.

6. Moving Forward

According to a pioneer technical person, IStar has adopted this process of software development for more than 20 years. No high-level documents such as requirement or design documents were created. Informal documents created are mainly reminders, such as personal notes and online FAQs about the application's operations. It is worth noting that there is one document, "data description note," that is created and maintained regularly and is shared among developers. This note lists database fields, including their description for IStar's applications. This is not unexpected, as IStar's applications are "information processing" systems. Therefore, a description of the data to be processed is important and it is impossible, even for pioneer developers, to remember all of the data description.

With 20 years of using these processes successfully, one would expect IStar to be settled and secure with these practices. However, this is not the case. As we can see, knowledge in IStar is maintained in its staff, especially its

pioneer people. As the prospect of some pioneer people retiring is near, IStar is concerned about losing important information, such as critical application behaviors and usage, domain knowledge and various kinds of organization knowledge as these people leave. Surprisingly, they are trying to deal with this problem using documentation. Approximately five years ago, the organization initiated a documentation project by hiring an external consultant to document information from all the pioneer people. The focus of that document was on the operational aspect of the applications and business processes. However, it is doubtful that the document will be as useful in terms of completeness of information and effectiveness in distributing knowledge because interchanges among the staff, who are so used to informal communication to gain required information rather than reading. In other words, despite management concerns, this agile development group is reverting to type by favoring communication over paper.

7. Conclusion

In this paper, we reported on a field study of a small organization that has been using an agile development process for over two decades. Two successful agile software projects were examined in detail in order to investigate how information has been transferred and sustained without using traditional documentation. We found that its knowledge is maintained by using living documents, well-connected communication, and working software. In other words, scrums, or frequent, informal, and intense communication, is enough to sustain knowledge within an organization for a long period of time. Furthermore, when human memory fails, the environment provides aids to recall in the form of ingrained business processes and exemplar operational software systems.

Finally, using an agile process successfully also requires deep commitment at an organizational level. IStar is no exception. We also identified important factors that enable staff to use the current practices successfully. These factors are shared values, overlapping knowledge among team members, low turnover rate, and well-understood requirements. In summary, IStar demonstrates that with the right mechanism and a hospitable culture it is possible to use agile processes and work successfully without documentation for many years.

8. Acknowledgments

We thank IStar staff for their time and patience with our questions.

9. References

[1] M. Cohn and D. Ford, "Introducing an agile process to an organization [software development]," Computer, vol. 36, pp. 74-78, 2003.

[2] H. Svensson and M. Host, "Introducing an agile process in a software maintenance and evolution organization," pp. 256-264, 2005.

[3] J. Highsmith, Agile Software Development Ecosystems, USA: Addison-Wesley Publishers, 2002, pp. 448.

[4] B. Boehm and R. Turner, Balancing Agility and Discipline: A Guide for the Perplexed, USA: Addison-Wesley Publishers, 2003, pp. 304.

[5] S. Ambler, Essay: Agile Documenation, 2005.

[6] B. Curtis, H. Krasner and N. Iscoe, "A field study of the software design process for large systems," Communication of ACM., vol. 31, pp. 1268-1287, 1988.

[7] D.L. Parnas and P.C. Clements, "A rational design process: how and why to fake it." IEEE Transactions on Software Engineering, vol. SE-12, pp. 251-257, 1986.

[8] I. Jacobson, G. Booch and J. Rumbaugh, The Unified Software Development Process, 1999.

[9] W.W. Royce, "Managing the Development of Large Software Systems," Proc. WESTCON, 1970.

[10] S.C.B.d. Souza, N. Anquetil, K. Oliveira and thia M.de, "A study of the documentation essential to software maintenance," in SIGDOC '05: Proceedings of the 23rd annual international conference on Design of Communication, pp. 68-75, 2005.

[11] F.A. Cioch, M. Palazzolo and S. Lohrer, "A Documentation Suite for Maintenance Programmers," in ICSM '96: Proceedings of the 1996 International Conference on Software Maintenance, pp. 286-295, 1996.

[12] T. Sauer, "Using design rationales for agile documentation," in Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE 2003. Proceedings. Twelfth IEEE International Workshops, pp. 326-331, 2003.

[13] M. Visconti and C.R. Cook, "An overview of industrial software documentation practice," in Computer Science Society, SCCC 2002. Proceedings. 22nd International Conference of the Chilean, pp. 179-186, 2002.

[14] T.C. Lethbridge, J. Singer and A. Forward, "How software engineers use documentation: the state of the practice," Software, IEEE, vol. 20, pp. 35-39, 2003.

[15] A. Forward and T. C. Lethbridge, "The relevance of software documentation, tools and technologies: a survey," in Proceedings of the 2002 ACM symposium on Document engineering, McLean, Virginia, USA: ACM Press, pp. 26-33 , 2002.

[16] L. Horst, S. Matthias, Z. Heinz and llighoven, "Prototyping in industrial software projects: bridging the gap between theory and practice," pp. 221-229, 1993.

[17] C.Z. Jean and D.T. Peter, "An insider's survey on software development," pp. 178-187, 1982.

Applying Models of Technology Adoption to Software Tools and Methods: An Empirical Study

Scott A. Bailey
Department of Informatics
University of California, Irvine
baileys@uci.edu

Susan Elliott Sim
Department of Informatics
University of California, Irvine
ses@ics.uci.edu

ABSTRACT

This study examines the adoption of these tools and methods among software engineers. The study compares two models of technology adoption: Davis's Technology Adoption Model (TAM) and Ajzen's Theory of Planned Behavior (TPB). Data was collected through a combination of individual questionnaires and group interviews conducted with students in an upper division project course in software engineering. The information gathered was then used to evaluate the models to measure the importance of their identified factors as well as to create a new Hybrid Model that included the most significant factors from both models. We found that the new Hybrid Model was able to account for 45% of the variance in adoption, followed by the TAM which accounted for 39% of the variance and the TPB which accounted for 32% of the variance. The factors of Perceived Usefulness and Peer Views consistently proved to have the greatest influence on adoption over other factors such as ease of use or organizational mandate. Additionally, cost and time constraints, while not factors in any of the models, were found to have significant influence on those adopting technology.

1. INTRODUCTION

Software Engineering is an inherently complex process. Projects can often involve thousands of lines of interrelated code and multiple developers working simultaneously. As a result of this complexity software engineers have been faced with many recurring problems, for example, effectively sharing code, testing all components of a system, and compiling large sets of source code. Researchers have been striving to find ways to make software engineering more predictable in spite of these difficulties. While no silver bullet or perfect solution has been found, there have been some promising advancements in the use of tools and methods during development.

Tools and methods bring more predictability and consistency to the engineering process. Tools help to automate tasks and assist the engineer in complex activities. In so doing, tools can eliminate potential errors and cut down on the time required for certain tasks. Some examples of tools would be integrated development environments which provide an easy way for engineers to write code and software repositories which automate the sharing of code between engineers. Methods seek to organize and structure part or all of the engineering process. For example, large-scale methods that prescribe the entire lifespan of a software project or smaller methods such as regression testing. Together, tools and methods can add some standardization to software development. This standardization helps to keep software projects on time and progress more visible.

As a result of the additional predictability that tools and methods add to software development, there has been significant interest in their adoption in the field. In order to encourage this adoption one avenue of research has sought to identify the primary factors that affect technology adoption among engineers. For instance, what is more important, usability or functionality? Some have argued that poor usability can mask good functionality, while others feel that appropriate functionality is paramount. Similarly, is it enough to have managers or customers require software engineers to use a particular tool? Or are positive reviews from friends and colleagues needed as well? Last, but not least, to what extent do we need to establish an engineering culture where the use of advanced tools and methods is expected professional practice? Answers to such questions can aid the creation of tools and methods that are more likely to be adopted as well as allow for more effective support for their introduction, thereby increasing the likelihood that software engineers will adopt sophisticated tools and methods.

Past research in a variety of fields has attempted to identify factors affecting technology adoption decisions. To this end, we studied the applicability of two prominent models of technology adoption from different fields, applying them specifically to software engineering tools and methods. We examine both the Technology Adoption Model as well as the Theory of Planned Behavior. We conducted an empirical study to measure how well both models explain adoption of tools and methods and to create a new Hybrid Model which combined the most significant factors from both. The gathered data was then used to determine which model had the greatest predictive power: the TPB, the TAM, or the new Hybrid Model. In addition, the study explored possible new relationships between factors of the TPB and TAM.

2. RELATED WORK

In this section, we review a number of key studies in technology adoption. Researchers in fields such as management, psychology, and software engineering, have shown an interest in gaining a better understanding of how people choose to use or not use technology. Multiple approaches have been used to examine the process of adoption. One approach is to study decisions made by individuals regarding technology use in order to identify the primary factors that influence adoption of a technology. This type of adoption is our primary concern in this study.

Prior research has resulted in a wide range of adoption models, each identifying a different set of factors as affecting adoption. The Technology Adoption Model (TAM) by Davis was one of the first such models. The TAM was created to try to explain and predict adoption of technology within the work place. The model identified the factors of Perceived Usefulness, Perceived Ease of Use, and Attitude [1]. While results from Davis's study supported the model, further research indicated the importance of other factors [2]. Research by Sultan and Chan, for example, identified an entirely different set of factors from the TAM [3]. Here, their model concentrated on both individual and team properties rather than only the perceptions of individual users. Factors identified included team communication, leadership, and values congruent with the rest of the group. Another prevalent model in technology adoption is the Theory of Planned Behavior (TPB) by Ajzen [4]. Emerging from social psychology, this model attempts to predict behavior in general with technology adoption being one possible application for the model [5]. Identified factors included Attitude, Subjective Norm, and Perceived Behavioral Control. While each of the models identified a different set of factors, they each have been found to explain technology adoption to a certain degree, suggesting that at least a portion of each model is effective.

One approach employed to examine this large number of models has been to combine the factors from multiple model together for comparison [5]. Riemenschneider, Hardgrave, and Davis took this approach and compared the TAM, TAM2, Perceived Characteristics of Innovating Model, Theory of Planned Behavior, and the Model of Personal Computer Utilization, creating a matrix of comparable factors. Examining this matrix revealed that some of these models share common factors, some factors differing only in the name they were given in each model. Additionally, a case study considering all models found merit in unique factors of each model. This suggests it would be worthwhile to create a combination of previous models which includes both shared and unique factors of each. This is the approach that we have taken with our Hybrid Model, by combining elements of the TAM and the TPB.

3. ADOPTION MODELS

For this study two adoption models are used in the context of tools and methods adoption: the TAM and the TPB.

3.1 TAM – Technology Adoption Model

The TAM was originally created by Davis et al. for predicting user acceptance of technology [6]..The model has been referred to widely in the literature, both in confirmation of its findings and as the basis of future research [2, 7, 8].

The TAM identifies three factors which contribute to user adoption: Perceived Usefulness, Perceived Ease of Use, and Attitude. Perceived Usefulness is defined as the "the belief that using a particular system would enhance a persons job *performance*" [italics added] [7]. Perceived Ease of Use is defined as "the degree to which a person believes that using a particular system would be *free of effort*" [italics added] [7]. The final factor, Attitude, is a mediating variable which refers to a person's positive or negative feelings about using a particular system.

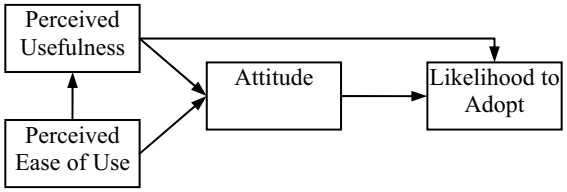


Figure 1: Technology Adoption Model

3.2 TPB – Theory of Planned Behavior

The TPB is the second model being analyzed in this study. Unlike the TAM, the TPB does not explicitly deal with technology. Originating in social psychology, the model's primary purpose is to explain, as well as predict, behavior of people in general. The model has had many applications, of which explaining the behavior of technology adoption is one.

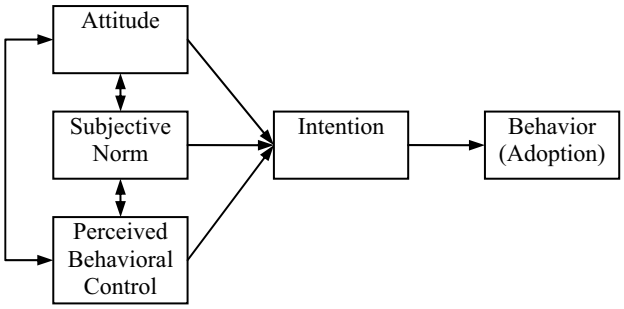


Figure 2: Theory of Planned Behavior

Overall, the TPB looks at the "behavioral disposition" of people in order to predict their future actions [4]. However, only relying on general disposition of a person towards an action has proved to be a poor indicator of future action. As a result, the TPB looks at a set of four factors to more clearly define this disposition towards an action [4]: Attitude, Subjective Norm, Perceived Behavioral Control, and Intention.

Attitude refers to the degree to which person has a favorable or unfavorable view of a particular behavior. Subjective norm refers to the perceived social pressure to perform a certain behavior. Perceived Behavioral Control is defined as the perceived ability to perform a given action. Perceived Behavioral Control can be further divided into two separate subcomponents, Internal and External. In the case of software tools and methods, External Behavioral Control would refer to outside forces influencing a user's ability to perform an action, such as mandates or rules, while Internal Behavioral Control would refer to internal factors such as experiences or memories influencing perception. Finally, Intention is a mediating variable of the above three factors

and is used to measure the degree to which a person plans to perform a given action [4].

4. METHOD

The study was conducted by administering a series of interviews and questionnaires to students enrolled in two separate upper division software engineering project classes. One class answered questionnaires while a second class participated in interviews. These classes were chosen as they were representative of real world software development. Students were formed into teams then given contact information for some outside organization that needed software developed. From this point on it was the students' responsibility to deliver the desired product within a ten-week time span. Students were given free choice for choosing a set of tools and methods. We employed the two different means of data collection in order to gather a broader range of information. The questionnaires were intended to gather statistics on the tools and methods that students used and whether they eventually adopted them. This provided the information needed to test the predictive power of the three different models and determine the significant factors. The interviews, on the other hand, were intended to gather only supplementary information and bring to light any considerations that might have been omitted from the questionnaires. An electronic copy of both the questionnaires and the interview questions is available at http://www.ics.uci.edu/~ses/research/techadoption/ .

4.1 Questionnaires

The quantitative data for this study came from the questionnaires conducted within one of the classes. Four questionnaires were administered to individual students throughout the ten week course. The questions were aimed at discovering individual's choices for tools and methods, their various perceptions of them, and whether each tool or method that they choose was eventually adopted. Questions were asked for each specific tool or method that the student choose to use. Four questionnaires were given at throughout the class in order to observe how an individual's perceptions and use of tools and methods changed over time. A total of 19 out of 42 students (45%) agreed to answer the questionnaires with 9 students answering all four. Each student used 3-10 tools and methods during the project, making for a total of 54 adoption decisions to consider.

4.2 Interviews

Qualitative data was obtained through interviews with teams of students. A different data collection technique was employed with a second equivalent group to triangulate the phenomenon. Two interviews were conducted near the end of the course with students from the second section of the course. Interviews centered on the tools and methods chosen by a team, experience they had with them, and any planned future use. The interviews were planned mostly for supplementary information on student's views on tools methods in comparison to any data obtained from the questionnaires in the first class. In all, 6 out of the 10 teams (60%) participated in the interviews.

4.3 Threats to Validity

Some threats to validity arise from our research design. One consideration is that our participants are students rather than engineers in the field. The question that stems from this is whether our data will accurately reflect adoption behavior of software engineers in general or only students. In this case, we feel that the classes under question are close enough to simulating real world experience to make results from our data applicable to software engineers in general. A second consideration to our data is the possibility of a sampling bias. Since not all of the students in the class responded to the questionnaires or interviews the possibility remains that only certain types of students participated. In this case were able to examine the final class grades of students in the class and found that participants did represent a random sampling of the class in terms of class grade.

4.4 Analysis

Each of the questions from the questionnaires was mapped to a specific factor identified in one of the models. User responses to each question were given a rank between one and five to represent the significance of that factor for the user. A rank one response represented a factor that was not at all significant in affecting adoption while a rank five response represented a factor that was very significant. In cases where multiple questions mapped to the same factor, the ranks were averaged in order to determine the factor significance. Relationships between factors were then determined using multivariate logistic regression. The R^2 statistic shows the amount of variance in the dependent variable that is accounted for by the independent variables. The Beta statistic is the coefficient for the dependent variable in the equation. It shows the strength of the relationship and can be conceptualized as the slope of the regression line for that dependent variable.

5. RESULTS

To measure the explanatory power of each model we examined the amount of variance in adoption explained by each. In this way the explanatory power of each model could be ranked against the others. The equations representing these results are shown in Table 4, below.

We found that out of the two models, the TAM was the best fit for the data and accounted for 39% of the variance in adoption. The TPB followed and accounted for 32% of the variance. However, after examining the data we were able to create a new Hybrid Model, discussed below, by combing the most significant factors from both models. This new Hybrid Model was able to account for 45% of the variance in adoption, more than either the TAM or TBP. In the remainder of this section, we will discuss the statistical results and significant factors for each of these models in detail.

Table 4: Summary of Statistical Results

TAM		
Equation	**R^2**	**Beta**
Adoption = PU + PEU	.396	
PU		.585***
PEU		.127
TPB		
Equation	**R^2**	**Beta**
Adoption = A + SN + PBC	.328	
A		.396**
SN		.362*
PBC		-.081
Hybrid Model		
Equation	**R^2**	**Beta**
Adoption=PU + SN	.453	
PU		.552***
SN		.234*

Key	
PU: Perceived Usefulness	PEU: Perceived Ease of Use
A: Attitude	SN: Subjective Norm
PBC: Perceived Behavioral Control	

* p < .05 ** p <.01 *** p < .001

5.1 TAM Results

The TAM was able to account for 39% of the variance in adoption of tools and methods among the students. Results showed Perceived Usefulness (β=.560 p<.001) as being the most significant influence on Adoption. The influence of mediating variable Attitude, however, was not found to be significant. Perceived Ease of Use was also not found to have a significant relationship with either Attitude or Adoption.

These results point towards the importance of Perceived Usefulness in influencing adoption of tools and methods. Perceived Usefulness, by far, showed the strongest relationships in the model, while the other factors of Attitude and Perceived Ease of Use showed very little significance. This suggests that software engineers may be willing to put up with more difficulties associated with learning or using a tool if it is believed to be useful.

5.2 TPB Results

The TPB accounted for only 32% of the variance in Adoption. The most significant factors within the model proved to be Subjective Norm (β=.362, p<.05) and Attitude (β=.396, p<.01). The third factor of Perceived Behavioral Control did not have a significant influence on Adoption.

The TPB results show the factors of Subjective Norm and Attitude as being as being the most influential when deciding adoption of software tools and methods. In the case of Subjective Norm, this implies that social pressure can have a big effect on software engineers' adoption. For example, having the majority of software engineers in an office use a certain tool could pressure engineers to adopt the same tool. The results also suggest that this social influence from peers can have more effect than mandates from managers or customers requiring tool or method use, as represented by Perceived Behavioral Control. Continuing the previous example, this means that the social pressure from the other engineers could be enough overrule a direct requirement to use a different tool.

5.3 Post Hoc Analysis: Hybrid Model

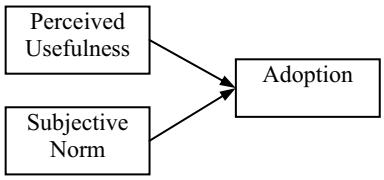


Figure 3: Hybrid Model

Using the data collected from the analysis we were able to construct a new Hybrid Model which combined the most significant factors from the TAM and TPB. As suggested by Davis et al. "combining the beliefs of the TRA (an earlier version of the TPB) and TAM into a single analysis may yield a better perspective of the determinants of adoption" [1]. The new Hybrid Model is composed of two primary factors: Perceived Usefulness from the TAM and Subjective Norm from the TPB. Earlier versions of the Hybrid Model also included Attitude from the TPB, however, it was later removed as no significant relationship with adoption was found. This Hybrid Model was able to account for 45% of the variance in adoption, more than either the TAM or TPB alone. Perceived Usefulness and Subjective Norm were both found to have a correlation with adoption of β=.628 (p<.001) and β=.266 (p<.05) respectively.

In addition to reinforcing the importance of Usefulness, the Hybrid Model displayed a strong relationship between Subjective Norm and Adoption. That is, hearing positive things about a tool or method from peers made people more likely to adopt. This tendency was observed during the group interviews as well. One team commented, "Oh, for SmartDraw… somebody told us about it because they used it in their project… [he] showed us the functionality and how there's pre-made templates… we saw that and kinda thought it would be easier to just use that." For them, Peer Views in combination with the functionality were responsible for their adopting decision.

6. DISCUSSION

6.1 Models of Adoption

The results from the study showed the Hybrid Model as being able to explain the greatest amount of variance in Adoption and thus having the greatest explanatory power. Following the Hybrid Model in effectiveness was the TAM and finally the TPB. Throughout the results of all three

models, the significance of individual factors remained consistent. The most significant factors proved to be Perceived Usefulness and Subjective Norm, On the other hand, the factors of Perceived Behavioral Control (both Internal and External) and Perceived Ease of Use did not show any significant relationships with Adoption in any of the models.

This brings us to the question of why the Hybrid model was able to explain a greater amount of variance than either the TAM or TPB. From a statistical viewpoint, this greater explanatory power can be traced back to its inclusion of both Perceived Usefulness and Subjective Norm the two factors with the strongest correlations to adoption in the TAM and TPB. Perceived Usefulness was the most significant factor in the TAM while Subjective Norm was the most significant factor in the TPB. The Hybrid Model contains both of these factors together, in a sense, taking the best of both models.

6.2 Relationships Between Factors

In addition to measuring the effectiveness of the three models we explored possible correlations between previously separate factors from the TAM and TPB. One of the most noticeable aspects from our findings is the strong negative correlations that External Perceived Behavioral Control (EPBC) had with the other factors. In the case of software development, EPBC would be the equivalent of mandates or organizational rules requiring the use of certain tools or methods. EPBC was found to have a significant negative correlation with Internal Perceived Behavioral Control (IPBC) ($r=-.725$, $p<.001$), Perceived Usefulness ($r=-.491$, $p<.001$), and Subjective Norm ($r=-.393$, $p<.001$).

Table 5: Hybrid Model Correlations

		EPBC	PV	PU	PEU	IPBC
EPBC	r					
PV	r	-.393**				
PU	r	-.491**	.365*			
PEU	r	-.007	.284	.253		
IPBC	r	-.725**	.371*	.407**	.001	
Adoption	r	-.178	.435**	.617**	.275*	.191

* p <0.05 level (2-tailed) ** p< 0.01 level (2-tailed)

IPBC can be thought of as experience and was measured as familiarity with a large number of tools. The strong negative correlation with IPBC suggests that EBPC may be strongly influenced by how much software engineering experience someone has. That is, for those with less experience the requirement to use a particular tool or method creates a greater conflict since their current set of known tools and methods is relatively small. Thus, they perceive that they are being forced to do something out of their usual routine whereas those with more experience have learned many new tools or methods in the past and do not feel as if they are being forced to use something new. This interpretation is consistent with a correlation between Internal Perceived Behavioral Control and Perceived Usefulness of $r=.407$ ($p<.001$).

6.3 Constraints on Adoption

In the group interviews, students brought up additional factors that were not originally considered in either of the models, specifically time and cost constraints. Some teams expressed interest in using and adopting certain tools and methods, but due to these constraints they were forced to look at other options. Consequently, some teams adopted tools and methods that the majority of members already knew or had a short ramp up time. One team, looking at alternatives for future development would have made a different decision if more time was available, they commented, "I would say to go the other way, the Eclipse and CVS way, but… [it]… depending on the amount of time… if you have more time to setup all the stuff… or if it were already preset for you then… that would be a different issue… so if we didn't have time, we'd definitely use the same tools… because it let us do what we needed to get done fast." The second factor that the interviews revealed was cost constraints. In searching for different tools to use, students would often select tools that were free or made available to them through the school. Students would recognize other tools might be better, but would go with what they could easily afford.

While these constraint factors were not in any of the models, they did prove to play an important role in adoption for students. For one team being interviewed, these were deciding factors for adoption. When asked why they used the tools they did, they responded "…mainly because they are free... and we already knew them. Yeah, so cost and previous experience were definitely important to us." While the time and cost constraints here were a result of the classroom setting, these factors can also exist in a business settings. Developers are often under pressure to deliver a product to their end customer as quickly as possible while staying under a budget. These factors should be considered for inclusion in future models or revisions to the Hybrid Model.

6.4 Application to Software Development

The results of this study have some interesting implications for actual software development. One of the most interesting discoveries in this research was that students were found to be very tolerant of the difficulty in using tools and methods and were not influenced significantly by external control. These findings bring up the question of how adoption of tools methods might be improved in the future.

Perceived Usefulness was found to much more significant to adoption in comparison to Perceived Ease of Use. Students greatly valued the usefulness of a tool or method whereas the ease in learning and using that tool or method was not as important. That is, students were willing to work through difficulties using a tool or method as long as it was able to complete the task at hand. The relationship between

Perceived Ease of Use and Adoption in our study also appears to be weaker than that observed in previous studies involving non-software engineers [6]. This suggests that software engineers in particular, as builders of new technology, are willing to put up with technology that is not as easy to use. As one of the student groups suggested when questioned about the difficulty of using their tools, "we're all fairly computer savvy people, we could pretty much figure out any tool." This finding is consistent with Berlin's observation that experienced software engineers were willing to use a specialized software tool in order to utilize specific powerful features. They were willing to face a steep learning curve and acquire skills to use the tools, so they could facilitate complex sub-tasks or automate error-prone ones [9].

Another interesting aspect of the results is the lack of significance of mandates and organizational requirements as represented by External Perceived Behavioral Control. Contrary to original expectations, mandates had no major effect on adoption of tools or methods among the students. However, peer views, as represented by Subjective Norm, did play an important role. This brings into question the effectiveness of simply dictating use of a particular tool or method, even in the field. For example, when trying to migrate to a new tool within the workplace, company policy alone may not be enough to ensure that engineers begin to use it. Other means, such as highlighting the features and effectiveness of a tool or method or its prominence in the engineering field could be useful in promoting a tool to engineers to increase the likelihood of adoption.

7. CONCLUSION

The goal of our study was to identify some of the significant factors that contribute to the adoption of tools and methods so that in the future better tools and methods might be developed. To this end, we examined two prominent, preexisting technology adoption models to see the factors that had already been identified. Concentrating on the TAM and TPB, we created an empirical study by following two upper division software engineering classes to test the predictive power of both. The resulting data was used in the creation of a new Hybrid Model which was built out of the most significant factors observed in the TAM and TPB. This Hybrid Model was found to be the most effective of the models, explaining 45% of the variance in adoption.

Perceived Usefulness and Peer Views consistently proved to be the significant factors in Adoption throughout the models. Respondents were more likely to adopt tools and methods that had functionality that they needed and that their peers thought were worthy of using. In other words, functionality was far more important than usability. Positive opinions as well as positive expectations were key factors in adoption intentions.

Our study interviews also discovered the importance of cost and time constraints, two factors not previously identified. When there's not enough time, software engineers use what they know; when there's no money, they use what's free. These factors can cause a software engineer to choose a technology that is sub-optimal for the job. The learning curve for a new tool or method is too daunting when deadlines are looming. Similarly, when there are no resources to purchase or support new tools, software engineers must work around this constraint as well.

Finally, the results showed interesting negative correlations between Mandate and Adoption, bringing into question the effectiveness of using company policy to bring about a change in tool or method use for software engineers. Combined with the findings regarding Peer Views and Attitude, this result suggests that a change in policy is more likely successful when championed by a fellow software engineer or the technology being advocated and liked by the broader community.

8. ACKNOWLEDGMENTS

This research was generously supported by a grant from the University of California, Irvine Undergraduate Research Opportunities Program (UROP). We would like to thank Professors André van der Hoek and Hadar Ziv for making their classrooms available to us and to all the students who generously gave up their time to participate in this study.

9. REFERENCES

[1] F. D. Davis, "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Infomation Technology," *MIS Quarterly*, vol. 13, pp. 320-340, 1989.

[2] H.-d. Yang and Y. Yoo, "It's all about attitude: revisiting the technology acceptance model," *Decision Support Systems*, vol. 38, pp. 19-31, 2004.

[3] F. Sultan and L. Chan, "The Adoption of New Technology: The Case of Object-Oriented Computing in Software Companies," *IEEE Transactions on Engineering Management*, vol. 47, pp. 106-126, 2000.

[4] I. Ajzen, "The Theory of Planned Behavior," *Organizational Behavior and Human Decision Processes*, vol. 50, pp. 179-211, 1991.

[5] C. K. Riemenschneider, B. C. Hardgrave, and F. D. Davis, "Explaining Software Developer Acceptance of Methodologies: A Comparison of Five Theoretical Models," *IEEE Transactions on Software Engineering*, vol. 28, pp. 1135-1145, 2002.

[6] F. D. Davis, R. P. Bagozzi, and P. R. Warshaw, "User Acceptance of Computer Technology: A Comparison of Two Theoretical Models," *Management Science*, vol. 35, pp. 982-1003, 1989.

[7] F. D. Davis, R. P. Bagozzi, and P. R. Warshaw, "Perceived Usefulness, Perceived Ese of Use, and User Acceptance of Information Technology," *MIS Quarterly*, pp. 320-340, 1989.

[8] D. L. Moody, "The Method Evaluation Model: A theoretical Model for Validating Information Systems Design Methods," presented at 11th European Conference on Information Systems, ECIS 2003, Naples, Italy, 2003.

[9] L. M. Berlin, "Beyond Program Understanding: A Look at Programming Expertise in Industry," presented at Empirical Studies of Programmers: Fifth Workshop, Palo Alto, CA, 1993.

Measuring the Usability of Online Stores

Dr. Ernest Cachia, Mr. Mark Micallef
Software Engineering Process Improvement Research Group
Department of Computer Science and Artificial Intelligence
University of Malta
ernest.cachia@um.edu.mt, mmica01@um.edu.mt

Abstract

As online shopping turnover continues to increase year after year [1], more and more business are rushing to establish an online presence. In a previous publication [2], the authors of this paper identified usability as being one of the attributes which most influence the success of an e-commerce systems. This paper explores the issues involved in measuring the usability of e-commerce systems and presents a usability measurement framework specifically tailored to online stores. The framework provides a quick way to evaluate the usability of online stores through easily automatable mathematical analysis of a site's structure and visual layout, followed by a short checklist to assure that essential usability criteria have been met. References are made to a number of research exercises carried out by the authors amongst online shoppers and case studies of well-known e-commerce sites are also presented.

Keywords: *Software Metrication, Software Quality Assurance, Usability, Navigability, E-Commerce*

1 Introduction and Terminology

In the years since the dot-com crash, online sales figures have recovered and continue to increase annually across the globe. As a result, business are finding it increasingly necessary to have an online presence so as not to loose their proverbial slice of pie. However, having an online presence by no means guarantees commercial success. The online business environment can be unforgiving. Research reveals that it only takes users a few seconds to decide whether they stay in a particular online store or move on to a competitor [11]. The authors of this paper carried out a survey amongst 350 online shoppers and discovered that a substantial number of transactions were being abandoned by users due to usability issues. 35.6% of users who have abandoned transactions mid-way through, have done so because of usability problems. This is a worrying statistic and raises the issue of how the usability of an online store can be evaluated or better still, measured.

The International Standards Organisation (ISO) defines usability as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use" [5]. In terms of online stores, this could arguably be translated into "the extent to which an online store can be used by shoppers to find and purchase products or services with effectiveness, efficiency and satisfaction". Quite a number of guidelines have been published to help designers create usable websites [8][9]. Further more, work has also been done to provide guidelines for usability in specific types of websites (e.g. hotels and travel sites) [6][7]. In a previous publication [2], the authors of this paper showed that although e-commerce systems were a subset of web applications in general, they had enough distinguishing attributes to merit specialised treatment in a number of situations. It is the opinion of the authors, that generic usability guidelines and metrics can, and should, be fine-tuned and tailored to e-commerce systems.

In this paper, an integrated usability evaluation framework specifically tailored for online stores is being proposed.

2 Criteria for a Usability Framework

A good way to kick-off the development of such a framework would be to identify a number of criteria which the framework should satisfy. In a research ex-

ercise carried out by the authors, ten local e-commerce solution development companies were approached and questioned about a number of issues [4]. One interesting outcome of this exercise was that most e-commerce systems are developed in a period of six months or less, which ties in with standard RAD expectations. In fact, 60% of participants claimed that their systems were developed in a three month period. This leads us to the first criteria which states that the framework ***must be easy to utilise and consume as little time as possible***, so as to conform to the RAD model.

The term "web design" inherently implies that there is a substantial amount of creativity involved in the process. Therefore, a usability evaluation framework must ***provide objective guidelines without overly restricting designers' creativity***.

Finally, one can argue that such a framework **need not be overly thorough and accurate**. This follows from two main trains of thought. Firstly, a usability measurement framework that involves lengthy processes will be more of a hinderance than a help. Secondly, trying to precisely measure and quantify the usability of a system is not of much value when one considers the basic way in which humans make decisions. Many designers assume that users will scan a page, consider all available options, and choose the best one [8]. However, in actual fact, people tend to use a strategy known as *satisficing* [16] (a cross between satisfying and sufficing). In this strategy, users will tend to go for the first reasonable options and discard other options which they may not have yet considered. This behaviour was also observed by a number of detailed studies into human decision making carried out by Klein [17] at MIT. Hence, it can be argued that it is desirable to have a quick and sufficient method of usability evaluation even though a more lengthy analysis would provide more accurate results.

3 Current Evaluation Techniques

There are two main techniques for evaluating usability of websites. The first of these involves the use of **checklists** whereby engineers can crosscheck a site against a number of criteria before signing it off for release. Criteria may include items such as "*There is a clear and consistent navigation scheme*", "*There is an easy to use search facility*", and so on. Checklists are extremely helpful but can become tedious to use if they get too long without some form of automated and cross-linked checking being developed.

Another popular method for evaluating the usability of a system is **usability testing** [10] whereby a representative sample of the system's target users are given access to it and asked to perform a number of tasks. Feedback from the participants is then analysed and if necessary, changes are made. This method, although effective, could be time consuming and expensive to organise, especially if multiple usability tests are required before a system is approved.

In this paper, a short checklist-based approach, largely based on mathematical analysis techniques which can be easily automated is presented. This is aimed reducing the time and expense required to evaluate the usability of a system before release.

4 Framework Overview

As depicted in figure 1, it is being proposed that the framework consist of three elements. The first of these involves *analysing the system from a structural point of view*. The way in which pages are linked together throughout the site will affect the ease with which users can navigate and perform tasks on the site. Secondly, key pages in the site are individually analaysed with respect to their visual layout. Finally, a checklist is provided which checks the site against a number of criteria, including checking values from the other two elements of the framework.

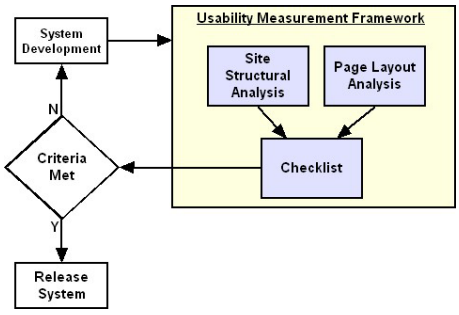
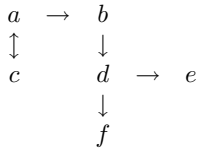


Figure 1. The proposed framework

5 Structural Analysis

Results from our survey of online shoppers showed that 30% of participants chose a site's navigability as their primary first-impression factor. This effectively means that a site's navigability has a potential of attracting or repelling 30% of a site's first time users.

Structural analysis deals with drawing navigabilty evaluations from the way pages in a site are linked together. At first glance, one may be tempted to use metrics that count the number of pages and links in a website, maybe even deriving a ratio of the two. However, these types of metrics do not really provide any useful information. Instead, it would be better to liken a website to a directed graph and as such be able to analyse mathematical properties of that graph. The example below shows a 6-page website being converted to a graph by encoding it into a converted distance matrix.



	a	b	c	d	e	f
a	0	1	1	2	3	3
b	6	0	6	1	2	2
c	1	2	0	3	4	4
d	6	6	6	0	1	1
e	6	6	6	6	0	6
f	6	6	6	6	6	0

Note that a value of 6 was chosen to represent the distance between unreachable nodes. The logic behind this is that in a graph of n nodes, the maximum distance between any two nodes is n-1. Therefore n can be used to represent infinity. There are a number of issues surrounding the choice of this value but they are beyond the scope of this paper.

In 1992, Botafogo [12], proposed a number of metrics derived from the converted distance matrix which could be used to analyse hypertexts. Although this was very interesting work, it stopped short of recommending ranges of acceptable metric values which hypertexts should exhibit. Part of the framework proposed here will utilise three of Botafogo's metrics and based on a number of studies, recommend ranges of acceptable values which e-commerce systems should exhibit. The following metrics will be utilised:

Compactness analyses the level of connectedness of a website. A fully disconnected site has a compactness of 0 whilst a fully connected site (i.e. every page has a link to every other page in the site) has a compactness of 1.

Stratum gives an indication of the hierarchical ordering of a graph. The values of stratum can vary between 0 and 1 whereby a completely linear graph has a stratum of 0.

Relative Out Centrality is an indicator of the social importance of individual nodes in a graph. This metric can be very useful for identifying candidate root nodes in a website and thus checking that your actual chosen root node exhibits the ideal mathematical properties to merit its root status. Although this metric is not bounded by a maximum value, it is normalised to the size of the site/graph in question so as to make it possible to compare metric values across different sites.

The authors of this paper are also proposing a metric called the **Depth Distribution** metric. This metric counts the number of nodes at each hierarchical level starting from the root node (level 0). During a research exercise carried out by the authors, it was observed that users started becoming increasingly irritated if they had to click more than 3 times when trying to find a particular product. The Depth Distribution metric will help identify pages which are too far away from a the root. These pages can then be analysed and the site could possibly be reorganised to provide a better navigation scheme.

5.1 Recommended values

As discussed in [2], e-commerce systems are a particular class of web applications with a number of distinguishing characteristics that justify individual treatment when it comes to a number of issues, including usability. To this end, two research exercise were carried out to identify ideal metric values for e-commerce systems. The first consisted of a research study amongst 78 participants age between 18 and 55. The exercise consisted of three parts. The first involved asking participants a number of questions regarding usability issues. Following that, participants were asked to perform some tasks on a number of mock e-commerce websites which although sold the same items, were organised in different manners so as to exhibit varying levels of compactness and stratum. The final part of this research exercise involved a similar task whereby users were asked to give feedback about the visual layouts of key pages in a number of purposely designed mock websites. However, this will be discussed in section 6.1 below. A second research exercise involved calculating compactness, stratum and depth distribution metrics for top-ranked e-commerce sites [15]. This was done in an attempt to establish a link between successful websites and particular value ranges.

Although Botafogo [12] commented that high levels of compactness were likely to invoke symptoms of the so

called *Lost in Hyperspace Syndrome*, it was observed that participants in our survey were more comfortable with sites that exhibited high levels of compactness. In practice, high compactness results in being able to reach any other page in a site in a minimal number of clicks (typically 3 to 5). This is a valid assumption in most e-commerce systems due to the use of persistent global and local navigation schemes which enable to user to jump from any page in the site to a "hub" page (a page which is in close proximity to many other pages) such as the root node or a particular product category. From this hub, the user can then reach his target page in a relatively quickly. Participants consistently approved of sites which had compactness readings of 0.75 or higher. Analysis of popular e-commerce sites such as Amazon.com, LLBean.com and BarnesAndNoble.com also revealed that these sites exhibited high level of compactness. It is therefore being recommended that engineers building online stores strive to attain levels of 0.75 or higher.

With regards to stratum, many of the users who were asked to find products in sites with low stratum readings exhibited signs of irritation because they were forced to navigate through the linear structure of the sites. Participants seemed to be most comfortable with sites that exhibited a stratum of 0.7 or higher. Even sites with stratum values of 0.65 evoked negative reactions. With regards to the analysis of successful e-commerce websites, the trend held true in that most websites exhibited values of stratum hovering around 0.75. However, limited numbers of exceptions where observed where a successful site exhibited lower values of stratum.

Finally, depth distribution is discussed. A number of observations were made in this regard. Firstly, participants got increasingly irritated when having to go further than 3 clicks to find a product. Secondly, it was observed that although the successful e-commerce sites consisted of thousands of pages, they manage to employ navigation schemes that make it possible to reach any other page within 3 to 5 clicks. It is being recommended that a site have an ideal maximum depth of 3 with values of 4 or 5 being deemed acceptable. This leads us to propose that any pages with a depth greater than 5 should be reexamined in an effort to better organise the site's structure or ascertain that the deep position of such pages is necessary in the given circumstances.

6 Page Layout Analysis

Besides the navigability of a site, users are also affected by the visual layout of individual pages in the site. One might argue that visual layouts lie purely in the realm of creativity and cannot be measured. However, studies of human psychology that humans are attracted to familiarity [13]. Although the human being is attracted to innovation and changes, he is more likely to be lost when presented with something that is completely new or radically different to what is expected [14]. In the context of e-commerce systems, this implies that a potential shopper may find it easier to switch to a competitor's website (which offers a more familiar layout) instead of taking the time to understand a new layout. It therefore follows that designer creativity is important to keep user interest high but should not be allowed to impact the system's intended functionality. In fact, if one examines a number of successful e-commerce websites, they are all different but yet seem similar in many ways. Most have a search facility somewhere near the top of the page; most display small product images towards the centre; most offer some form of categorisation scheme for their products.

When it comes to measuring the visual attributes of a webpage, it is being proposed that the framework utilise a content distribution metric similar to the one used by Nielsen[9] in one of his numerous publications about web usability. This metric analyses visual layout in terms of the amounts of screen space used by different types of content. In his approach, Nielsen used the metric on generic websites and did not make any recommendations about desirable content distributions. To this end, the authors of this paper wanted to explore the relationship between values of content distribution metrics and the usability of a e-commerce systems (as distinct from generic websites). This was approached by a research study amongst online shoppers whereby participants were shown a number of websites exhibiting different screen content distributions and asked to give feedback. Furthermore, successful e-commerce sites were analysed in an effort to link success to screen content distribution. Results are discussed in section 6.1 below.

6.1 Results and Recommendations

During the above-mentioned studies, screen content was classified as being one of 5 possible types. These are described by example in the table below:

Content Type	Examples
Navigation	Navigation and search schemes
Product Information	Description, User comments, Price, etc
Advertising	Sales, offers, etc
Whitespace	Unused space

Whilst discussing site identity, 50% of participants where comfortable with sites where site identity consumed 25% of screen space. However, when questioned further, most conceded that this was purely an aesthetic decision and it may be more practical to free up space for use by other screen content. The other 50% of participants preferred sites where site identity consumed up to 5% of screen space. An analysis of successful sites revealed that the vast majority (71%) of them used between 1% and 2% for site identity. CrateAndBarrel.com was the site from the top-10 list [15] which used most screen space for site identity (10%). It is being recommended that a site use a maximum of 5% of screen space for site identity.

Navigation is an important factor for e-commerce users. While using sites with varying levels of navigation space, participants consistently approved of sites which used between 15% and 35% of screen space for navigation. However, beyond this point, reactions varied. Only 20% of participants remained positive about a site when navigation space exceeded the 35% mark. Comments were made that aesthetics did play a part in this decision and if a designer could make the navigation visually appealing without cognitive overload [18], higher levels of navigation would be acceptable. Analysis of popular sites revealed that except for CrateAndBarrel.com (due to its relatively large usage of space for site identity) all sites on the list used more than 25% of the screen for navigation purposes. Most hovered around the 25% to 33%, however LLBean.com and Art.com utilised 47% and 86% of their screen respectively. This may seem like an overkill but the aesthetic layout of the page was non-intimidating and pleasant to look at. As a general guideline a navigation usage of 25% to 35% is being recommended but larger usage would be acceptable if supported by approval from usability tests involving a sample of the site's intended users.

When considering the balance between adverts and product content, it surprising to discover that users did not mind adverts overpowering content on a site's home page. This was mainly due to the fact that participants were attracted to adverts in the hope of being told about sales, discounts, new products and so on. However, when participants venture deeper into the site, they expect most of the space left over after navigation and site identity to be consumed by useful information. This trend was substantiated by analysing successful e-commerce sites. Apart from Art.com (which utilised 86% of screen space for navigation purposed), all sites used an average of 41% for advertising on their home page. However, as the user progresses to pages with information about products, this space is almost exclusively utilised by product information and adverts decrease substantially. It is therefore being recommended that 25% to 55% of screen usage be dedicated to a mix of content and advertising. The emphasis on the homepage of a site should be on advertising since the user is still trying to decide where to go. However, once the user leaves the homepage, product information should be given much more prominence.

The last aspect to be analysed was whitespace. When questioned about unused space, there was a consensus amongst participants in our study that whitespace is important in e-commerce websites due the fact that using up all the available space would substantially decrease the ease by which such content could be assimilated. Analysis of successful e-commerce sites revealed that on average, sites maintained 18% of unused space. Therefore, based on data from our survey and site analysis, it is being recommended that designers consciously leave between 15% and 25% of screen space unused when designing e-commerce sites.

One must keep in mind that these are simply guidelines based on studies and statistical data. As such, they are not meant to restrict designer creativity since it may indeed be possible for someone to violate these recommendations and still provide a usable e-commerce system just as a cook might choose to deviate from standard recipe yet still produce an exquisite dish. However, these recommendations could be used to identify potential problems and single out particular issues and decisions for reexamination.

7 The Checklist

Once the structure of a store's website and the visual layout of its key webpages have been analysed, the framework provides a checklist which the designers can go through as an indication of the site's usability. The checklist consists of eleven criteria which should be met before a site is released. The the criteria

presented here stem from the body of work represented by this paper together with work from another publication by these same authors [2]. However a detailed treatment justifying their extraction cannot be included in this paper due to length restrictions. Instead, this will be published in a separate technical report. The proposed checklist is as follows:

Criteria	Condition
Compactness	≥ 0.75
Stratum	≥ 0.7
Depth Distribution	$3 \geq x \leq 5$
Choice of homepage	Verified using ROC [12]
Screen Content Usage	As discussed in section 6.1
Search Facility	Exists and effective
Navigation Scheme	Clear and consistent
Screen Layout	Liquid or fixed to 800×600
Product Photos	Required
Visual Layout	As discussed in section
Product information	Suitable detailed for target audience

The authors of this paper feel that affective modern e-commerce website development should address all eleven criteria. It is felt that if any of these criteria is left unconsidered, the resulting e-commerce site might suffer from substantially decreased usability and consequently a loss of potential revenue.

8 Conclusions and Future Work

This paper proposes a usability measurement structure substantiating a checklist to be used within a global e-commerce development framework currently being developed by the authors [4]. The final scope of such a framework would be to enable engineers to develop high quality e-commerce systems in short time-spans in the spirit of extreme software engineering practices.

References

[1] Emarketer.com, "Online Retail Update: Latest Q4 Quote", www.emarketer.com/news/article.php?1002631, Emarketer.com, 2004

[2] Cachia E.,Micallef M., "Towards Appraising Online Stores", Software Engineering Knowledge Engineering (SEKE) Conference, 2004

[3] Cachia E., Micallef M., "A Functionality Measurement Framework for E-Commerce Systems", IEEE Computer Applications Software Applications Conference, 2005

[4] Cachia E., Micallef M., "Towards a RAD Framework for E-Commerce Systems", International Conference on Web Information Systems and Technologies, 2006

[5] "ISO/IEC 9126:2001 - Software Engineering Product Quality", ISO, 2001

[6] Bainbridge, A. Hotel booking process: Design and Usability, http://www.travelucd.com/research/pdf/ hotel_booking_process_february2003.pdf

[7] Sater D., Patterson P., "Usability Study of Travel Websites", Journal of Usability Studies, 2005

[8] Krug S., "Don't make me think!", New Riders, 2005

[9] Nielsen J., Tahir M., "Homepage Usability: 50 Websites Deconstructed", New Riders, 2001

[10] Prescott J., Crichton M., "Usability testing: a quick, cheap, and effective method", 27th annual ACM SIGUCCS conference on User services: Mile high expectations, 1999

[11] Dustin E. et al, "Quality Web Systems", Addison Wesley, 2001

[12] Botafogo R.A., Rivlin E., Shneiderman B, "Structural Analysis of Hypertexts: Identifying Hierarchies and Useful Metrics", ACM Transactions on Information Systems, Vol. 10 No. 2, 1992

[13] Hoorn J.F., "A Model for Information Technologies that Can Be Creative", 4th conference on Creativity and Cognition, ACM Press, 2002

[14] Rosch E., Mervis C.B., "Family resemblances: Studies in the internal structure of categories." Cognitive Psychology (573-603), 1975

[15] Burns E., "Retailers, Clean Up Your Online Stores", ClikzStats http://www.clickz.com/stats/sectors/retailing/ article.php/3567181#table, 2005

[16] Simon H. "Models of Man: Social and Rational", Wiley, 1957

[17] Klein G., "Sources of Power: How People Make Decisions", MIT Press, 1998

[18] Albers M. J., "Cognitive strain as a factor in effective document design", International Conference on Computer documentation, ACM, 1997

Key Issues and Metrics for Evaluating Product Line Architectures [1]

Soo Ho Chang, Hyun Jung La, and Soo Dong Kim
Department of Computer Science
Soongsil University, Seoul, Korea
{shchang, hjla}@otlab.ssu.ac.kr, sdkim@comp.ssu.ac.kr

Abstract

Product Line Engineering (PLE) has been widely accepted as a representative software reuse methodology by using core assets. As a key element of core assets, product line architecture (PLA) is generic to a set of applications in the product line (PL). However, the difference between PLA and single system architecture has not been treated well enough, so evaluating PLA still remains as one of the difficult tasks in PLE. In this paper, we first propose the meta-model of PLA, and identify two intrinsic but overlooked issues in PLA; variability propagation chain and conflicts between architectural elements. And, we present a metric-based method to evaluate PLA from the perspective of the two issues. We believe that the two issues in PLA and the evaluation method would make designing high-quality PLA more feasible and effective.

1. Introduction

Product line engineering (PLE) has been widely accepted as a representative software reuse methodology. In PLE, there are two key phases; one is core asset engineering where reusable assets are identified and developed, and the other is application engineering where a target application is generated by instantiating the predefined core assets. In the core assets, product line architecture (PLA) is an essential element to represent the overall structures of the member applications and it also defines the scope of a product line (PL) [1][2][3]. There are two main characteristics to consider in evaluating PLA; architectural variability and non-functional requirements to a family of applications.

In PLE, variability often exists not only on components but also on its architecture, i.e. PLA. Validating the architectural variability largely remains as an unresolved research issue. Validating non-functional requirements such as performance and security for single systems is known as a highly conceptual and technical task. Validating non-functional requirements for a family of applications in PLE presents even more technical and complicated problems. These characteristics of PLA lead to the two key issues in designing PLA; *variability propagation chain* and *conflicts between architectural elements*, which were largely overlooked in PLE community.

In this paper, we first present a reference model of PLA, which defines key elements of a PLA and provides the basis for presenting further arguments and evaluation methods. Then, we identify and explore these two key issues of designing PLA in details. With the consideration of the two key issues, we then define a metric-based method to evaluate PLA. We believe that the two issues in PLA and the evaluation method would make designing high-quality PLA more feasible and effective.

2. Related Work

In Bosch's work [1], PLA is explicitly structured with components and their relationships. Moreover, it defines an activity called architecture transformation in which quality attributes are applied to functionality-based architecture and architectural variability is treated. In QADA [3], conceptual and concrete architectures are designed with components and relationships from three viewpoints; Structural View, Behavioral View, and Deployment View. This work states that PLA variability should be treated with explicit constructs of variation points, variants, and decision model, however, it does not elaborate how to apply the constructs in designing PLA.

Thiel[4] extended a description framework for single system architectures, P1471[6], with variability elements for PLE. This work introduces PLA-concerned variation point and its specification. Ceron [5] defined a meta-model of PLA based on P1471, and

[1] This work was supported by grant No. (R01-2005-000-11215-0) from the Basic Research Program of the Korea Science & Engineering Foundation.

the meta-model includes a decision model, variation points and variants.

As a related work for evaluating PLA, Leire and Goiuria suggest a classification scheme of product-line requirements; *product line quality attributes, domain-relevant quality attributes,* and *common behavior* [8]. This work surveys and compares existing PLA evaluation methods and metrics from the perspectives of the classification scheme. In HoPLAA[9], an evaluation method, Holistic Product Line Architecture Assessment, is proposed. It extends ATAM[10] which deals with quality attribute tradeoffs, and this method can be applied to both single systems and product line systems. However, the method relies on less systematic techniques such as scenarios rather than quantity-based metric-based approaches.

3. Meta-model of Product Line Architecture

As a prerequisite to define our meta-model of PLA, we propose three dimensions as the underlying perspectives of the meta-model as in Fig. 1. The *Propagation* dimension is to specify the dependencies among architectural elements, and the *Structure* dimension is to specify the static organization of the architecture, and the *Genericiy* dimension is to model the variable features of the architecture.

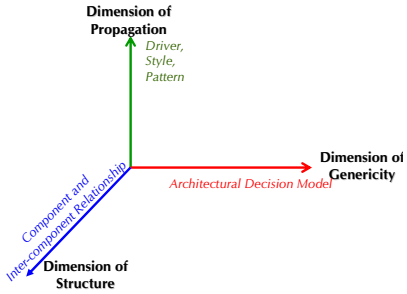


Fig. 1. Dimensions for the PLA Meta-Model

- The dimension of *Propagation* is a viewpoint to clarify factors affecting PLA design. The dimension includes architectural drivers, architectural styles, and architectural patterns. Architectural Driver(called 'driver') is the combination of influential functional, quality, and business requirements which shape the software architecture for the base set of application [4]. *Architectural Style (called 'style')* is a specialization of elements and their relation types[7]. The style is generally derived from architectural drivers.
- The dimension of *Structure* is a viewpoint to enumerate constituents shaping PLA. The elements generally include components and their relationship. Components implement functional and non-functional requirements directly or indirectly. Inter-component relationship is a link which connects components
- The dimension of *Genericity* is a viewpoint to enable architecture to be generic. Components in PLA can be specialized to *common and variable components*, and this dimension is to specify the variable elements. As an essential difference between traditional software architecture and PLA, it includes architectural decision model. Decision model describes the groups of different features for different members of a PL in PLE[13] and the *architectural decision model(ADM)* in this paper specify the differences in PLA. ADM consists of *Variation Point*, *Variant*, *Effect*, and *Attached Task*.

By considering the three dimensions and by surveying representative PLAs in [3][4][5][10], we define a meta model of PLA as in Fig. 2. The PLA consists of *elements* each of which can be a component or the inter-component relationship. Each *element* may have *variable* feature, which is defined by an *architectural decision model*. The PLE itself is designed with styles and/or patterns, which in term satisfy the given architectural driver.

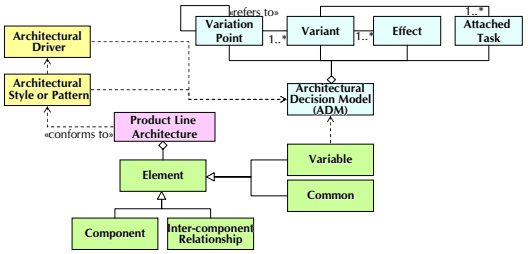


Fig. 2. Meta Model of PLA

4. Issues in Designing PLA

In PLE, architectural variability among the members should be captured and designed into PLA. We find that engineering PLA presents quite new and complex issues and problems which are not typically presented in single system architecture. We now address such issues; *variability propagation chain* and *conflicts between architectural elements*.

4.1. Issue of Variability Propagation Chain

***Variability on Drivers*:** By definitions, a PL consists of several products P_1, P_2, ..., P_n which share a core asset and each P_i has a set of drivers;

$$SetD_i = \{D_{ij} \mid D_{ij} \text{ is the } j^{th} \text{ driver of product } i\};$$

Here, $SetD_i$ is the set of drivers for *product i*.

Typically, products in a PL have a great deal of commonality, but there can be minor differences in their requirements. Hence, it is possible that two products, P_i and P_j, in a PL may have different architectural requirements and different drivers, i.e. $SetD_i$ and $SetD_j$ may not be same. This difference is called variability on architectural drivers. This variability must be properly handled in designing PLA.

While some drivers are common among products, i.e. *mandatory*, others are *alternative* or *optional*. Some quality attributes may be specialized sub-quality attributes. In [10], quality attributes are specialized to more detailed sub-quality attributes. This notion can be applied the driver. Moreover, for one quality attributes, detailed requirement may be slightly different. We define these cases as alternative drivers. We define three classes of drivers as followings;

- *MandatorySetD* is a set of drivers that appear in all products.
- *AlternativeSetD* is a set of drivers which are shared by some members in abstract-level driver, but which are slightly different in detailed-level drivers. We name the abstract-level driver and the detailed-level driver as super-driver and sub-driver respectively. For example, two members may require security as a super-driver for their system. However, one member may want a firewall for the system, but the other may just want to check log. The firewall and logging are sub-drivers of the super-driver.
- *OptionalSetD* is a set of drivers which appear not in all products but in some products.

Therefore, a set of drivers of PL, *PLSetD*, consists of *MandatorySetD*, *AlternativeSetD*, and *OptionalSetD*;

PLSetD = {MandatorySetD, AlternativeSetD, OptionalSetD}

***Variability on Styles*:** Drivers are the basis for selecting styles. For example, *scalability* may be realized by layered or peer-to-peer styles. It can be safely stated that variability on drivers introduces variability on styles.

$SetS_i$ is the set of styles that satisfy a driver set $SetD_i$;

$$SetS_i = \{S_{ij} \mid S_{ij} \text{ is the } j^{th} \text{ style of product } i\};$$

For each $SetS_i$, we can apply similar cases of common and variable drivers into defining mandatory, alternative, and optional styles. For a driver in *MandatorySetD*, one chooses a style from a set of possible styles satisfying the driver. Like the mandatory case, candidate styles for drivers in *AlternativeSetD* and *OptionalSetD* may be one or more. In this situation, the styles from alternative drivers should be distinguished from alternative styles derived from a driver or styles satisfying a driver as shown in Fig. 3.

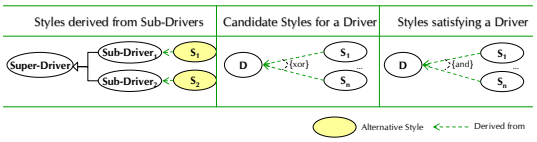


Fig. 3. Style derived from driver vs. alternative styles which may be derived from a driver

The first case in the left column on the figure shows alternative styles S_1 and S_2 which are derived from drivers Sub-Driver$_1$ and Sub-Driver$_2$ respectively. In this case, we consider S_1 and S_2 to be alternative styles since the styles are derived from alternative drivers Sub-Driver$_1$ and Sub-Driver$_2$ which are specialized from the Super-Driver. The second case is that a driver is satisfied with several candidate styles S_1, .., S_n but one style should be finally selected among them. Therefore each candidate style is related with exclusive OR. The third case is that a driver is satisfied by integrating several styles.

We now define three classes of styles as follows;

- *MandatorySetS* is a set of styles which are common among all products.
- *AlternativeSetS* is a set of all alternative styles which are derived from sub-drivers of super-drivers in *AltnerativeSetD.*
- *OptionalSetS* is a set of styles which are applied to some products.

Therefore, a PLA is instantiated from *PLSetS* which consists of *MandatorySetS*, *AlternativeSetS*, and *OptionalSetS*. These three classes of styles should be properly handed in designing PLA.

PLSetS = {MandatorySetS, AlternativeSetS, OptionalSetS}

***Variability on Component and Relationship*:** Functional requirements are realized to components and inter-component relationships *(called 'relationship')* and non-functional requirements also affect configuring PLA with the components and relationships. The components and relationships vary because of variability on functional and non-functional requirements. From this observation, we define three classes of components as follows;

- *MandatorySetCom* is a set of components which are applied to all products.
- *OptionalSetCom* is a set of components which that are applied optionally only to some products.
- *AlternativeSetCom* is a set of components which are alternatively applied.

For the inter-component relationship, *MandatorySetComRel*, *OptionalSetComRel*, and *AlternativeSetComRel* can be similarly applied as components except for only dealing with inter-component relationship.

Therefore, a PLA is configured with *PLSetCom* which consists of *MandatorySetCom*, *OptionalSetCom*, *AlsternativeSetCom, MandatorySetComRel*, *AlternativeSetComRel*, and *OptionalSetComRel*. These classes of components and relationships should be properly handed in designing PLA.

PLSetCom = { MandatorySetCom, AlternativeSetCom, OptionalSetCom, MandatorySetComRel, AlternativeSetComRel, OptionalSetComRel }

Variability Propagation Chain: Since *PLSetD* derives *PLSetS*, variable drivers result in variable styles. Similarly, variable components and relationships are derived from variable drivers and styles. Now, we call it *Variability Propagation Chain*. We should trace from variability on drivers to variability on components. Fig. 4 shows the chain of propagating variability from requirement specification to component and relationship.

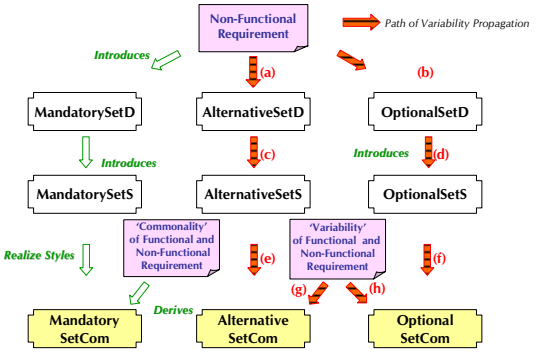


Fig. 4. Architectural Variability and their Propagations

(a) → ***(c)*** → ***(e)***: For edge (a), the variability embedded in non-functional requirement is captured onto *AlternativeSetD*, which in turn introduces alternative styles as in edge (c). And sub-styles of an alternative super-style are realized to various configurations consisting of variable components and relationships as in edge (e). Other propagation paths, i.e. (b) → (d) → (f), (g) and (h), can be similarly applied as above cases.

4.2. Issue of Conflicts among Architectural Elements

As we discussed in the previous section, PLA is a generic architecture that reflects diverse requirements of product members, unlike single system architecture. Therefore, more complicate cases between architectural elements may occur because PLA should be generic.

We now introduce *conflict* as one of the cases. Conflict is a state of disharmony between elements which are incompatible or antithetical [1]. There are several places where a conflict may occur during designing PLA and instantiating PLA. These places are marked with a '✖' mark in Fig. 5.

The first case of conflicts occurs in defining drivers, where the different requirements conflict among PL members. And, if drivers conflict each other, it causes conflict between styles derived from the drivers. The second case occurs in defining concrete components and relationships. In this case, the conflicts are caused by conflicts both in functional requirement and in non-functional requirements. The third case occurs in instantiating architecture for a single system from PLA. Requirements specific to the system may conflict with PLA requirements, and the requirement conflict may cause style and component conflicts between PLA and application-specific architectural requirement.

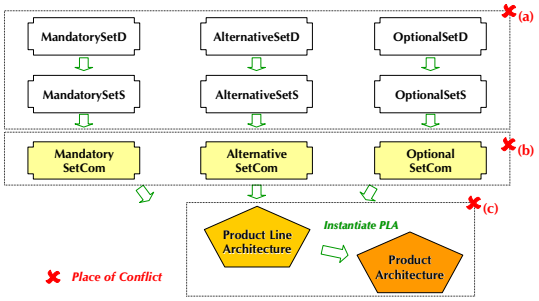


Fig. 5. Places of Occurring Conflicts

We now more precisely specify the three types of conflict as shown in Fig. 6. Case (a) presents a conflict between mandatory elements. This case is same to conflict of elements in single system architecture. Case (b) presents a conflict between mandatory element and variable element whose variation type is alternative or optional. In this case, the conflict can be ignored if the variable element may not be selected in a target system. However, the conflict in designing PLA should be considered in order to satisfying the case where the variable element is selected. Case (C) presents a conflict between variable elements. This conflict should be considered in the case where two variable elements are required by a target system.

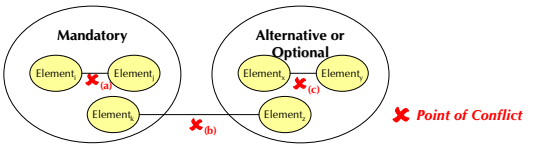


Fig. 6. three types of conflicts between elements

For the three types of conflicts, an element can be specialized to driver, style, component, and relationship. That is, there are three classes of conflicts where an element can be a Driver, Style, Component, and Relationship;

- Conflict between Mandatory(element) and Mandatory(element)
- Conflict between Mandatory(element) and Alternative or Optional(element)
- Conflict between Alternative or Optional(element) and Alternative or Optional(element)

We believe that the issues of *variability propagation chain* and *Conflicts among Architectural Elements* should be properly considered in evaluating PLA.

5. Evaluation Method

In this section, we present a method to evaluate PLA using three metrics; *Architectural Requirement Conformance (ARC), Free of Conflicts (FoC),* and *Tailorability (TAL)*, as in Fig. 7. The three metrics are proposed by considering three dimensions in section 3 and by resolving two issues in section 4. For example, the metric *FoC* measures what degree the PLA is free of conflicts.

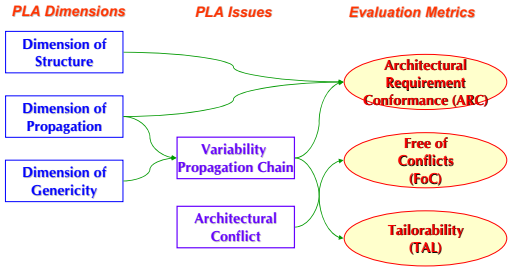


Fig. 7. Evaluation Metrics derived from Dimensions and Issues

5.1. Measuring Architectural Requirement Conformance (ARC)

This metric is to verify how completely the architectural requirement has been satisfied by the resulting PLA. Drivers are derived by analyzing the architectural requirement part of the family requirements. In contrast to requirement specification which is constructed in less formal and textual form, the drivers are typically itemized and numbered for the systematic management purpose. Once we identify the drivers, then all subsequent activities will refer to this organized set of drivers, not the original requirements. Therefore, measuring the Architectural Requirement Conformance (ARC) will be to analyze how many drivers are completely satisfied by the resulting PLA.

ARC is indirectly measured from three sub-measures; Degree of Mandatory Driver Conformance (DMDC), Degree of Optional Driver Conformance (DODC), and Degree of Alternative Driver Conformance (DADC).

$$DMDC = \frac{(n(SetD_{MandatorySetS} \cup SetD_{MandatorySetCom}))}{n(MandatorySetD)}$$

DMDC is to measure how many mandatory drivers are realized in the resulting PLA, and this can be acquired by dividing the total number of mandatory drivers which were based to derive *MandatorySetS* and *MandatorySetCom* by the number of mandatory drivers, i.e. *MandatorySetD*. In the numerator, $SetD_{MandatorySetS}$ and $SetD_{MandatorySetCom}$ are the sets of mandatory drivers which were based to derive mandatory styles and mandatory components respectively.

$$DODC = \frac{(n(SetD_{OptionalSetS} \cup SetD_{OptionalSetCom}))}{n(OptionalSetD)}$$

DODC is defined similarly to DMDC except it only deals with optional elements.

$$DADC_i = \frac{(n(SubSet(AlternativeD_i)_{AlternativeSetS} \cup SubSet(AlternativeD_i)_{AlternativeSetCom}))}{n(SubSet(AlternativeD_i))}$$

where $SubSetD_i$ *is a set of specialized variants for alternative super driver,* $SuperD_i$

$DADC = Avg(DADC_i)$, *for all alternative driver i*

DADC is to measure how many alternative drivers are successfully realized in the resulting PLA. When considering DADC, we should focus on sub-drivers (i.e. $SubSet(AlternativeD_i)$) rather than their super-driver(i.e. $AlternativeD_i$), because PLA can fully satisfy an alternative super-driver when it realizes all the sub-drivers. For each super-driver, we first calculate $DADC_i$ and find the *DADC* by averaging $DADC_i$ values for all its sub-drivers. $DADC_i$ can be acquired by dividing the total number of sub-drivers which were based to driver *AlternativeSetS* and *AlternativeSetCom* by the number of sub-drivers for each super-driver. In the numerator, $SubSet(AlternativeD_i)_{AlternativeSetS}$ and $SubSet(AlternativeD_i)_{AlternativeSetCom}$ are the sets of sub-drivers which were based to derive alternative styles and alternative components for a super-driver respectively.

Now, ARC can be computed as the average of the 3 sub-metrics;

$$ARC = (DMDC + DODC + DADC) / 3$$

Hence, the range of ARC is 0..1. A lower value of ARC indicates that a larger part of architectural requirement is not realized in PLA, while a higher value of ARC indicates that a greater portion of architectural requirement is realized in PLA

5.2. Measuring Free of Conflicts (FoC)

This metric is to verify how many architectural elements are free from architectural conflict. FoC is indirectly measured from two sub-measures; *Free of Driver-Conflict(FoDC)* and *Free of Component-*

Conflict(FoCC).

$$FoDC = 1 - (\frac{the\ number\ of\ driver\ conflicts}{_{n(PLSetD)}C_2})$$

FoDC is to measure how much PLA is free from driver conflicts, and this can be acquired by subtracting the degree of driver conflicts from 1. And, the degree is acquired by dividing the number of driver conflicts by the number of all possible pairs of two drivers. The number of such pairs is denoted as a combination function, $_{n(PLSetD)}C_2$.

$$FoCC = 1 - (\frac{the\ number\ of\ component\ conflicts}{_{n(PLSetSetCom)}C_2})$$

FoCC is defined similarly to FoDC except it only deals with components. Now, FoC can be computed as the combination of the two sub-metrics;

$$FoC = FoDC * W_d + FoCC * W_c,\ where\ W_d + W_c = 1$$

Since conflicts between drivers and conflicts between components have different affects on designing PLA, weights are applied to *FoDC* and *FoCC*; W_d and W_c. Since the impact of driver-conflicts on PLA in general is greater than that of components, the relationship of the weights should be $W_d > W_c$.

Hence, the range of FoC is 0..1. A lower value of FoC indicates that the larger number of architectural conflicts occurs in PLA, while a higher value of FoC indicated that the smaller number of architectural conflicts occurs in PLA and one can acquire more or less stable PLA.

5.3. Measuring Tailorability (TAL)

This metric is to verify how flexibly PLA can be tailored for target applications. TAL is indirectly measured from three sub-measures; *Degree of Driver-Variability(DDV), Degree of Style-Variablity(DSV)* and *Degree of Component-Variability(DCV)*.

$$DDV = \frac{(n(OptionalSetD \cup AlternativeSetD(super\ driver))}{n(PLSetD)}$$

DDV is to measure how many optional and alternative drivers are realized in PLA and this can be acquired by dividing the total number of optional and alternative drivers by the number of all drivers. As shown in the numerator, DDV computation considers the super-drivers rather than sub-drivers.

DSV and DCV are defined similarly to DDV except they only deal with styles and components respectively as followings.

$$DSV = \frac{(n(OptionalSetS \cup AlternativeSetS(super\ style))}{n(PLSetS)}$$

$$DCV = \frac{n(OptionalSetCom \cup AlternativeSetCom \cup OptionalSetComRel \cup AlternativeSetComRel)}{n(PLSetCom)}$$

Now, TAL can be computed as the combination of the 3 sub-metrics;

$$TAL = DDV * W_d + DSV * W_s + DCV * W_c,\ where\ W_d + W_s + W_c = 1$$

Since driver, style, component can differently affect designing PLA, different weights should be applied to *DDV*, *DSV*, and *DCV*. We define the weights, W_d, W_s, and W_c, for driver, style, and component respectively. Generally the impact of drivers on PLA is greater than those of styles and components, because a driver is a key factor on which styles and components are based. The relationship of the three weights should be defined as $W_d > W_s > W_c$.

Hence, the range of TAL is 0..1. A lower value of TAL indicates that the larger part of PLA can not be customized in deriving an application, while a higher value of TLA indicates that a greater portion of PLA can be adapted in deriving an application.

5.4. Combining and Analyzing Three Metrics

Metrics represented in the above sections measure the PLA for its requirement conformance, conflicts, and variability. Now, we combine the metrics into one overall metric, the *Degree of PLA Applicability (DoPLAA)*.

$$DoPLAA = (ARC + FoC + ATAL) / 3$$

This metric is to measure how applicable the PLA is in engineering core assets. In this metric, TAL is refined to *Adoptability of TAL(ATAL)* which presents how adoptable TAL is. ATAL can be computed as following;

$$ATAL = degree\ of\ acceptable\ PLA\ variability\ from\ ROI\ analysis\ /\ TAL$$

The range of ATAL is 0..1 or greater than 1. If the ATAL is greater than 1, it means a degree of identified PLA variability is smaller than that of expected PLA variability and the identified variability is acceptable in core asset engineering. Therefore, the ATAL that is grater than 1 is considered as 1. However, a lower value of ATAL indicates that there is a lot of exceeded variability so it cannot be accepted to core asset engineering. In this case, the variability should be discarded according to its priority and it makes ARC get lower. So an appropriate trade off between TAL and ARC is needed.

The range of DoPLAA is 0…1. In general, a lower value of DoPLAA indicates that it is difficult to adopt the PLA as an element of core assets, while a higher value of DoPLAA indicates that PLA is more applicable in core asset engineering.

When the overall value of DoPLAA is high and one metric has a much lower value than the other two metrics, the following observations may be applicable in interpreting the value of DoPLAA;

- Wen the value of ATAL is much lower than the other two metrics, the variability was not considered and reflected in the PLA, and hence the PLE has a limited flexibility.
- When the value of FoC is much lower than the other two metrics, some elements of the PLA make conflicts with other elements even though architectural requirements are well realized and variability are well modeled. Hence, an architect can have a difficulty in instantiating this kind of PLA.
- When the value of ARC is much lower than the other two metrics, the PLA does not fully realize original architectural requirements. Hence, the PLA needs refinements.

6. Case Study

To illustrate how the evaluation can be applied, we first present architectural requirements and show elements of the corresponding PLA. The example is for *Rental Management* domain of which members can be *Library System* and *Car Rental System*. From the several architectural requirements, we identify and classify commonality and variability.

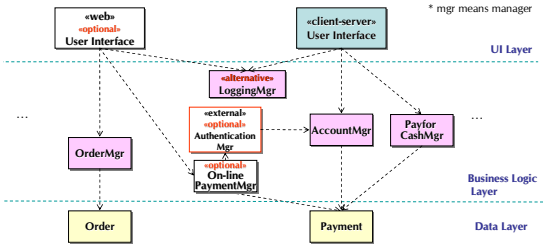


Fig. 8. PLA for Rental Management Domain

Fig. 8 shows a part of the entire PLA from a module view. In the figure, two styles and ten components derived from five drivers including two sub-drivers are represented. From the design, we specify detailed architectural elements and their relationships as in Table 1.

Table 1. Specification of Architectural Elements and Relationships for Rental Management Domain

Driver		Style/ Pattern	Component	
			From Driver	From Func. Req.
Mandatory	Maintainability	Layered	n/a	• Order • Payment • OrderMgr • AccountMgr • PayforCash Mgr • User
Optional	Reliability	Pipe-and-filter	Authentication Mgr On-linePayment Mgr	
	Accessibility	n/a	User Interface for Web	
	Performance	Distribute data	n/a	Interface for C/S
	Data Integrity	Shared Data	n/a	
Alternative	Security	Security with Firewall	n/a	Firewall
		Security with Logging	n/a	LoggingMgr

Based on the table, we now evaluate the PLA by computing ARC, FoC, and TAL. In Table 2, we compute three sub-metrics *DMDC, DODC,* and *DADC* which are used to derive *ARC* value. Similary, *FoDC* and *FoCC* are computed for *FoC*, and *DDV, DSV,* and *DCV* are computed for *TAL*.

Table 2. Metrics and Results

Metrics	Solution
DMDC	$\frac{(n(SetD_{MandatorySetS} \cup SetD_{MandatorySetCom}))}{n(MandatorySetD)} = \frac{1}{1} = 1$
DODC	$\frac{(n(SetD_{OptionalSetS} \cup SetD_{OptionalSetCom}))}{n(OptionalSetD)} = \frac{4}{4} = 1$ * $SetD_{optionalSetS}$ = {Reliability, Performance, Data Integrity} * $SetD_{optionalSetCom}$ = {Reliability, Accessibility} * OptionalSetD = { Reliability, Accessibility, Performance, Data Integrity
DADC	$DADC_1 = \frac{(n(SubSet(AlternativeD_1)_{AlternativeSetS} \cup SubSet(AlternativeD_1)_{AlternativeSetCom}))}{n(SubSet(AlternativeD_1))} = \frac{2}{2} = 1$ $DADC = Avg(DADC_1) = Avg(1) = 1$ * $SubSetD(AlternativeD_1)_{AlternativeSetS}$ = { } * $SubSetD(AlternativeD_1)_{AlternativeSetCom}$ = {Security with Firewall and Logging} * $SubSet(AlternativeD_1)$ = {Security with Firewall and Logging}
ARC	(DMDC + DODC + DADC) / 3 = 3/3 = 1
FoDC	$FoDC = 1 - (\frac{the\ number\ of\ driver\ conflicts}{n(PLSetD)\ C_2}) = 1 - \frac{1}{15} = 0.93$ * Data of car rental system should be distributed in several places because the users want to access the data with time limitation. And, data of library system should be stored in a sharable place in order to maintain data integrity. However, conflict occurs between the distribution and shared data.
FoCC	$FoCC = 1 - (\frac{the\ number\ of\ component\ conflicts}{n(PLSetSetCom)\ C_2}) = 1$
FoC	$FoC = (FoDC + FoCC) / 2 = (1 + 0.94) / 2 = 0.965$
DDV	$DDV = \frac{(n(OptionalSetD \cup AlternativeSetD(super\ driver)))}{n(PLSetD)} = \frac{5}{6} = 0.83$
DSV	$DSV = \frac{(n(OptionalSetS \cup AlternativeSetS(super\ style)))}{n(PLSetS)} = \frac{3}{4} = 0.75$

DCV	$DCV = \frac{(n(OptionalSetCom \cup AlternativeSetCom(super\ component))}{n(PLSetCom)} = \frac{4}{11} = 0.36$ * The firewall component would be represented in deployment view.
TAL	$TAL = DDV * W_d + DSV * W_s + DCV * W_c = 0.83$ * 0.5 + 0.75 * 0.3 + 0.36 * 0.2 = 0.415 + 0.225 + 0.072 = 0.712 * We used values of 0.5, 0.3, and 0.2 for the weights.
ATAL	*ATAL= degree of acceptable PLA variability from ROI analysis / TAL* = 0.4 / 0.712 = 0.56 * We define 0.4 as the degree of acceptable PLA variability from ROI analysis.
DoPLAA	*DoPLAA = (ARC + FoC + ATAL)* / 3 = (1 + 0.965 + 0.56) / 3 = 2.525/3 = 0.84

From the result, we can realize that the design PLA is acceptable with appropriate reliability which is over the half. But, in the aspect of variability, a lot of variability is designed so there is possibility to discard designed variability.

7. Concluding Remarks

PLA is a key element of core assets in PLE. Since it is generic to a family of applications, the evaluation method of PLA should be defined and differs from single software architecture. If the difference between PLA and single system architecture is not treated well enough, so evaluating PLA still remains as one of the difficult tasks in PLE.

In this paper, we first proposed the meta-model of PLA, and identified two intrinsic but overlooked issues in PLA; *variability propagation chain* and *conflicts between architectural elements*. And, we presented a metric-based method to evaluate PLA from the perspective of the two issues. We defined three essential metrics in the method; *Architectural Requirement Conformance (ARC), Free of Conflicts (FoC),* and *Tailorability (TAL)*, and we elaborated how to apply them in practice with instructions and examples. These three metrics can be easily combined with relative weights to yield the overall quality of PLA.

Following our research of exploring the theoretic foundation of design PLA, our future research is to define practical process and instructions to apply the foundation in practice. We believe that the two issues in PLA and the evaluation method would make designing high-quality PLA more feasible and effective.

References

[1] Bosch, J. *Design and Use of Software Architectures*, Addison-Wesley, 2000.

[2] Anastasopoulos, M., Bayer, J., Flege, O., and Gacek, C., *A Process for Product Line Architecture Creation and Evaluation PuLSE-DSSA-version 2.0, Technical Report, No. 038.00/E*, IESE, June 2000.

[3] Matinlassi, M., Niemela, E., and Dobrica, L., "Quality-driven architecture design and quality analysis method: A revolutionary initiation approach to a product line architecture," *VTT Technical Research Center of Finland, proceedings of ESPOO2002*, 2002.

[4] Thiel, S., and Hein, A., "Systematic Integration of Variability into Product Line Architecture Design," *proceeding of SPLC2, LNCS 2379*, Springer, 2002.

[5] Ceron, R., et.al., "Architectural Modeling in Product Family Context," *proceeding of EWAS, LNCS 3047*, Springer-Verlag Berlin Heidelberg, 2004.

[6] *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems (IEEE Standard P1471)*; IEEE Architecture Working Group (AWG); 2000.

[7] Clements,P.,et al., *Documenting Software Architectures Views and Beyond*, Addison-Wesley, 2003.

[8] Etxeberria, L. and Sagardui, G, "Product-Line Architecture: New Issues for Evaluation," *proceedings of SPLC 2005, LNCS 3714*, Springer, 2005.

[9] Olumofin, F. and Misic, V., "Extending the ATAM Architecture Evaluation to Product Line Architecture," *Proceedings of Fifth Working IEEE/IFIP Conference on Software Architecture (WICSA 5), 2005*.

[10] Clements, P., Kazman, R., and Klein, M., *Evaluating Software Architectures*, Addison Wesley, 2002.

[11] Thiel, S., "On the Definition of a Framework for an Architecting Process," *proceeding of PFE-4, LNCS 2290*, Springer-Verlag Berlin Heidelberg, 2002.

[12] Bass, L., Clements, P., Kazman, R., *Software Architecture in Practice*, Addison-Wesley, 2003.

[13] Atkinson, C., et al., *Component-based Product Line Engineering with UML*, Addison Wesley, 2001.

[14] Jacobson, I., Griss, M, and Jonsson, P., *Software Reuse*, Addison Wesley, 1997.

Multiple Imputation of Software Measurement Data: A Case Study

Taghi M. Khoshgoftaar*
Jason Van Hulse

Department of Computer Science and Engineering
Florida Atlantic University, Boca Raton, Florida

Abstract

The treatment of missing data is an important task in the analysis of software measurement data. Multiple imputation (MI) has been demonstrated to be a robust and theoretically sound statistical technique developed for handling missing data. We present a case study using MI to impute missing data in a real-world software measurement dataset called CCCS. The main purpose of this study is to explore multiple imputation in the context of missing software metrics data and to examine the impact of different missingness levels and mechanisms on the imputation accuracy of MI. We conclude that both the level of missing data and missingness mechanism are important factors when utilizing MI for imputation of missing software metrics data.

Keywords: missing data, multiple imputation, software metrics, missingness mechanism, software quality.

1. Introduction

Software defect estimation models utilize data mining and knowledge discovery algorithms to generalize patterns inherent in software measurement data. The instances or observations in a software measurement dataset typically represent software program modules. Software measurement information regarding the program modules are used as independent variables, while the dependent variable indicates the number of observed faults in the program module. A data mining model can be used to encapsulate the relationship between the software metrics and the number of faults. Such a model can be used to predict the number of faults for program modules after the software metrics have been measured for recently developed program modules (i.e., test data). Limited project resources can then be allocated in an ef cient manner, with emphasis placed on inspecting program modules which are predicted to contain the most faults.

One of the dif culties encountered in real-world software defect estimation applications is missing data. Data is de ned to be missing if it is unavailable or unde ned, and therefore a value is not provided. Many data mining techniques such as linear regression cannot directly handle missing data, leading to the proposal of numerous methodologies for incomplete data. The simplest and most direct procedure is to delete any instance with missing data, which can obviously lead to signi cant information loss, especially with small datasets. Alternatively, techniques have been proposed to ' ll-in' or impute missing software metrics data, for example mean, regression, k-Nearest Neighbor (k-NN), hot deck and cold deck imputations. Many studies have examined imputation techniques for software cost estimation [1, 8], while relatively little published work is available in the domain of software defect estimation [5].

In this paper, we present a case study focusing specifically on Bayesian multiple imputatation (MI), rst introduced in [9], in the context of software quality estimation. Further, we examine the impact of the missingness mechanism and the amount of missing data on the ability of MI to produce accurate imputations. To our knowledge, this is the rst study to examine the impact of missingness mechanisms and the amount of missing data with respect to multiple imputation for software defect estimation. Generally speaking, MI has received relatively little attention in the software engineering domain [1].

Multiple imputation [11] is a robust imputation technique which replaces each missing value with a list of $m > 1$ imputed values. A dataset $\mathfrak{D}$ with missing values can be divided into two datasets, $\mathfrak{D}_{obs}$ and $\mathfrak{D}_{mis}$, where $\mathfrak{D}_{obs}$ and $\mathfrak{D}_{mis}$ are the observed and missing portions of $\mathfrak{D}$, respectively. MI creates m imputed datasets $\mathfrak{D}_{mis}^{(1)}, \ldots, \mathfrak{D}_{mis}^{(m)}$ which combined with $\mathfrak{D}_{obs}$ can be analyzed using standard complete data methods.

Data augmentation (DA) can be used to generate imputed datasets $\mathfrak{D}_{mis}^{(k)}$ by repeatedly sampling from the distribution $P(\mathfrak{D}_{mis} \mid \mathfrak{D}_{obs})$. DA proceeds iteratively through

*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.

two steps known respectively as the I-(or Imputation) Step and the P-(or Posterior) Step [12] as follows:

(I) Suppose that the current estimate for parameters θ are given by $\theta^{(t)}$. Draw an estimate of the missing data $\mathfrak{D}_{mis}^{(t+1)}$ from the distribution $P(\mathfrak{D}_{mis} \mid \mathfrak{D}_{obs}, \theta^{(t)})$.

(P) Using the estimate for the missing data, draw $\theta^{(t+1)}$ from $P(\theta \mid \mathfrak{D}_{obs}, \mathfrak{D}_{mis}^{(t+1)})$.

With m large enough, $\mathfrak{D}_{mis}^{(m)}$ is a random draw from the distribution $P(\mathfrak{D}_{mis} \mid \mathfrak{D}_{obs})$. Parameters $\theta^{(0)}$ are often initialized using the maximum likelihood estimates calculated by the Expectation Maximization algorithm.

Schafer [11] de nes *Bayesian proper* multiple imputations as those which are independent draws from the posterior predictive distribution of the missing data $P(\mathfrak{D}_{mis} \mid \mathfrak{D}_{obs})$. The sequence of datasets generated by data augmentation must undergo further processing in order to be considered proper, and two common approaches are to use sequential or parallel chains. Sequential chains subsample from the sequence generated by data augmentation. Parallel chains require the execution of multiple runs of MI, where the dataset at the nal iteration is the only one that is used.

Our experiments consider a real-world software measurement dataset called CCCS. We focus on the injection of missing data into the dependent variable $nfaults$, which records the number of faults attributed to the program module. The performance of MI is evaluated under the MCAR, MAR and NI missingness mechanisms and with 5%, 10%, 15% and 20% missing data. An analysis of variance (ANOVA) is included to evaluate the impact of these factors on the imputation results produced by MI.

The remainder of this paper is organized as follows: Section 2 provides a description of the dataset used in the case study. The empirical results and ANOVA analysis are presented in Section 3. Conclusions and directions for future work are presented in Section 4.

2. CCCS Dataset

The CCCS dataset is a large military command, control and communications system written in Ada [4]. CCCS contains 282 instances (program modules), where each instance is an Ada package consisting of one or more procedures. CCCS contains 8 independent variables or attributes along with an additional attribute $nfaults$ indicating the number of faults attributed to the module during the system integration and test phases and during the rst year of deployment.

Our recent studies [5] have demonstrated the adverse impact of poor quality software metrics data on the imputation process. Software metrics data is of low quality if it contains errors or noise. Measuring the imputation results derived from low quality data may present a misleading picture. Therefore, before performing any imputation, the dependent variable in the CCCS dataset was cleansed using the input of an expert with many years of experience in the software engineering domain as explained in the Appendix (Section 5). Our data cleansing concentrated speci cally on the dependent variable, because missing values were only injected into $nfaults$. The software metrics data used in this study, therefore, is relatively clean with respect to the dependent variable, although noise may exist in the independent variables.

3. Empirical Investigation

Our case study uses multiple imputations generated from a normal distribution, which has been shown to perform extremely well even if the data is highly non-normal [11]. We also use sequential chains with 1500 iterations, where 15 imputed values are stored for each chain. Further, we execute MI 15 times, so each missing value has $15 \times 15 = 225$ imputed values. To generate a single imputed value for each instance with missing data, we nd the average imputed value over all 225 iterations.

3.1 Missingness Mechanisms

In general, three missingness mechanisms have been identi ed in related literature [7]: MCAR (missing completely at random), MAR (missing at random), and NI (non-ignorable). If the input dataset $\mathfrak{D}$ contains n instances and m attributes, let R be an $n \times m$ matrix such that the entry $r_{ij} = 1$ if the j^{th} attribute of instance i is missing, and 0 otherwise. In other words, R identi es where the missing values in $\mathfrak{D}$ are located. Then MCAR missingness occurs if $\mathrm{P}(R \mid \mathfrak{D}_{obs}, \mathfrak{D}_{mis}) = \mathrm{P}(R)$. Missingness is MAR if $\mathrm{P}(R \mid \mathfrak{D}_{obs}, \mathfrak{D}_{mis}) = \mathrm{P}(R \mid \mathfrak{D}_{obs})$ (i.e., the occurrence of missing data depends only on the observed values and not on the missing values). If the missingness pattern is related to the missing values themselves, then the missingness is called non-ignorable (NI).

This study considers MCAR, MAR and NI missingness mechanisms with missing data only in the dependent variable $nfaults$. Missing data was not injected into any of the independent variables in the dataset. MCAR was implemented by simple random selection. Given a missingness level of $\delta\%$, $\delta\% \times 282$ instances were randomly selected and $nfaults$ was set to missing for those instances.

MAR missingness was implemented by making the random selection of instances dependent on the value of another fully observed attribute in the dataset, the total lines of code (or $tloc$). Given a missingness level of $\delta\%$, instances were randomly selected only if $tloc \geq 900$. Note that 70 instances in CCCS (or 24.82% of the dataset) have $tloc \geq 900$. To generate $\delta\%$ missingness in $nfaults$ under the MAR mechanism, $\delta\% \times 282$ instances were randomly selected from the set of instances with $tloc \geq 900$. In some applications, it is sensible that larger program modules have a higher likelihood to have unreported and hence missing fault data. One possibility is that numerous people

Table 1. Number of Instances with Missing Data over all 5 Selections

	mmech			
mper	MCAR	MAR	NI	Total
5%	70	70	70	210
10%	140	140	140	420
15%	210	210	210	630
20%	280	280	280	840
Total	700	700	700	2100

may be working on the same large program module, making the tracking fault data for the module more dif cult.

NI missingness was generated by randomly selecting instances with $nfaults \geq 2$ since missing values are only injected into the dependent variable $nfaults$. 83 instances from the CCCS dataset have $nfaults \geq 2$. NI missingness is a potentially common occurrence in the imputation of the number of faults in software quality data, since a large number of faults in a program module may be indicative of poor performance of the individual or group involved in the software development project. There may be a tendency for a variety of reasons to either misrepresent the number of faults or not to report fault data at all.

3.2 Evaluation Metric

Let $\mathcal{M}$ denote the set of instances with missing values for the dependent variable $nfaults$, which in this section we denote by y. For each instance $i \in \mathcal{M}$, denote the value of the dependent variable for instance i by y_i and the imputed value by $\hat{y}_i$. For all instances $i \in \mathcal{M}$, the absolute (imputation) error err_i of instance i is calculated as:

$$err_i = \mid y_i - \hat{y}_i \mid .$$

3.3 Notation

Our case study uses the CCCS dataset, which a priori has no missing data. In order to test the imputation ability of MI, missing values were injected at various levels $mper$ and with different missingness mechanisms $mmech$ into the dependent variable $nfaults$. We considered $mper = 5\%, 10\%, 15\%,$ and 20% and $mmech =$MCAR, MAR and NI. As CCCS contained 282 instances, 5% missingness is 14 instances, 10% missingness is 28 instances, 15% is 42 instances and 20% is 56 instances. Further, 5 different random selections were made for each missingness level and mechanism. In other words, for a given missingness mechanism and missingness level, the process of randomly selecting instances was done ve times. Note that we only include up to 20% missingness due to constraints on the random selection process for MAR and NI missingness. For MAR missingness, for example, no more than 25% of the instances could be randomly selected since we require $tloc \geq 900$, which includes only 24.82% of the dataset.

Table 1 displays the number of observations in dataset with missing data for each combination of the factors $mmech$ and $mper$ over all 5 random selections. For example, a missingness level of 5% over 5 sets generates a total of 210 instances with missing data, 70 with mechanism MCAR, 70 with mechanism MAR and 70 with mechanism NI.

Our experimental setup can be summarized as follows:

1. For a given missingness mechanism $mmech$ and missingness percent $mper$, generate ve independent datasets with missing values from CCCS.
2. Use MI to impute the instances with missing data in each of the ve datasets separately.
3. Round the imputed value for $nfaults$ to the nearest non-negative integer, and record the absolute errors.
4. Repeat 1 – 3 for each $mmech$ and $mper$.

3.4 ANOVA model

A two factor ANOVA model [2] can be represented as:

$$err_{jkl} = \mu + \alpha_j + \beta_k + (\alpha\beta)_{jk} + \epsilon_{jkl} \qquad (1)$$

where err_{jkl} is the absolute error of the l^{th} instance for the j^{th} level of α and the k^{th} level of β; α_j is the mean err of level j for attribute α; β_k is the mean err of level k for attribute β; $(\alpha\beta)_{jk}$ is the mean err of the instances with $(\alpha = j, \beta = k)$; and ϵ_{jkl} is the random error. Attributes α and β are called *main* effects and $(\alpha\beta)_{jk}$ is called a *crossed* effect.

In this study, the main effects are the missing percentage ($mper$), which has four levels, and the missingness mechanism ($mmech$), which has three levels. We also evaluate the crossed effect $mper \times mmech$. Note that based on the observation counts presented in Table 1, we have an unbalanced two factor experimental design.

In Section 3.2, instances with missing values are indexed by $i = 1, \ldots, 2100$. The ANOVA model presented in Equation 1, however, uses indices $\{j, k, l\}$. The later notation is preferred for clarity of presentation, and a simple reparametrization $i \mapsto \{j, k, l\}$ can be performed. If $\alpha = mper$ and $\beta = mmech$, then $j = 1$ corresponds to $mper = 5\%$, $j = 2$ corresponds to $mper = 10\%$, and so on. $k = 1$ corresponds to $mmech =$ MCAR, $k = 2$ corresponds to $mmech =$ MAR, and $k = 3$ corresponds to $mmech =$ NI. Then $i = 1 \mapsto \{j, k, l\} = \{1, 1, 1\}, \ldots, i = 70 \mapsto \{j, k, l\} = \{1, 1, 70\}, i = 71 \mapsto \{j, k, l\} = \{2, 1, 1\} \ldots, i = 2100 \mapsto \{j, k, l\} = \{4, 3, 280\}$.

The ANOVA model can be used to test the hypothesis that the mean imputation error for each combination of the main factors $mper$ and $mmech$ are equal against the alternative hypothesis that at least one mean is different. If the alternative hypothesis (i.e., that at least one mean is different) is accepted, numerous procedures can be used to determine which of the means are signi cantly different from the others. This involves the comparison of two means, with the null hypothesis that the means are equal. A type I error occurs when the null hypothesis is incorrectly rejected.

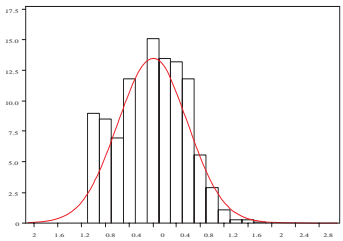


Figure 1. ANOVA Residual Plot using $\sqrt{err}$

Table 2. Average and Standard Deviation of $\sqrt{err}$

	$mmech$					
	Mean $\sqrt{err}$			Standard Deviation $\sqrt{err}$		
$mper$	MCAR	MAR	NI	MCAR	MAR	NI
5%	0.6139	0.8599	0.8027	0.5851	0.4852	0.6243
10%	0.4566	0.8445	0.8726	0.5741	0.5636	0.6139
15%	0.5119	0.9199	0.8629	0.5827	0.5800	0.6095
20%	0.4504	0.9771	0.9446	0.5558	0.6439	0.6139

When analyzing the differences between multiple means, two types of error rates are generally used. The *comparison-wise* error rate is the type I error for each comparison. The *experiment-wise* error rate, on the other hand, controls the type I error rate for the entire experiment. Two very popular multiple comparison tests [2] are Fisher's Least Signi cant Difference (LSD) test, which controls the type I comparison-wise error rate, and Tukey's Studentized Range (HSD) test, which controls the type I experiment-wise error rate.

3.5 Statistical Analysis

This section presents the results of an ANOVA analysis of the absolute errors produced by multiple imputation. One assumption when performing an ANOVA analysis is that the residuals ϵ_{jkl} are normally distributed. Therefore, we transformed the absolute error by taking the square root. All ANOVA results are presented using $\sqrt{err}$ instead of err. A histogram of the residuals using $\sqrt{err}$ is presented in Figure 1, with an overlay of a normal distribution. The data appears to be reasonably approximated by a normal distribution, and there are no signi cant outliers.

Table 2 presents the mean and standard deviations of the $\sqrt{err}$ for different levels of $mper$ and $mmech$. First note that MCAR generally has a smaller absolute error than MAR and NI, and MAR and NI have average absolute errors that are very similar. Also note that the average absolute errors generally increase as $mper$ increases for both MAR and NI, while for MCAR missingness, $mper$ has little effect on $\sqrt{err}$. The standard deviation of the absolute error stays relatively stable, except for MAR, which increases from 0.4852 with 5% missingness to 0.6439 with 20% missingness. Equality of the variances is another assumption of an ANOVA analysis, and using a Levine's test for homogeneity the variances are not signi cantly different at the 0.5% level. The ANOVA model has been shown to be robust to moderate deviations from this assumption [2].

Table 3. Analysis of Effects

Source	df	Sum of Squares	Mean Square	F-statistic	p-value
$mmech$	2	51.264	25.632	72.81	< 0.0001
$mper$	3	1.244	0.4148	1.18	0.3165
$mmech \times mper$	6	4.042	0.6737	1.91	0.0751

The ANOVA model has an F-statistic of 22.72 and a p-value of less than 0.0001, implying that there is strong statistical evidence that the means of $\sqrt{err}$ for the different combinations of $mper$ and $mmech$ are not equal. Table 3 displays the individual effects used in the ANOVA model. More speci cally , the SAS GLM procedure was used [3, 10], and Table 3 presents the sum of squares (SS) for the main effects $mper$ and $mmech$ and the crossed effect $mmech \times mper$. The SS measures the contribution of each term to the model including all other possible terms [3, 10]. $mmech$ has a very large F-statistic with a value of 72.81, which is highly signi cant. $mper$ is not a signi cant factor with a p-value of 0.3165. This implies that with the factors $mmech$ and $mmech \times mper$ in the model, the main effect $mper$ is not signi cant. The crossed term $mmech \times mper$ is signi cant at the 10% level but not at the 5% level.

Table 4 presents the details of the main effect $mmech$ in the ANOVA model. MCAR has the lowest mean for $\sqrt{err}$ with a value of 0.48646. The mean value for $\sqrt{err}$ for the NI missingness mechanism is 0.89152 and for MAR is 0.92170. Each mechanism type had exactly 700 observations. The nal two columns present the groupings based on the multiple comparison test. Means with the same letter are not signi cantly different based on the statistical test used, either Tukey's HSD or Fisher's LSD. Table 4 uses a signi cance level of $\alpha = 5\%$. MCAR is deemed signi cantly different than MAR and NI based on both the LSD and HSD tests, while there is no signi cant difference between MAR and NI. It was expected that the MCAR mechanism would have the lowest mean, as completely random injection of missing data can be handled ef ciently by multiple imputation. MAR and NI result in similar performance, with NI being slightly better. This is not surprising, because the correlation between independent variables and $nfaults$ in CCCS is high in many cases, and the selection of instances based on MAR and NI is similar for this dataset. In other words, instances with $tloc \geq 900$ also tend to have a large number of faults, so the instances randomly selected under MAR also had a high probability of being randomly selected under NI (which explicitly only considered instances with $nfaults \geq 2$).

The mean $\sqrt{err}$ does not exhibit any signi cant differences for different levels of $mper$ (Table 3). By examining

Table 4. Main Effect: $mmech$

$mmech$	Mean $\sqrt{err}$	Number of Observations	Grouping HSD	Grouping LSD
MAR	0.92170	700	A	A
NI	0.89152	700	A	A
MCAR	0.48646	700	B	B

Table 5. Main Effect: $mper$ with Mechanisms NI and MAR Only

$mper$	Mean $\sqrt{err}$	# obs	$\alpha = 5\%$ HSD	$\alpha = 5\%$ LSD	$\alpha = 10\%$ HSD	$\alpha = 10\%$ LSD
20%	0.96087	560	A	A	A	A
15%	0.89139	420	A	AB	AB	AB
10%	0.85858	280	A	B	AB	B
5%	0.83131	140	A	B	B	B

Table 2, however, it does appear that when the missingness mechanism is MAR or NI, the percentage of missing data does effect the mean error. For MCAR, however, $mper$ appears to have no effect on the mean error. These observations are consistent, since MCAR is completely random missingness, there is enough non-missing data for multiple imputation to ' ll-in the gaps'. According to MAR and NI, which have a very high percentage of missing information restricted to the large values of $nfaults$, either explicitly in the case of NI or implicitly due to the correlation structure of CCCS in the case of MAR, as more missingness is injected into the dataset, a signi cant amount of information regarding the tail of the distribution of $nfaults$ (i.e., large values) is removed. MI, therefore, has very few instances with large values of $nfaults$ that are non-missing as $mper$ increases. For MAR and NI, we therefore anticipate a deterioration of the imputation results for larger values of $mper$.

To test the hypothesis that $mper$ is a signi cant factor when considering the NI and MAR missingness mechanisms only, we removed the MCAR observations and re-executed the ANOVA analysis. In this case, $mper$ is a signi cant factor with a p-value of 0.0336. Table 5 presents the analysis of $mper$ considering only the NI and MAR missingness mechanisms. 20% missingness results in the largest mean error, followed by 15%, 10%, and 5%. Tukey's and Fisher's groupings are also presented with $\alpha = 5\%$ and 10%. These results demonstrate that the amount of missingness in the data does effect the imputation results in CCCS when the missingness mechanism is either MAR or NI.

4. Conclusions and Future Work

The handling of missing data is an important issue in the eld of software fault estimation. Bayesian MI has been successfully applied in many different application domains but has not received much attention in software engineering. We present the results of many experiments using MI with a real-world software measurement dataset called CCCS. More speci cally , we arti cially introduced missing data into the dependent variable of CCCS using three different missingness mechanisms $mmech$ and at four different missingness levels $mper$. To our knowledge this is the rst study to examine in detail the effect of missingness mechanisms and the level of missingness on MI in the software quality domain.

Using an ANOVA model, we analyzed the signi cance of the main effects $mmech$ and $mper$ on the imputation results, and found that the missingness mechanism was highly signi cant. MCAR clearly demonstrated the best results, while NI and MAR were not signi cantly different. When considering the full ANOVA model, the percentage of missing data was not a signi cant factor.

Upon further analysis, it was determined that removing the absolute errors derived from MCAR resulted in $mper$ being a signi cant factor. In other words, the percentage of missing data at the levels considered in this study (5%, 10%, 15% and 20%) was not signi cant with MCAR, but was only signi cant when the data was missing according to the MAR or NI mechanisms. Based on the results presented in this study, we feel MI is a very useful technique for imputation in software measurement datasets.

Future work will include additional experiments with MI, along with a comparative study with other imputation techniques. We will also examine higher levels of missingness, as well as multi-attribute missingness.

References

[1] B. Twala and M. Cartwright. Ensemble Imputation Methods for Missing Software Engineering Data. In *Proceedings of 11th IEEE Intl. Software Metrics Symposium*, page 30, 2005.

[2] M. L. Berenson, D. M. Levine, and M. Goldstein. *Intermediate Statistical Methods and Applications: A Computer Package Approach*. Prentice-Hall, Inc., 1983.

[3] G. Der and B. S. Everitt. *A Handbook of Statistical Analyses using SAS*. Chapman & Hall/CRC, second edition, 2002.

[4] T. M. Khoshgoftaar and E. B. Allen. Classifcation of fault-prone software modules: Prior probabilities, costs and model evaluation. *Empirical Software Engineering*, 3:275 – 298, 1998.

[5] T. M. Khoshgoftaar, A. Folleco, J. Van Hulse, and L. Bullard. Multiple imputation of missing values in software measurement data. *Technical Report*, 2006.

[6] T. M. Khoshgoftaar and J. Van Hulse. Identifying noise in an attribute of interest. In *Proceedings of the IEEE International Conference on Machine Learning and Applications*, pages 55–60, Los Angeles, CA, December 2005.

[7] R. J. A. Little and D. B. Rubin. *Statistical Analysis with Missing Data*. John Wiley and Sons, Hoboken, NJ, 2nd edition, 2002.

[8] P. Jonsson and C. Wohlin. An evaluation of k-nearest neighbour imputation using likert data. *10th IEEE Intl. Symposium on Software Metrics (METRICS'04)*, pages 108–118, 2004.

[9] D. B. Rubin. *Multiple Imputation for Nonresponse in Surveys*. J. Wiley and Sons, 1987.

[10] SAS Institute. *SAS/STAT User's Guide*. SAS Institute Inc., 2004.

[11] J. L. Schafer. *Analysis of Incomplete Multivariate Data*. Chapman and Hall/CRC, 2000.

[12] M. A. Tanner and W. H. Wong. The calculation of posterior distributions by data augmentation. *Journal of the American Statistical Society*, 82:528–550, 1987.

[13] J. Van Hulse, T. M. Khoshgoftaar, and H. Huang. The pairwise attribute noise detection algorithm. *Knowledge and Information Systems Journal, Special Issue on Mining Low Quality Data*, 2006. In Press.

5. Appendix

The appendix describes our hybrid procedure for cleansing a quantitative dependent variable and our application of that methodology to the CCCS dataset, which is known to contain noise in the dependent variable. Our hybrid technique relies on two components described individually in 5.1 and 5.2. In 5.3, the hybrid procedure itself is explained along with its application to the CCCS dataset. The objective of this procedure is to create a relatively cleaner dataset from CCCS, which will be used to conduct further experiments.

The rst component described in 5.1 uses our proposed technique for identifying noise in a user-speci ed *attribute of interest* (AOI) [6], which can be any attribute in the dataset. A relative ranking of the instances is provided based on the level of noise contained in the AOI. Alternative values are then calculated for those instances determined to be noisy relative to the AOI.

Independent of the AOI technique, we have proposed another novel procedure called the multiple imputation (MI) quantitative noise detector (5.2). This procedure calculates an alternative value for the dependent variable of each observation in the dataset. Noisy instances are identi ed when the average imputed value for the dependent variable differs signi cantly from the actual value.

Using the alternative values suggested by these techniques, the outcome variable can be cleansed based on *expert* input. The combination of these procedures into a hybrid technique is presented in 5.3. Expert input was a critical component when cleansing the CCCS dataset. The input of a software engineering domain expert ensured that the alternative values suggested by our procedure were sensible given the values for the independent variables. Cleansing occurred only if the expert agreed that the instance was noisy and if the estimated values were sensible. Therefore in this case study, a conservative, expert-guided approach to noise cleansing was used.

5.1 Determine Noisy Instances Relative to an Attribute of Interest

A brief overview of our procedure for ranking noisy instances relative to an attribute of interest or AOI is presented here. This technique utilizes a procedure called PANDA (Figure 2), which provides a ranking of instances from most to least noisy based on the Noise Factor S_i. For each observation in the dataset, PANDA examines each pair of attributes and computes the deviation of the second attribute from its mean value given the partitioned value of the rst attribute. For a given instance, if these deviations occur often and severely enough when compared to the remainder of the dataset, that instance will appear more noisy. Additional details on PANDA can be found in [13].

PANDA Algorithm

input: Dataset $X = [x_{ij}]_{n \times m}$ with n observations and m attributes, where x_{ij} is the value of the j^{th} attribute for the i^{th} observation. x_{*j} denotes the j^{th} attribute.

output: Noise Factor S_i for all the observations in Dataset X

1. Partition each attribute x_{*j} into disjoint bins $\{1, \ldots, L\}$, where $L = \#$ of partitions. Denote the partitioned attribute by $\hat{x}_{*j}$ and the value of the j^{th} partitioned attribute for instance i as $\hat{x}_{ij}$. Note in particular that $\hat{x}_{ij} \in \{1, \ldots, L\}\ \forall\ i$.
2. Calculate $Mean(x_{*k} \mid \hat{x}_{*j} = l)$ and $Std(x_{*k} \mid \hat{x}_{*j} = l)$ for each pair of atributes $(\hat{x}_{*j}, x_{*k})$, $1 \leq j \neq k \leq m$ and for $l = 1, \ldots, L$.
3. For each instance $i \in X$, calculate
$$S_i = \sum_{k=1}^{m} \sum_{\substack{j=1 \\ j \neq k}}^{m} |x_{ik} - Mean(x_{*k} \mid \hat{x}_{*j} = \hat{x}_{ij})| \ / \ Std(x_{*k} \mid \hat{x}_{*j} = \hat{x}_{ij}).$$
4. Sort the instances based on S_i from the largest to the smallest value.

Figure 2. PANDA

Using PANDA, our procedure for detecting noise relative to an attribute of interest is presented here, with further details available in [6]. Suppose there are m attributes in the dataset and AOI is selected by a user of the technique to be one of these attributes. The noise ranking $rank$ of the instances is calculated using PANDA with all m attributes. The rank of instance x_i is denoted $rank(x_i)$, where values for $rank(x_i)$ closer to one represent the overall most noisy instances in the dataset. The AOI is then removed from the dataset and PANDA calculates the instance noise ranking using the remaining $m - 1$ attributes. Denote the obtained instance ranking as $rank_{AOI}$. The difference in ranking before and after the AOI is removed is calculated for all instances ($\Delta(x_i) = rank_{AOI}(x_i) - rank(x_i)$). Instances with the largest (positive) difference in ranking are considered the most noisy relative to the AOI. Therefore, our procedure provides a ranking of instances relative to the amount of noise contained in the AOI.

5.2 MI Quantitative Noise Detector

Our technique for the detection of noise in a quantitative outcome using multiple imputation is presented in Figure 3. The input dataset $\mathfrak{D}$ contains a quantitative dependent variable Y. Given a user-de ned parameter L, $\mathfrak{D}$ is randomly partitioned into disjoint, equal-sized datasets $\mathfrak{P}_1, \ldots, \mathfrak{P}_L$. L additional datasets are created where $\mathfrak{M}_l = \mathfrak{P}_l$ with Y (the dependent variable) set to missing for all instances. Finally, the dataset $\mathfrak{D}_l$ is created on Line 4.

$Impute(\cdot)$, on Line 6, imputes Y for those instances in $\mathfrak{D}_l$ with a missing value for Y. The rst parameter to $Impute()$ speci es the input dataset. Next, the type of chain (either single or multiple) is speci ed. The third parameter Γ is the total number of values for Y that are retained from a single execution of $Impute()$. lag is the number of cycles in a chain between imputed values, and $burn$ is the

Multiple Imputation Quantitative Noise Detector

1. For $\lambda = 1, \ldots, \Lambda$ do:
2. Randomly divide $\mathfrak{D}$ into disjoint, equal-sized partitions $\mathfrak{P}_1, \ldots, \mathfrak{P}_L$
3. Create additional datasets $\mathfrak{M}_1, \ldots, \mathfrak{M}_L$ where $\mathfrak{M}_i = \mathfrak{P}_i$ with Y missing $\forall x_i \in \mathfrak{P}_i$
4. $\mathfrak{D}_l = \mathfrak{M}_l \bigcup_{\substack{j=1, \\ j \neq l}}^{L} \mathfrak{P}_j, l = 1, \ldots, L.$
5. For $\omega = 1, \ldots, \Omega$ do:
6. Execute **Impute**($\mathfrak{D}_l$, $chain$, Γ, $burn$, lag) $\forall\, l = 1, \ldots, L$
7. $\forall\ x_i \in \mathfrak{M}_l \subset \mathfrak{D}_l$, $Impute()$ calculates Γ imputed values for the dependent variable Y of instance x_i denoted $\hat{Y}^{(\lambda,\omega,\gamma)}(x_i)$, $\gamma = 1, \ldots, \Gamma$.
8. End
9. End
10. $\hat{Y}(x_i) = (\Lambda \times \Omega \times \Gamma)^{-1} \sum_{\lambda=1}^{\Lambda} \sum_{\omega=1}^{\Omega} \sum_{\gamma=1}^{\Gamma} \hat{Y}^{(\lambda,\omega,\gamma)}(x_i)$.
11. $\forall\ x_i \in \mathfrak{D}$, calculate the absolute error (ϵ_a) and relative error (ϵ_r) between $\hat{Y}$ and Y:

$$\epsilon_a(x_i) = \left|\hat{Y}(x_i) - Y(x_i)\right| \qquad \epsilon_r(x_i) = \left|\frac{\hat{Y}(x_i) - Y(x_i)}{Y(x_i) + 1}\right|.$$

12. Instances with larger values for either ϵ_a or ϵ_r contain a relatively larger amount of noise.

Figure 3. Quantitative Noise Detection

number of burn-in iterations.

$Impute()$ will execute several times as determined by the parameter Ω. Due to the random nature of the MI procedure, each execution will generate slightly different imputed values. Each execution ω will calculate Γ imputed values for the instances with a missing value for Y. Additionally, the entire process (Lines 2 to 8) is executed multiple times as determined by the parameter Λ. Multiple executions Λ of the entire procedure are used to reduce any potential bias introduced due to one particular random selection.

The imputed value for the outcome variable for instance x_i is denoted by $\hat{Y}^{(\lambda,\omega,\gamma)}(x_i)$. Each instance has $\Lambda \times \Omega \times \Gamma$ imputed values for Y. On Line 10, the average imputed value $\hat{Y}(x_i)$ over all of the executions of the entire procedure is computed for each instance. The absolute (ϵ_a) and relative (ϵ_r) errors are calculated for each instance. Instances where ϵ_a or ϵ_r are small are less noisy than instances where the error is large. This procedure, therefore, returns a ranking of instances relative to the amount of noise contained in the dependent variable.

5.3 A Hybrid Approach to Quantitative Outcome Correction

The hybrid technique proposed for the correction of a quantitative outcome variable is based on the components discussed in 5.1 and 5.2. Our procedure utilizes two different approaches to detect and correct noise in the dependent variable. The output of these two components is combined by our hybrid procedure to calculate a cleansed value for the dependent variable.

By setting AOI equal to $nfaults$, our procedure ranks instances relative to the noise contained in a single attribute, which in this case is the dependent variable. In particular, $\Delta(x_i) = rank_Y(x_i) - rank(x_i)$ is calculated for each instance $x_i \in \mathfrak{D}$. Alternative values for Y were derived for each instance with $\Delta(x_i) > 0$ using imputation and regression. Denote the estimated values for those instances determined to contain noise relative to the AOI as $\hat{Y}_{AOI}$.

The second component in our hybrid approach uses the MI quantitative noise detector proposed in 5.2. The alternative value calculated by our technique is denoted by $\hat{Y}_{MI}$ instead of $\hat{Y}$, which is used in Figure 3, to distinguish $\hat{Y}_{MI}$ from $\hat{Y}_{AOI}$. Note that $\hat{Y}_{MI}$ was computed for each instance in the dataset $\mathfrak{D}$. This is different than $\hat{Y}_{AOI}$, which was only computed for those instances with $\Delta(x_i) > 0$. Based on the alternative value $\hat{Y}_{MI}$ and the original value Y, ϵ_a and ϵ_r were computed for each instance in $\mathfrak{D}$. Lastly, noisy instances are identi ed and the correct value Y^c for the dependent variable Y is derived utilizing the information provided by these two components.

In the case study considered in this work, CCCS was the dataset to be cleansed. The MI noise detector used a sequential chain with parameters $\Gamma = 15$, $lag = 100$, $burn = 100$, $\Lambda = 3$ and $\Omega = 11$. Based on previous experiments, these parameters were deemed reasonable for the CCCS dataset. After applying our hybrid procedure, the values for the 8 independent variables, the original value for Y, the estimated values $\hat{Y}_{AOI}$ and $\hat{Y}_{MI}$, and the statistics Δ, ϵ_a and ϵ_r were presented to a software engineering domain expert. After considering the alternative values, $nfaults$ was cleansed for 81 instances in the dataset. The expert maintained careful oversight of the entire noise cleansing process, ensuring that the cleansed instances were in fact noisy.

A hybrid approach to data cleansing is proposed due to the dif culty of the problem. Generally speaking, hybrid approaches may obtain better performance than their individual components, hence our reliance on numerous procedures for data cleansing. Any data cleansing procedure should be careful not to modify data that is a priori relatively clean. The inadvertent injection of noise into previously clean instances during the cleansing process is obviously undesirable. By relying on multiple techniques for both the detection and correction of noise, con dence in the cleansing process increases. Furthermore, cleansing of the CCCS dataset was performed under the careful guidance of a domain expert. Using 10-fold cross validation, regression models were constructed using both the original and cleansed versions of the CCCS dataset. The error statistics for the regression model constructed using the cleansed data were signi cantly lower than those of the regression model constructed on the original CCCS dataset. Based on the involvement of the domain expert and the results obtained from the linear regression models, we can therefore conclude that the data cleansing process applied to the CCCS dataset was carried out properly and that the resulting data is indeed relatively cleaner than the original.

Polishing Noise in Continuous Software Measurement Data

Taghi M. Khoshgoftaar*
Christopher Seiffert
Jason Van Hulse

Department of Computer Science and Engineering
Florida Atlantic University, Boca Raton, Florida

Abstract

Low quality data can have an adverse effect on any data analysis or decision-making tasks based on the data. Consequently, high quality data is essential when performing data mining tasks. We present a useful modification to ***Polishing****, making it applicable to continuous data – it was originally designed to work only with categorical data. The practical appeal of Polishing is that it attempts to correct noisy data instead of simply eliminating the associated instances, an approach used by many noise filters. The elimination of noisy instances can cause the loss of valuable information, and is especially harmful when the dataset is small in size.*

Keywords: noise detection, data cleaning, attribute noise, software metrics, polishing, continuous data.

1 Introduction

Ensuring that data is of high quality is an important process prior to performing any data mining or knowledge discovery task in any application domain. It has been shown that the presence of noise in a dataset can have adverse effects on a machine learner's ability to correctly identify patterns in the data. In the context of software fault prediction or classi cation, utilizing a high quality software measurement dataset is essential. Previous studies using software measurement data have demonstrated that classi ers built on noisy data may have a higher misclassi cation error rate when applied to unseen or test data [2].

Noise can be introduced from any number of sources. For example, data entry errors or measurement problems are two common causes. Generally speaking, two types of noise can occur in a given dataset: class noise and attribute noise [1, 5]. Class noise, or labeling errors, occur when the dependent attribute is incorrectly recorded. Attribute noise involves incorrect values in the independent attributes. *Polishing* [5], the method studied throughout this paper, attempts to address both types of noise.

There are three primary approaches used to handle noise. One possibility is to use classi ers that are robust in the presence of noise. The second method identi es records that are suspected of containing noise, as in the Ensemble Filter [1]. The noisy instances can either be removed completely from the dataset or they can be treated by some separate mechanism. *Polishing* attempts to correct noisy data [5]. In contrast to the noise elimination techniques, *Polishing* has the added bene t that no records are removed from the dataset, preventing the loss of information as a result of data cleansing. This bene t is more critical when dealing with smaller datasets where eliminating even a few records can lead to a substantial loss of vital information.

This study introduces a modi ed procedure allowing for the application of *Polishing* on continuous data. This is done by using *relative error* to determine the *differentness* (de ned in Section 2.2) between two continuous data values. Our investigation is based on a case study using a real-world software measurement dataset that is known to contain noise. The experiments conducted in this work were overseen by a software engineering expert with many years of domain experience. Our previous experience has demonstrated the importance of close expert supervision during the evaluation of noise-handling procedures.

The remainder of this paper is organized as follows. Section 2 describes the *Polishing* procedure and our modi cations. Section 3 provides a system description, Section 4 shows the empirical results and Section 5 concludes our study including suggestions for future work.

2 Methodology

2.1 Polishing

Polishing is a two stage procedure as introduced and described by Teng [5]. The rst phase, *prediction*, consists of switching the roles of the class attribute and a target independent attribute. Using the class attribute and the re-

*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.

maining independent attributes, the value of the target independent attribute is estimated. This is done with a 10-fold cross validation where each fold has a similar distribution of the respective classes. This phase is repeated so that each independent attribute is treated as the target independent attribute. The nal result is an estimation for each attribute which will be used in the next phase, *adjustment* phase.

The *adjustment* phase begins with performing 10-fold cross validation on the training dataset. For each instance misclassi ed we "selectively change" [5] the attribute values so that the instance is correctly classi ed by a majority of 10 classi ers built by the 10-fold cross validation. The procedure attempts to change as few attribute values as possible to minimize the risk of introducing new noise.

The attributes are then sorted from most to least predictable based on the *prediction* phase, where those with the lowest misclassi cation rate are considered the most predictable. For those attributes where the estimated value is different from the actual value, the attributes are selected for adjustment in order of attribute predictability.

Additional adjustments are made until the instance is either correctly classi ed by a majority of the 10 classi ers or some stopping criterion is met. If the stopping criterion is met, the class label is changed to the most commonly predicted class among the 10 classi ers. In this case, the attributes remain unadjusted.

2.2 Modification to Polishing

To apply *Polishing* using continuous data we de ne *differentness* as a criterion for determining the degree of difference between two continuous data values. We use the *relative error* measure to determine *differentness*. If relative error is greater than a user de ned threshold, then a prediction is considered to have suf cient *differentness* from the original value. This not only allows the *Polishing* procedure to be applied to continuous data, but also allows the user some e xibility in choosing the level of aggressiveness in adjusting the attribute values.

$$RelativeError = \left| \frac{OriginalValue - PredictedValue}{OriginalValue} \right|$$

By selecting a larger or more conservative threshold fewer attribute values will be changed, minimizing the risk of introducing new noise. However, the tradeoff for using a more conservative threshold is that the procedure is likely to detect less noise. Alternatively, a smaller or more aggressive threshold would result in more noise being detected. However, since more adjustments are being made the potential for introducing new noise also increases. Our modi cation allows *Polishing* to be used with both continuous and categorical data.

3 System Description

The experiments conducted in this case study used a dataset (JM1-2863) derived from a software measurement dataset of a NASA software project written in C. The original dataset consisted of $10,883$ instances [3]. Software metrics data was collected at the function level; hence a program module or instance was de ned as a function or subroutine. After removing instances with identical attributes but different class labels the dataset was reduced to $8,850$ observations, and denoted as JM1-8850.

The software metrics (independent variables or attributes) in the JM1-8850 dataset included McCabe Metrics, derived and basic Halstead Metrics, Line Count and Branch Count Metrics for a total of 21 attributes. The eight derived Halstead metrics were not used in our study, leaving 13 software metrics remaining.

The JM1-8850 dataset also contained a dependent variable, Class, with values nfp (not fault prone) and fp (fault prone). An instance was labeled fp if it contained at least one software fault and nfp otherwise. The JM1-2863 dataset used in our case study was constructed from JM1-8850 using clustering techniques and input from a domain expert with many years of experience in both the software engineering eld and using the JM1 datasets.

In a related study [7], an unsupervised clustering technique (*k*-means) was used to cluster the JM1-8850 instances. Some descriptive statistics for the entire dataset and each cluster were computed, such as mean, standard deviation, etc. The software engineering expert then labeled (nfp and fp) those clusters for which he was completely con dent, resulting in the JM1-2863 dataset. This dataset contained $2,445$ instances with class nfp and 418 instances with class fp. In JM1-2863, 235 of the $2,445$ nfp instances and 183 of the 418 fp instances contained class noise. In other words, the expert-assigned (i.e., clean) label did not match the (i.e., noisy) label originally provided for those 418 instances in the dataset.

4 Empirical Case Study

The *Polishing* procedure was repeated by varying the parameter *changes* and the threshold *t*. The *changes* parameter indicates the maximum number of independent attributes that can be changed in a given instance. The threshold, *t*, indicates the relative error required for a predicted attribute value to be considered different from the original value. If *changes* number for independent attributes are adjusted and the given instance still cannot be correctly classi ed then the dependent (class) attribute is adjusted instead.

4.1 Building Classification and Prediction Models

The learning algorithm used in our experiments is Reduced Error Pruning Tree (REPTree). This fast decision tree learner uses information gain/variance to build a tree and prunes it using reduced error pruning. Reduced Error Pruning [4] reserves a portion of the data as pruning data. A tree is built on the training data and tested on the pruning data. If, after removing a subtree, the model performs as well or better than the original, the subtree is pruned from

the original model. The optimal classi cation model was built using WEKA [6] by varying the minimum variance proportion, minimum instances per leaf node, number of folds for pruning, and the cost ratio with the goal of balancing false positive and false negative errors. Any learner can be used with the polishing procedure, but we chose to use REPTree due to its simplicity and the common usage of decision tree learners in related research.

4.2 Expert-Based Analysis

Since the dataset being used is a real-world data set with no injected noise, the use of an expert is required to analyze the results. The expert, with over 15 years experience in the software engineering domain, examined the data set to identify noise and to judge the quality of the adjustments made to attribute values. We examine the results of *Polishing* separately for class and attribute noise. It should be noted that the expert is used only to analyze the results of the procedure, and is not required as part of our modi ed *Polishing* procedure.

4.2.1 Attribute Noise

All instances indicated by the procedure as containing attribute noise were inspected by the expert. Each attribute value in each detected instance was labeled as either "red", "yellow"or "white" by the expert. A "white" attribute value has a value which is indicative of the class and agrees with the other attributes (i.e., no noise). A "yellow" attribute value is one where the expert sees some discrepancy either between the attribute and the class or between the attribute and other attributes in that instance (i.e., minor noise). A "red" attribute is one for which the value is very different from what is expected (i.e., severe noise). The *Polishing* technique is analyzed based on its effectiveness and ef - ciency in detecting both "red" and "yellow" attribute values.

In addition, the expert analyzed every adjustment made by the procedure. Each adjustment was categorized as "good", "bad" or "neutral". A "neutral" change is one that, while the relative error is large enough to cause the adjustment, the expert does not feel the adjustment will have any real impact on the data. In other words, although the prediction is "different" from the original value, the expert feels that both the prediction and the original value are legitimate based on the other attribute values in the record. A "bad" adjustment is one where the expert believes the original value was better than the adjusted value, and a "good" change is one where the expert believes the adjusted value is better than the original value.

This information is used to analyze the changes based on %good, %bad, %neutral as well as the *Net Reduction* of noise in the dataset.

$$NetReduction = \frac{N_{GC} - N_{BC}}{N_{NA}}$$

where N_{NA} is the number of noisy attributes among the inspected instances, N_{GC} is the total number of "Good" changes, and N_{BC} is the total number of "Bad" changes.

Table 1. Number of instances detected as having class or attribute noise

t	$changes$	Insts Class	Insts Attr
0.05	1	706	0
	3	685	21
	5	638	68
0.1	1	704	2
	3	657	49
	5	626	80
0.25	1	677	29
	3	652	54
	5	644	62

4.3 Noise Detection and Correction

The JM1-2863 dataset is known to contain 418 instances with class noise. The effectiveness and ef cienc y of *Polishing* with respect to class noise are reported in Table 3 and discussed in more detail in Section 4.3.2. Effectiveness and ef cienc y are de ned as follows.

$$Effectiveness = \frac{N_{CNF}}{N_{CN}}$$

$$Efficiency = \frac{N_{CNF}}{N_{CNI}}$$

where N_{CNF} is the number of instances correctly identi ed as having class noise, N_{CN} is the total number of instances in the dataset with class noise, and N_{CNI} is the total number of instances identi ed as having class noise.

Table 1 shows the number of instances detected as having class noise (Insts Class) and attribute noise (Insts Attr) for different combinations of the threshold t and $changes$.

In the experiment where *t*=0.1 and *changes*=5, 626 instances were determined by *Polishing* to have class noise. Using Table 3 we can see that for this experiment the effectiveness was 77.27% and the ef cienc y was 51.60%. This means that of the 418 instances known to have class noise, 77.27% (323) were caught by polishing and of the 626 instances in which *Polishing* changed the class, 51.60% of these (323) were correct changes. The results with respect to class noise are more closely examined in Section 4.3.1.

Table 1 also shows the number of instances identi ed as having attribute noise in each experiment. Notice that the total number of instances identi ed as having noise (class and attribute noise combined) remains the same (706) for each experiment because the base classi cation lter used does not depend on the parameters which are varied throughout the experiments. However, what does change is the number of instances selected as having either class noise or attribute noise. The results for those instances determined to have attribute noise are more closely examined in Section 4.3.2

4.3.1 Results with Respect to Class Noise

Polishing's ability to detect class noise is compared with standard cross validation-based Classi cation Filters (CF). The effectiveness and ef cienc y of several Classi cation Filters are presented in Table 2 along with the average over

Table 2. Classification Filters for JM1-2863

Classi er	Effectiveness	Ef cienc y
SMO	89.23	40.06
LWLS	80.62	48.28
J48	80.86	46.05
JRipper	82.30	45.68
IBk	77.51	47.72
REPTree	83.73	45.56
Average	82.38	46.74

Table 3. Effectiveness and Efficiency of *Polishing* experiments for Class Noise

		changes		
t	*Performance*	1	3	5
0.05	Effectiveness	79.19	78.71	77.99
	Ef cienc y	46.88	48.03	51.10
0.10	Effectiveness	79.19	77.99	77.27
	Ef cienc y	47.02	49.62	51.60
0.25	Effectiveness	78.95	77.51	77.27
	Ef cienc y	48.74	49.69	50.16

the six lters. For each of the six lters, models were built using 6 different classi cation techniques with a modeling selection strategy to maintain relative equality between the false positive and false negative error rates.

The Sequential Minimal Optimization (SMO) is an optimized implementation of support vector machines. The LWLS learner is the locally weighted learning scheme as applied to a decision stump classi er . The J48 classi er is an implementation of the commonly used C4.5 decision tree learner. The JRipper classi er is the rule-based learning algorithm, Repeated Incremental Pruning to Produce Error Reduction (Ripper). The IBk classi er is an instance-based learner with 'k' nearest neighbors and uses a majority voting scheme [6].

Comparing Tables 2 and 3, the performance of *Polishing* and Classi cation Filters are very similar with respect to class noise. This is expected since *Polishing* uses a CF to decide which instances are candidates for adjustment. At best, *Polishing*'s effectiveness can only be as high as the CF on which is it based. In our case, that Classi cation Filter is based on the REPTree classi er . Comparison with REPTree (which is similar to the other classi ers) shows that this is indeed the case, with *Polishing*'s highest effectiveness (78.71%) being lower than the effectiveness of the REPTree CF (83.73%).

Naturally, this raises the question: What bene t does *Polishing* have over the simple CF? In addition to detecting and correcting class noise, *Polishing* has the added bene t of detecting and correcting attribute noise. In some cases, instances with legitimate class noise will erroneously have their attributes, rather than class, changed. This results in a lower effectiveness with respect to class noise. However, in other cases, *Polishing* will recognize that some instances which were found to have class noise by the CF actually have attribute noise instead. This results in both a higher ef cienc y with respect to class noise and a cleaner overall dataset when attribute noise is considered, as will be shown in Section 4.3.2. Since the procedure's effectiveness and ef cienc y are closely related to that of the CF on which it is based, it stands to reason that if this CF is replaced by a technique which is more effective and/or ef cient, then the *Polishing* procedure will likely become more effective and/or ef cient.

Table 4. Effectiveness in detecting noisy attributes

		%			
t	*changes*	EfvRed	EfvYel	EfvTot	Insts
0.05	1	-1	-1	-1	0
	3	100.00	38.10	66.67	21
	5	85.45	70.69	77.88	68
0.1	1	100.00	20.00	33.33	2
	3	85.71	58.33	71.83	49
	5	85.45	72.06	78.05	80
0.25	1	82.61	32.36	53.70	29
	3	83.78	46.00	62.07	54
	5	85.37	51.72	65.66	62

4.3.2 Results with Respect to Attribute Noise

In addition to correcting class noise, *Polishing* also attempts to detect and correct attribute noise. The domain expert examined each instance containing attributes values that were adjusted by the *Polishing* experiments. Attribute values were labeled as "Red", "Yellow" or "White" as described in Section 4.2.

Table 4 shows the effectiveness in detecting attribute noise based on these labels. "EfvRed" is the effectiveness in detecting attributes labeled as "Red". "EfvYel" is the effectiveness in detecting attributes labeled as "Yellow", and nally "EfvTot" is the total effectiveness in detecting both "Red" and "Yellow" attributes. "Insts" is the number of instances found to have attribute noise for the given *t* and *changes*. A table entry of -1 indicates that no noise was detected (since there were no instances found to have attribute noise).

For *t*=0.05 and *changes*=5, 68 instances received attribute adjustments, and the *Polishing* procedure detected 85.45% of the "Red" attribute values and 70.69% of the "Yellow" attribute values. On the other hand, when *t*=0.25 and *changes*=1, 29 instances had attribute adjustments, and only 82.61% of the "Red" and 32.36% of the 'Yellow" attribute values were detected. However, the ability to correct noise without introducing new noise is more interesting than simply detecting noise.

The domain expert examined all attribute adjustments and labeled them as "Good", "Bad" or "Neutral" as described earlier in Section 4.2. The changes labeled as "Good" are further broken down into two types – *corre* are "Good" changes on attributes labeled as either "Red" or "Yellow" by the expert as described above; *imprv* includes all *corre*, but also includes changes to attributes not specifically labeled as noise by the expert, replacing a reasonable

Table 5. Comparison of Attribute Changes

t	$changes$	corre	imprv	bad	neutr	insts
0.05	1	-1	-1	-1	-1	0
	3	40.74	48.15	18.52	33.33	21
	5	31.08	41.83	29.88	28.29	68
0.10	1	100.00	100.00	0.00	0.00	2
	3	43.24	54.05	27.93	18.02	49
	5	38.94	50.00	32.30	17.70	80
0.25	1	100.00	100.00	0.00	0.00	29
	3	70.42	74.65	18.31	7.04	54
	5	60.61	68.69	26.26	5.05	62

Table 6. Net Noise Reduction

t	$changes$	reduc	imprv	noise	insts
0.05	1	0.00	0.00	0.00	0
	3	9.09	40.74	19.70	21
	5	2.27	31.08	66.67	68
0.1	1	1.52	100.00	1.52	2
	3	12.88	43.24	38.64	49
	5	11.36	38.94	72.73	80
0.25	1	21.21	100.00	21.97	29
	3	28.03	70.42	40.91	54
	5	25.76	60.61	49.24	62

attribute value with an even better (more typical) value. Table 5 shows percentage of changes that are classi ed as either *corre*, *imprv*, *bad* or *neutr*.

The information in Table 5 certainly seems to favor the higher t value of 0.25. Both *corre* and *imprv* are significantly higher, and *bad* and *neutr* values are consistently lower. For example, while at most about 7% and 26% of the changes with t=0.25 are *neutr* and *bad*, respectively, up to about 18% and 32% are *neutr* and *bad* with t=0.1. Also, with t=0.25 over 60% of the changes are considered *corre* and over 68% are considered *imprv* while for t=0.1 less than 44% are *corre* and less than 55% are *imprv*.

One of the goals of *Polishing* is to change the original data (those detected as noisy) as little as possible so as to minimize the potential of introducing new noise. This is illustrated in Tables 5 and 6. While the higher t value may detect less noise, the quality of the adjustments is higher and therefore the overall quality of the resulting dataset is better.

Table 6 shows these results more clearly. In the table, *reduc* (de ned as *NetReduction* in Section 4.2.1) shows the overall performance of the procedure in cleaning the data; *imprv* is the percentage of changes made which are not "Bad" or "Neutral"; and *noise* is the effectiveness in detecting noise for that experiment. Note that these gures represent percentages of all noise found in all instances inspected throughout the course of all experiments. In contrast, Tables 4 and 5 re ect only the instances inspected for each individual corresponding experiment.

Table 6 shows that despite predicting less noise, the experiment with t=0.25 results in a cleaner overall dataset. For example, using t=0.25 the net reduction in noise is over 21%, while the highest net reduction among the other t values is only about 13%. This is due to the higher percentage of correct changes. It should be noted that the "known noise" that these results are based on is not necessarily all noise in the dataset. Instead, it is all noise found by the domain expert in the instances inspected. While this can have an effect on the actual values provided, the concept will remain the same. The experiments with t=0.25 would still result in a cleaner overall dataset even if all noise was truly known.

5 Conclusion

This work examines the use of *Polishing* on continuous data using *relative error* as a basis for judging whether a prediction is suf ciently different from the original value. The use of expert-based analysis provides a better understanding of the quality of the attribute changes made by the procedure, in contrast to injecting noise randomly into a dataset that may or may not already contain noise. Results are also presented on varying key parameters to assist a domain expert in choosing the correct values for their domain.

The overall quality of the dataset improved after *Polishing* was applied, implying the dataset was cleaner after the procedure. The least conservative approach (t=0.05) caught about the same quantity of noisy attribute values as the t=0.10 experiments. However, the quality of the adjustments was lower, resulting in a lower reduction in noise.

The most conservative approach, using t=0.25, yielded the cleanest dataset of all the experiments despite making the fewest changes. Using such an approach it is less likely to inject new noise, and while less noise is detected, one can be con dent that the noise that is detected is handled properly. This re ects *Polishing*'s goal of being careful not to insert additional noise into the dataset.

Future work will continue to investigate the effectiveness of *Polishing* on additional data sets with continuous data. Experiments with both real world and arti cial datasets will be performed to con rm the ndings of this paper.

References

[1] C. E. Brodley and M. A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.

[2] T. M. Khoshgoftaar and N. Seliya. The necessity of assuring quality in software measurement data. In *Proceedings of 10th International Software Metrics Symposium*, pages 119–130, Chicago, IL, September 2004. IEEE Computer Society.

[3] T. M. Khoshgoftaar, S. Zhong, and V. Joshi. Noise elimination with ensemble-classi er ltering for software quality estimation. *Intelligent Data Analysis: An International Journal*, 9(1):3–27, 2005.

[4] J. R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221–234, 1987.

[5] C. M. Teng. Correcting noisy data. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 239–248, 1999.

[6] I. H. Whitten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with JAVA Implementations*. Morgan Kaufmann, San Francisco, CA, 2000.

[7] S. Zhong, T. M. Khoshgoftaar, and N. Seliya. Analyzing software measurement data with clustering techniques. *IEEE Intelligent Systems*, pages 22–27, March 2004.

3D Visualization of Class Template Diagrams for Deployed Open Source Applications

Benjamin N. Hoipkemier, Nicholas A. Kraft and Brian A. Malloy
Computer Science Department
Clemson University
{bhoipke,nkraft,malloy}@cs.clemson.edu

Abstract

In this paper, we exploit the g^4re infrastructure to facilitate comprehension of generic programs written in the C++ language, including class templates, instantiated class templates and specialized class templates [7]. We evaluate our 3D visualization technique using ten deployed open source applications and provide analysis about the frequency and efficiency of generic programming in these applications.

1. Introduction

The generic programming paradigm is becoming increasingly popular as an ancillary tool to object technology, with conferences, seminars and other literature appearing to address these growing concerns [2, 4, 6, 13]. *Generic programming* deals with nding abstract representations of ef- cient algorithms and data structures and expressing these structures in an adaptable interoperable manner [6]. The canonical example of generic programming is the Standard C++ library (STL); however, recent libraries, such as Boost, Loki and Blitz++, rely heavily on both generic and generative programming to produce code that is more general, more ef cient and more easily incorporated into existing applications than their non-generic counterpart [1, 11]. In recognition of the importance of generics, they have been recently introduced into both the Java 5 and C# 2.0 programming languages [10, 12].

In this paper, we exploit the g^4re infrastructure to facilitate comprehension of generic programs written in the C++ language, including class templates, instantiated class templates, and specialized class templates [7]. Unlike other systems that use the source code to construct UML class diagrams, our system uses an abstract semantic graph (ASG), which contains additional information about templates not included in the source code. Our visualization technique leverages 3D class diagrams with classes and class templates presented in the X-Y plane, and instantiated, and specialized classes presented along the Z axis.

To evaluate our 3D visualization technique, we examine ten deployed open source applications and provide analysis about the frequency and ef cienc y of generic programming in these applications. Our case study illustrates the type of information that can be extracted using our visualizer that cannot be obtained using previous class visualization techniques. Moreover, our approach is the only reported work that includes template considerations into the visualization of class diagrams.

2. Background

There are several viable graphical source code representations and the most common of these are the abstract syntax tree (AST) and the abstract semantic graph (ASG). The AST is a pruned parse tree with many of the unnecessary productions removed from the tree. An ASG is an AST that is adorned with additional nodes and edges to represent semantic information such as the scope of identi ers or the types of variables. The ASG has a distinct advantage over analysis of a program at the source, AST or object code level: information about template instantiations, template specializations and template partial specializations is readily available in the ASG.

The g^4re tool chain exploits the gcc Abstract Semantic Graph (ASG), GENERIC, to provide an Application Programmers Interface (API) to facilitate easy access to information about types, including classes, templates, template instantiations, and template specializations, as well as scopes, variables, functions and statements. The advantages of the g^{4}re tool chain is that it can analyze any program that can be compiled by the gcc C++ compiler. We use the g^{4}re tool chain to create an API that abstracts the important information about the C++ application. Our 3D visualizer uses this information to build class diagrams.

To appear in The Eighteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'06)
San Francisco Bay, USA, July 5-7, 2006

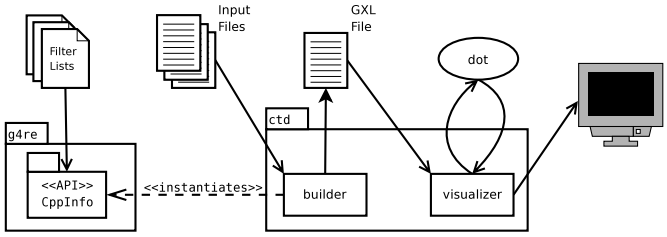


Figure 1. System overview. *This figure illustrates the important components in our 3D visualization of class template diagrams.*

3. System Overview

In this section we provide an overview of our system, including discussion of the important components and subsystems that we use to build a 3D class template diagram.

Figure 1 summarizes our visualization system, where a C++ program consisting of input files, shown in the middle left of Figure 1, are read by the builder component of our system. The builder component is part of the ctd subsystem, shown in the lower middle of the figure.

The builder component instantiates a CppInfo API, shown in the lower left of the figure, which provides information about class templates, class template instantiations and class template specializations for the input program under study. The builder component generates a class template diagram for the program and writes the diagram to a file in GXL format, shown in the upper middle of the figure. Finally, the GXL encoded class template diagram is used as input to our visualizer component, which uses the *graphviz* dot program to build a partial layout of the class diagram. Finally, the visualizer produces a 3D picture of the class template diagram, with classes and parent class templates appearing in the X-Y plane, and template instantiations, and specializations appearing along the Z axis.

The visualizer component reads GXL files output by builder, and builds a three dimensional environment for the class template diagram. After the visualizer parses the classes in the GXL file, the *dot* tool then generates a 2D layout of the classes and parent class templates. The visualizer component then uses OpenGL to create a 3D scene consisting of the classes and parent class templates in the X-Y plane, and template instantiations along the Z axis.

The user can maneuver around the scene in real time using our graphical visualizer tool. Within the OpenGL environment all classes and instantiations are represented by boxes. Class information about data attributes and functions is included on the side of the box. This is information as it would appear in a UML Class Diagram. Inheritance edges are represented as lines with the ancestor portion of the line drawn in green, and the descendant portion drawn in brown. Classes are illustrated as yellow boxes, while template classes are colored light blue. Template instantiations are colored in red and descend down the Z axis away from their parent template. Instantiations are connected by a blue line running from the template instantiation to the parent template. Template instantiations that have the same parent template reside on the same X-Y plane.

4. Results

The experiments reported in this section were run on a workstation with an *AMD Athlon64 3000+* processor, 1024 MB of PC3200 DDR RAM, and a 7200 RPM SATA hard drive formatted with version 3.6 of the *ReiserFS* filesystem, running the Slackware 10.1 operating system. The programs were compiled using *gcc* version 3.3.6.

4.1 The test suite of applications and libraries

Table 1 lists the ten deployed open source applications and libraries, or test cases, that form the test suite that we use in our study, together with important statistics about each test case[1]. The header of the table lists the names that we use to refer to each of the test cases: *Doxygen*, *FOX*, *FluxBox*, *HippoDraw*, *Jikes*, *Keystone*, *Licq*, *Pixie*, *Scintilla*, and *Scribus*. The remaining three rows of data in Table 1 list relevant details of the test cases. The second row of the table lists the version number and the third row lists the number of C++ translation units. Finally, the fourth row lists the approximate number of thousands of lines of non-commented, non-preprocessed lines of code.

4.2 Time Requirements of Visualization

The data in Table 2 summarizes the time requirements of our visualization system. The columns list the ten test cases and the four rows list results for the three phases of class template diagram construction, with the last row listing total times. All data are wall-clock timings in seconds.

The first row of Table 2, Instantiate API, lists timings for the first phase of our system: instantiation of the CppInfo API for each of the test cases, illustrated on the left of Figure 1. These timings include the time to parse the *.cpp.tu.gxl.gz* for the open source application, reconstruction of the GENERIC ASGs, transformation to the CppInfo API instances, and linking of the API instances.

The second row of Table 2, Generate CTD GXL, lists timings for the second phase of our system, including construction of a class template diagram and timings to write

[1] Additional information about each test case is available in our online repository.

Test Case	Doxygen	FOX	FluxBox	HippoDraw	Jikes	Keystone	Licq	Pixie	Scintilla	Scribus
Version	1.4.4	1.4.17	0.9.14	1.15.8	1.22	0.2.3	1.3.0	1.5.2	1.66	1.2.3
C++ Translation Units	69	245	107	249	38	52	28	78	78	110
NCLOC ($\approx$ K)	170	110	32	55	70	16	36	80	35	80

Table 1. Testsuite. *This table lists the ten test cases that we use in our study, including the version, the number of translation units, and the approximate number of non-commented, non-preprocessed lines of code (NCLOC).*

Test Case		Doxygen	FOX	FluxBox	HippoDraw	Jikes	Keystone	Licq	Pixie	Scintilla	Scribus
Phase 1	*Instantiate API*	286.02	507.43	492.66	749.93	274.77	241.85	92.46	115.17	52.91	754.62
Phase 2	*Generate CTD GXL*	6.43	3.07	2.08	1.77	2.98	0.50	1.18	1.26	0.49	1.84
	Parse CTD GXL	1.03	0.40	0.32	0.28	0.44	0.09	0.20	0.23	0.06	0.24
	Generate Dot	0.34	0.11	0.06	0.09	0.10	0.03	0.04	0.06	0.03	0.06
Phase 3	*Parse Dot*	231.55	91.87	59.20	49.51	80.14	17.80	38.75	36.51	12.89	44.11
	Render	0.21	0.06	0.05	0.04	0.08	0.01	0.03	0.04	0.02	0.03
Total		525.58	602.94	554.37	801.62	358.51	260.28	132.66	153.27	66.40	800.90

Table 2. Time (s). *The data in this table summarizes the time requirements of our visualization system. The columns list the ten test cases and the four rows list results for the three phases of class template diagram construction, with the last row listing total times. All data are wall-clock timings in seconds.*

the diagram in GXL format, as illustrated in the center of Figure 1. The third row of Table 2, Parse CTD GXL, Generate Dot, Parse Dot, Render, lists timings for the third and nal phase of our system. These timings include reading and parsing the GXL generated in the previous phase, generating a Dot graph to obtain a 2D layout of the classes and parent class templates, reading and parsing the generated Dot graph and generation and rendering of the OpenGL 3D representation of our class template diagram. The nal row of Table 2 summarizes timings for all three phases.

4.3 CTD Summary

The data in Table 3 summarizes the kinds of classes in the test cases. The columns list the ten test cases and the three rows list results for the three kinds of classes in the test cases: the rst row is the number of classes, the second row is the number of class templates, and the third row is the number of instantiated classes derived from the class templates. Only three of the test cases contained ten or more class templates: Doxygen with 31, FluxBox with 11 and Pixie with 10 class templates. The number of template instantiations can be large; for example, Doxygen's 31 class templates generated 311 instantiated classes and Jikes, with only 6 class templates, generated 120 instantiated classes.

5. Case Study

Our visualization research permits us to obtain information about the system that is not readily available from inspection of the source code alone. Figure 2 shows the visualizer tool interface. There are two slider bars that allow us to control the azimuth and the elevation. Horizontal movement, vertical movement, and zoom is controlled through keyboard input. The default view illustrates all classes in the system, illustrated as cubes, together with inheritance and template association relationships forming the edges that connect the cubes. We provide options that allow the user to view only the template classes in the system, including instantiations; there are check boxes that allow the user to hide the background and text.

The area to the right of the controls in Figure 2 illustrate a class diagram of Pixie-1.5.2. The cubes aligned on the left side of the diagram lie on the Z-axis; These cubes represent template instantiations of the $CArray < T >$ template class. There are other class template instantiations along the Z-axis further down along the X-axis, shown to the right of the gure; however, the majority of instantiations in this application are derived from the $CArray < T >$ template class. This shows that a modi cation to $CArray < T >$ will have a profound impact on the Pixie-1.5.2 system.

6. Related Work

Eiglsperger et al. observed that previous approaches to class diagram layout employ a hierarchical approach, where all edges of the same type should point in the same direction [3]. For example, inheritance edges point upward in the diagram. They proposed a new approach using topology-shape-metrics as a basis for the layout. They showed that their layout incorporates o w information into the diagram and works well for all types of diagrams. However, they did

Test Case	Doxygen	FOX	FluxBox	HippoDraw	Jikes	Keystone	Licq	Pixie	Scintilla	Scribus
ClassNonTemplate	409	464	211	248	270	77	225	239	89	211
ClassTemplate	31	1	11	1	6	4	0	10	0	0
ClassTemplateInstantiation	311	4	69	9	120	14	0	44	0	0
Total	751	469	291	258	396	95	225	293	89	211

Table 3. Class Type Summary. *The data in this table summarizes the kinds of classes in the test cases. The columns list the ten test cases and the three rows list results: the first row is the number of classes, the second row is the number of class templates, and the third row is the number of instantiated classes derived from the class templates.*

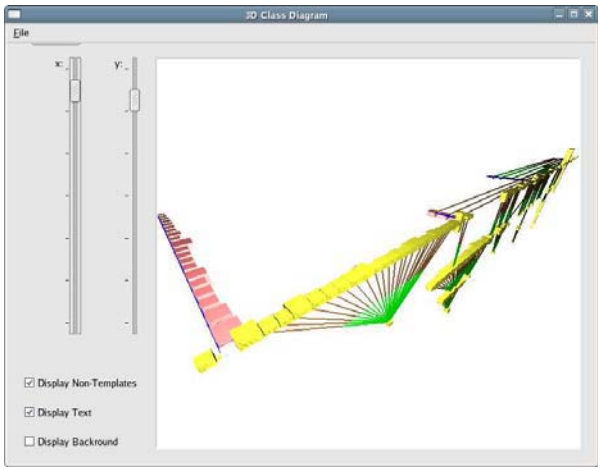


Figure 2. Pixie-1.5.2. *This figure shows the visualizer tool interface and a class template diagram for the Pixie-1.5.2 test case. The two slider bars permit control of the azimuth and the elevation. Horizontal movement, vertical movement, and zoom is controlled through keyboard input.*

not incorporate generics into their layout algorithm.

Gutwenger et al. distinguish two kinds of relationships in class diagrams: (1) inheritance, and (2) association, including aggregation and composition [5]. They identi ed ve anesthetic preferences for class diagrams that permit all directed edges of a component to follow the same direction. Their orthogonal layout is compact with a small number of crossing edges. However, they do not incorporate generics into their layout.

Malloy and Power exploit a molecular metaphor for 3D visualization of dynamic object relationships in Java applications [9]. They instrument Java bytecode to collect trace data, which is then analyzed and visualized in 3D using VRML. However, their approach does not accommodate Java generics.

Lewerentz and Simon present a metrics based approach to software visualization that supports quality assessment of large object-oriented software systems written in C++ and Java [8]. Their system utilizes a combination of software metrics data and program structure information to form a virtual information space. Their approach does not include visualization of C++ class templates or Java generics.

References

[1] A. Alexandrescu. *Modern C++ design: generic programming and design patterns applied*. Addison-Wesley Longman Publishing, Boston, MA, USA, 2001.

[2] H. Eichelberger and J. W. v. Gudenberg. UML description of the STL. In *First Workshop on C++ Template Programming, Erfurt, Germany*, October 10 2000.

[3] M. Eiglsperger, M. Kaufmann, and M. Siebenhaller. A topology-shape-metrics approach for the automatic layout of uml class diagrams.

[4] R. Glück and M. R. Lowry, editors. *Generative Programming and Component Engineering, 4th International Conference, 2005*. Springer, 2005.

[5] C. Gutwenger, M. Jünger, J. Kupke, S. Leipert, and P. Mutzel. A new approach for visualizing UML class diagrams. In *SoftVis '03*, pages 179–188, New York, NY, USA, 2003. ACM Press.

[6] M. Jazayeri, R. Loos, and D. Musser, editors. *Generic Programming: International Seminar, Dagstuhl Castle, Germany, 1998*. Springer-Verlag, Germany, 2000.

[7] N. A. Kraft, B. A. Malloy, and J. F. Power. Toward an infrastructure to support interoperability in reverse engineering. In 12^{th} *WCRE*, Pittsburgh, USA, Nov. 2005.

[8] C. Lewerentz and F. Simon. Metrics-based 3d visualization of large object-oriented programs. In *VISSOFT '02*, page 70, DC, USA, 2002. IEEE Computer Society.

[9] B. A. Malloy and J. F. Power. Using a molecular metaphor to facilitate comprehension of 3d object diagrams. In *VL/HCC*, pages 233–240, 2005.

[10] Microsoft Corporation. C# language speci cation. Third Edition, 10 March 2006.

[11] J. G. Siek and A. Lumsdaine. Essential language support for generic programming. In *PLDI '05*, pages 73–84, New York, NY, USA, June 2005. ACM Press.

[12] Sun Microsystems Inc. Java language speci cation, third edition. Version 5, 10 March 2006.

[13] T. L. Veldhuizen. Five compilation models for C++ templates. In *First Workshop on C++ Template Programming, Erfurt, Germany*, October 10 2000.

Parallel Monitoring of Design Pattern Contracts

Jason O. Hallstrom, Andrew R. Dalton
Clemson University
{jasonoh,adalton}@cs.clemson.edu

Neelam Soundarajan
Ohio State University
neelam@cse.ohio-state.edu

Abstract

Design patterns *are widely accepted as elements of best practice. We have previously shown that these benefits can be amplified by supplementing informal pattern descriptions with precise* pattern contracts*, and monitoring tools that identify runtime contract violations. In this paper, we present a parallel approach to monitoring pattern contracts that reduces the performance impact of runtime monitoring. We evaluate the performance of the approach relative to its sequential counterpart, and draw conclusions concerning its applicability to a range of systems.*

1 Introduction

Design patterns [7, 2, 18] have been widely adopted as an important component of software practice. They are valuable not only as reusable solutions that capture the design knowledge of expert communities, but as elements of an extended vocabulary useful in documenting software design. We have shown that the bene ts of design patterns can be ampli ed by supplementing informal pattern descriptions with precise *pattern contracts* [19, 22, 9]. The contracts capture the implementation requirements associated with a range of patterns, as well as the behavioral guarantees that result when those requirements are satis ed. To make the formalism more accessible to practitioners, and more applicable to commercial projects, we have also presented aspect-based tools that automate the generation of *pattern-centric* assertion checking code to identify runtime violations of the contracts underlying a design [22, 9]. Together, pattern contracts and *contract monitors* help to safeguard the design integrity of a system as it evolves.

In this paper, we focus on reducing the performance overhead introduced in monitoring a system's runtime behavior to determine whether the relevant pattern contracts are respected. The contributions of the paper are two-fold. First, we present the design and implementation of a parallel monitoring approach based on the use of *AspectJ* [12] and the *Java Thread Library* [20]. This is the rst parallel approach to monitoring pattern-centric behaviors to be discussed in the literature. Second, we evaluate the performance of the approach relative to its sequential counterpart. We consider runtime performance as a function of assertion complexity, and consider a range of execution environments. To be precise, we consider single processor execution, dual processor execution, and three levels of heap availability. We use the results of the analysis to draw conclusions concerning the applicability of both the parallel and sequential approaches to a range of system classes.

But why is performance a signi cant issue in the context of pattern-centric monitoring? One factor is that by improving monitor performance, the time required to execute pattern-centric test suites decreases. The time savings may be signi cant for systems with long-running test suites. More surprising, however, are the bene ts beyond testing — to deployed, *reusable* assets. To understand these bene ts, there are three observations to consider.

First, design patterns are commonly applied in the development of commercial frameworks, class libraries, and other reusable assets. The Java IO classes, for example, are designed according to the `Decorator` pattern to simplify the composition of IO services, such as buffering, encryption, and compression. The Swing API makes extensive use of the `Observer` pattern to assist in maintaining consistency across user interface elements. Similar pattern applications are found in other reusable assets, such as the C++ Standard Template Library, Microsoft's Foundation Class Library, and the .NET Framework. The ubiquity of patterns in this context means that the patterns must be unambiguously understood for the assets to be (re)used correctly.

Second, these assets are designed *defensively* to assist designers in localizing usage errors. The assertion checking code that identi es these errors imposes a performance penalty, but improves the accessibility of the assets to a large consumer base. Consider again the Java class library. A designer attempting to *pop* an element from an empty stack will be noti ed of the usage violation via an instance of `EmptyStackException` thrown by the target class. An instance of `IllegalArgumentException` will be thrown if a designer speci es a negative capacity for a collection, or an invalid orientation constant when constructing a page formatter. Similar checks are performed throughout the library. As we will see, design patterns impose requirements that are more subtle than those being checked in these examples, and are therefore even more likely to be violated. Hence, it is equally, if not more important that pattern-centric assertions be checked as well.

Finally, we observe that runtime ef cienc y plays a critical role in determining whether a reusable asset will be adopted. If an asset's performance is undesirable, potential users may favor redevelopment over reuse, creating a tension between ef cienc y and defensiveness. One result of

our work is to reduce this tension by reducing the overhead associated with pattern-centric monitoring without compromising the level of defensiveness assets provide.

Paper Organization. Section 2 presents examples of problems that may be encountered when reusing an asset if the patterns underlying its implementation are misunderstood. The parallel monitoring approach is presented in Section 3. The performance of the approach is evaluated in Section 4. Section 5 summarizes key elements of related work. Section 6 concludes with a summary of our contributions and pointers to future work.

2 Pattern Implementation Difficulties

To motivate the need for monitoring, it is useful to consider the dif culties that may be encountered when reusing an asset if the patterns underlying its design are not well understood. Consider, for example, using the Java Swing API to implement a graphical interface that includes a two-dimensional grid component. The JTable class is used for this purpose, and provides support for rendering and editing a two-dimensional grid. Each instance provides a graphical view into a *table model* that stores the data displayed by the grid; an instance of the TableModel interface serves this purpose. These components are designed to interact according to the *Observer* pattern. Instances of JTable *observe* instances of TableModel so that changes in the underlying model are re ected in the user interface. Stated in terms of the Observer pattern, instances of TableModel play the Subject role, and instances of JTable play the Observer role.

AbstractTableModel implements TableModel, and provides default method implementations that simplify development. Designers develop new models by inheriting from this class, and providing any additional methods appropriate to their application. A typical extension might, for example, involve the addition of accessor methods that manipulate the state of the model. To implement these methods correctly, the designer must have a clear understanding of the model's role in the Observer pattern, or her application will exhibit unexpected behaviors. Consider the addition of a setValue() method that updates the value stored within a particular cell. If this method modi es the state of the object but does nothing else, the user interface will display a peculiar defect. Modi cations will only be re ected in the user interface after the interface has been obscured by another window, or relocated to another region of the screen. Only then will the JTable instance query the state of its model to update the display. This defect is subtle and can be dif cult to debug (especially for novice designers) since it is manifested as an application that *sometimes* behaves properly.

The behavioral defect is a result of misunderstanding (or neglecting) the role of AbstractTableModel in this *specialization* of the Observer pattern. The requirements of the pattern dictate that if a Subject method *modifies* the state of a subject, its notify() method must be invoked to update each of its attached observers. In this example, fireTableCellUpdated() plays the part of notify(), and since a call to this method was omitted, the behavioral guarantee afforded by the pattern was not received. The observer is sometimes *inconsistent* with the state of its subject, violating the pattern's intent. A similar defect would occur if the state of the model was modi ed *after* the call to fireTableCellUpdated(). Or alternatively, if the addTableListener() method, used to attach observers to a JTable, was improperly overridden. These reuse errors share a common characteristic: They result from violating pattern requirements, and can be more easily detected, diagnosed, and corrected if pattern-centric assertion checks are included as part of an asset base.

3 System Design

The parallel monitoring approach is an extension of the sequential system introduced in [22, 9], and illustrated in Figure 1. In this approach, a *Monitor Generator* produces monitoring aspects in AspectJ [12] based on the pattern contracts underlying a system. When the aspects are woven with the original system source, the compiled system behaves as before, but generates suitable warnings in the presence of contract violations. The aspects produce this behavior by introducing assertion checking *advice* bound to the appropriate class methods. The bindings are realized as *pointcut* de nitions based on the *class-to-role* mappings speci ed as part of the contracts.

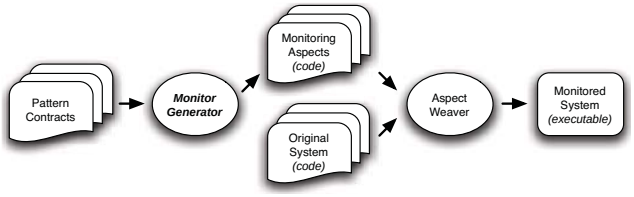


Figure 1: Monitor Generation Process

The parallel approach is similar in its use of aspect-oriented constructs. (It is not, however, integrated with the monitor generation framework — a point we will return to in Section 6.) The key difference involves the manner in which assertion checks are performed. Rather than directly performing the relevant assertion checks and bookkeeping operations, each advice block *defers* these tasks for later execution. More precisely, each block generates an assertion object that represents the appropriate assertion to be checked. This object stores the state information required to perform the check, and provides a check() method to initiate it. Each monitoring aspect includes a thread that manages a queue of *pending* assertions. The advice blocks populate this queue, and the checking thread depopulates the queue, executing the checks in parallel with the monitored system.

Before turning to an example of an assertion implementation, it is useful to consider elements of the checking work they perform. We begin by noting that there is a point of *pattern instantiation* at which an *instance* is thought to be constructed. In the case of the Observer pattern, this point coincides with the construction of a subject object. Once a pattern instance has been created, objects may *enroll* to participate in the instance. An observer may, for example, enroll in a pattern instance by invoking attach(ob) on the appropriate subject. Enrolled objects are required to abide by rules governing their interactions, while unenrolled objects are not. Hence, to avoid spurious checks, assertion objects must track the pattern instances active in a system, as well as the objects participating in those instances.

The assertion objects must also address pattern *specialization*. Consider the assertion object associated with the post-condition of Observer.update(). This method is required to leave the object in a state that is *consistent* with the state of its subject. But what exactly does *consistent* mean? Adopting a single de nition of the concept would be inappropriate since it would make the requirement irrelevant to systems that use a different notion of consistency. Our assertions are therefore expressed in terms of *auxiliary concepts*, relations involving one or more of the objects participating in a pattern. The speci cation of Observer, for example, de nes the *Consistent*(Subject,Observer) concept to capture the notion of consistency between a subject and an observer. The *Modified*(Subject,Subject) concept is similarly de ned, and captures the notion of *significant change* from one subject state to another. (A *significant change* must trigger a call to notify().) Concept de nitions are supplied on a per application basis to allow the assertions to be appropriately tailored. This is mirrored in the associated checking code. Each concept is represented by a boolean function used in implementing the relevant checks. Implementations of these methods are provided as appropriate to the system being developed.

A similar approach is used in checking assertions involving the state components required by a pattern. Consider the assertion object associated with the post-condition of Subject.attach(ob). The post-condition asserts that the observer passed as argument (ob) be added to the set of attached observers maintained by the subject. While checking this condition is conceptually trivial, the implementation of the check is complicated by the observation that different applications of the pattern will choose different realizations of the set component. A class might, for instance, maintain two Vector objects, the elements of which together represent the set of attached observers. To avoid imposing unnecessary specialization restrictions, we introduce a method for each state component de ned by a pattern, and implement our checking code in terms of these methods. Each method accepts an object as argument, and returns the appropriate *role field* when the object is viewed as an instance of its role. Implementations of these methods are provided as appropriate to the classes participating in a particular application.

In addition to checking state requirements, some assertion objects must check conditions imposed on the sequence of invocations that must occur during a method's execution. The notify() method, for example, is required to invoke update() on each attached observer. To check that such conditions are satis ed, we rely on the notion of a *trace* (or "*call sequence*") that records information about the invocations placed by a method. The trace is realized as a sequence of records, where each record corresponds to a single invocation, and records the name of the method invoked, the target of the invocation, and any relevant arguments. When checking a post-conditional assertion that involves call sequence conditions, the corresponding assertion object examines the relevant trace to determine whether the appropriate conditions are satis ed. This imposes overhead not only in checking these conditions, but in recording the relevant calls in the system trace.

```
1  public class AttachPost extends Assertion {
2   ...state elements referenced in post-condition...
3   public AttachPost(...) {
4     ...save state elements using role field accessors... }
5   public void check() {
6    Trace trace = traces.pop();
7    recordCall(sub,ATTACH);
8    boolean assertion =
9    obs.containsAll(obsPre) && obs.contains(ob)&&
10   (obs.size()==obsPre.size()+1) &&
11   !mod(subPre,subPst) && (trace.length()==1) &&
12   (trace.project(obs,UPDATE)==1);
13   checkCondition(assertion); } }
```

Figure 2: Assertion **Class Example**

Consider the AttachPost class shown in Figure 2. Instances of this class are associated with the post-condition of Subject.attach(). The constructor of the class executes in the thread of the monitored application, and is responsible for storing any state information necessary to check the corresponding assertions. The check() method executes the check, and performs the necessary bookkeeping operations. This method is executed in the context of an assertion checking thread. The method begins by terminating the trace associated with the execution of attach(), and recording the call to attach() within the *caller's* trace (lines 6–7). Finally, it checks the post-condition of attach() (lines 8–13). The rst three clauses of the boolean expression check that the attaching observer (ob) was properly added to the subject's observer set (obs). The fourth clause checks that the subject state was *unmodified* — according to the appropriate de nition of *Modified*() (mod()). Finally, the fth and sixth clauses check that only one role method was invoked, and that the call was to the update() method of the attaching observer. This call is required to bring the attaching ob-

server into a state that is consistent with the current state of the subject. The checkCondition() method displays a suitable warning if the boolean expression passed as argument evaluates to false (line 13).

Note that assertion violations are reported at execution points beyond those at which the violations actually occur. Indeed, the temporal distance between occurrence and noti cation can be signi cant. To prevent this distance from compromising a designer's ability to localize defects, each assertion object collects *context* information corresponding to the point at which the assertion was constructed. This information is captured using the re ecti ve facilities of AspectJ, and includes the signature and line number of the class method to which the assertion is bound. This information is included in any violation messages generated by the object.

4 Evaluation

To evaluate the performance of our parallel monitoring approach, we developed a test system constructed using the Observer pattern. The system implements a primitive simulation consisting of three main classes: Time, Clock, and Person. The system constructs 50 time objects that serve as subjects; 10 clocks and 10 persons are attached to each. A clock simulates a digital clock, and maintains information about the current state of the display. A person simulates a student who is either *sleepy* or *ready*, depending on the current time. The system effects a range of changes on each time object, then detaches all person objects from their respective subjects, and then again effects time changes. Status information is displayed to the console throughout.

We implemented two monitors that perform identical checking operations based on the pattern contract underlying this design. The rst monitor uses a sequential strategy; the second uses a parallel strategy. The system execution time was measured under a range of scenarios. The following variables were considered: (***i***) number of system processors (1, 2), (***ii***) runtime heap availability (512M, 256M, 128M), (***iii***) checking thread priority (normal, low), and (***iv***) assertion complexity. This last element requires some explanation. Recall that each instance of the Observer pattern is specialized as appropriate to some application. A key element of this process involves the selection of suitable auxiliary concept de nitions. These de nitions will likely vary from application to application. Since the assertions to be checked are de ned in terms of these concepts, their definitions have a direct impact on monitoring performance. To evaluate this impact, we simulated increased de nitional complexity by inserting a *no-op* loop within the methods corresponding to *Modified*() and *Consistent*(). We simulated variations in complexity by varying the number of no-op iterations from 0 to 15000 in increments of 1000.

The measurements were performed using the Java

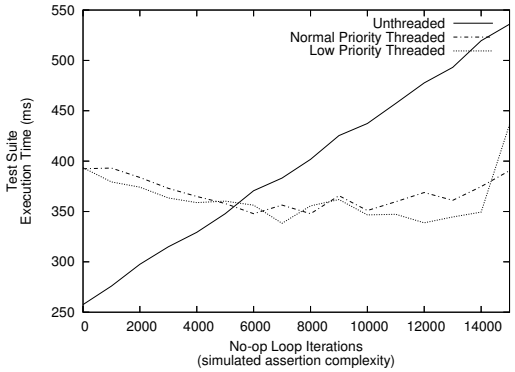


Figure 3: Performance (2 processors, 512M heap)

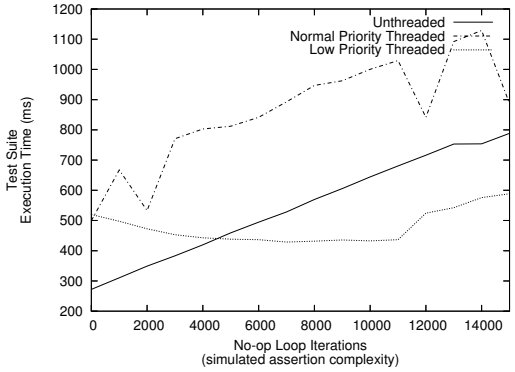


Figure 4: Performance (1 processor, 512M heap)

HotSpot VM (v1.5.0_06) on a workstation running Windows XP Pro, with dual 3.06Ghz Pentium 4 processors. The dual processor results are presented in Table 1; Table 2 presents the single processor results. The gures re ect an average computed over 50 runs of each con guration. The graphs shown in Figures 3–8 summarize this data.

Figures 3, 5, and 7 show that the parallel approach provides considerable performance improvements in the dual processor case for suf ciently complex assertions. This result was expected since the application thread and the assertion checking thread are *physically* concurrent. The spikes toward the right of each graph are due to long-running assertion checks that cause the assertion queue to become large, resulting in increased garbage collection activity. This factor is most signi cant for the low-priority case since the assertion queue grows more quickly than in the normal-priority case. (In the former case, the application thread *producing* assertions runs at a higher priority than the checking thread *consuming* them.) Figures 4 and 6 show that the parallel approach also provides improvements in the single processor case. However, this is only true if the assertion checks are performed in a low-priority thread to avoid contention with the application thread. Again, the increasing slopes indicate increased garbage collection activity. Figure 8 shows that the parallel approach performs poorly in the single processor case if heap availability is

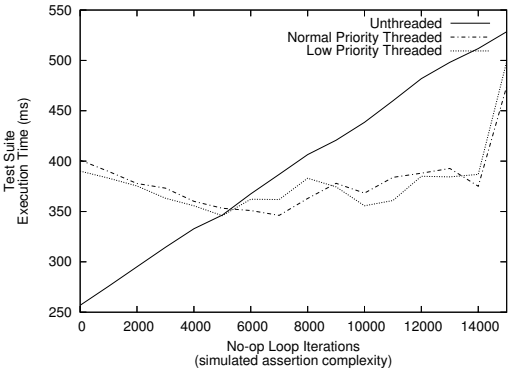


Figure 5: Performance (2 processor, 256M heap)

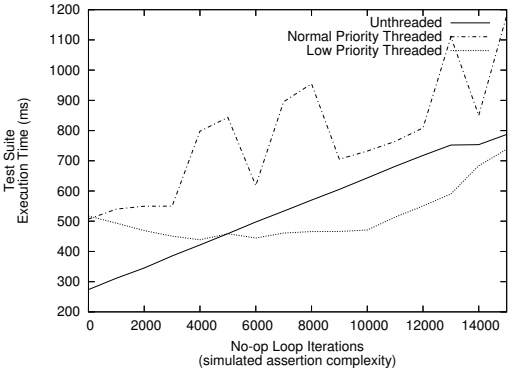


Figure 6: Performance (1 processor, 256M heap)

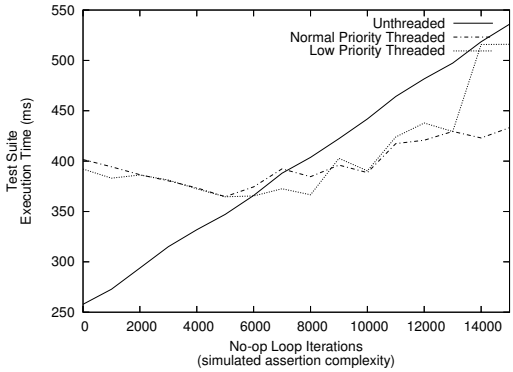


Figure 7: Performance (2 processor, 128M heap)

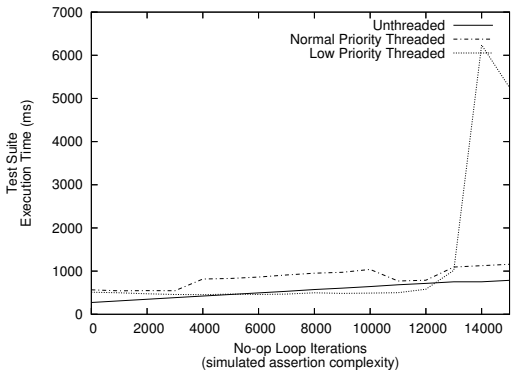


Figure 8: Performance (1 processor, 128M heap)

constrained. Although the results are not shown, if the assertions are sufficiently complex, the checking thread will fail, resulting in system termination.

5 Related Work

Our previous work focuses on specifying design pattern requirements using a contract formalism, and detecting violations of those contracts using a sequential monitoring approach [19, 22, 9]. Other authors have also considered techniques for improving the precision of design pattern descriptions. The approaches discussed in the literature span four categories. (***i***) UML extensions have been proposed that capture the ways in which classes map to their respective roles [3, 23]. (***ii***) Higher-order logic formalisms have been developed that capture the structural properties of patterns [4]. (***iii***) Action-system notations have been used to specify behavioral pattern properties [15]. Finally, (***iv***) hybrid approaches that combine elements of (*ii*) and (*iii*) have been discussed [21]. Our work is unique, however, in its focus on runtime monitoring and monitor parallelization.

Runtime assertion checking has a long history in the object-oriented community. The approaches reported in [14, 17, 1], for example, describe language extensions and compiler tools for annotating programs with assertions, and generating the appropriate monitoring code based on those assertions. Wrapper-based techniques have also been described [6], as well as the automated generation of such wrappers [5]. Some have considered aspect-based approaches, both in Java [13] and C++ [8]. We also mention that some have explored the use of aspect-oriented constructs in *implementing* design patterns [10, 11, 16]. We are the first, however, to investigate an approach to *monitoring* design pattern implementations to detect requirements violations, and this is the first manuscript to explore monitor parallelization and detailed performance evaluation.

6 Conclusion

The goal of our research program is to maximize the benefits of design patterns through the use of formal *pattern contracts*, and *contract monitors* that detect contract violations. In this paper, we presented an approach to parallelizing an existing contract monitoring system, and evaluated its performance relative to its sequential counterpart. We showed that the parallel approach provides significant performance improvements across a range of systems, reducing the performance impact associated with monitoring. The result is to decrease the time required to execute pattern-centric test suites, and to make the deployment of defensively designed reusable assets (with pattern-centric checks) more feasible.

One outcome of our study is a characterization of the point at which assertion complexity outweighs the overhead

Heap Space	*Threading*	*Assertion Complexity (10^3 No-op Loop Iterations)*															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
512M	None	258	276	298	315	329	348	371	383	402	425	437	457	478	493	520	536
	Normal	392	393	384	373	365	358	348	356	348	366	351	360	369	361	375	391
	Low	394	379	374	363	359	360	356	338	356	362	347	347	339	345	349	436
256M	None	257	276	295	315	333	346	367	387	406	421	439	460	482	498	512	528
	Normal	401	390	378	373	360	353	351	346	363	378	368	384	388	393	375	474
	Low	390	383	375	363	356	346	362	362	383	374	356	361	385	384	387	499
128M	None	258	273	294	315	332	347	366	388	404	422	442	464	482	497	519	536
	Normal	402	394	387	380	373	365	374	392	385	396	389	417	421	429	423	433
	Low	392	383	386	381	372	365	365	373	367	403	390	424	438	429	516	516

Table 1: Test Suite Execution Time for 2 Processors (in ms)

Heap Space	*Threading*	*Assertion Complexity (10^3 No-op Loop Iterations)*															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
512M	None	272	310	349	383	419	459	494	528	570	606	644	681	716	753	754	789
	Normal	499	667	533	771	803	812	841	893	947	962	1000	1029	842	1092	1129	887
	Low	520	498	472	453	443	438	437	429	432	435	433	436	524	542	576	589
256M	None	274	312	345	385	422	459	498	533	570	605	643	682	718	752	753	787
	Normal	507	540	550	550	798	844	618	895	955	705	732	764	809	1110	850	1181
	Low	517	494	469	451	438	458	444	461	466	466	471	514	551	590	684	738
128M	None	272	312	349	384	421	461	496	532	572	607	642	683	714	753	753	788
	Normal	567	542	547	546	816	830	863	912	952	972	1038	770	789	1093	1127	1158
	Low	512	497	474	460	455	467	459	468	498	484	491	500	580	1017	6240	5251

Table 2: Test Suite Execution Time for 1 Processor (in ms)

introduced through parallelization. Or stated another way, the results clarify the point at which a switch from a sequential monitoring strategy to a parallel strategy becomes advantageous. Our future work aims to incorporate these results as part of our monitor generation framework. By examining the complexity of the assertions associated with a particular pattern specialization, the monitor generator will be able to choose the implementation style that introduces the least impact on the monitored system.

Acknowledgements. The authors gratefully acknowledge the - nancial support of the South Carolina Space Grant Consortium, and thank the anonymous reviewers for their feedback on the rst draft of this manuscript.

References

[1] L. Burdy et al. An overview of JML tools and applications. *Software Tools for Technology Transfer*, 2005.

[2] F. Buschmann et al. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, 1996.

[3] J. Dong. UML extensions for design pattern compositions. *Journal of Object Technology*, 1(5):151–163, November–December 2002.

[4] A.H. Eden. *Precise Specification of Design Patterns and Tool Support in Their Application*. PhD thesis, Tel Aviv University, 2000.

[5] S.H. Edwards. A framework for practical, automated black-box testing of component-based software. *Software Testing, Verification, and Reliability*, 11(2):97–111, 2001.

[6] S.H. Edwards et al. A framework for detecting interface violations in component-based software. In *The 5^{th} International Conference on Software Reuse*, pages 46–55, June 1998.

[7] E. Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[8] T.H. Gibbs and B.A. Malloy. Weaving aspects into C++ applications for validation of temporal invariants. In *The 7^{th} European Conference on Software Maint. and Reeng.*, pages 249–258, March 2003.

[9] J.O. Hallstrom et al. Amplifying the bene ts of design patterns: From speci cation through implementation. In *Foundational Approaches to Software Engineering*, pages 214–229, March 2006.

[10] J. Hannemann and G. Kiczales. Design pattern implementation in Java and AspectJ. In *The 17^{th} ACM OOPSLA Conference*, pages 161–173, November 2002.

[11] R. Hirschfeld et al. Design patterns and aspects — modular designs with seamless run-time integration. In *The 3^{rd} German GI Workshop on Aspect-Oriented Software Development*, March 2003.

[12] G. Kiczales et al. An overview of AspectJ. In *The 15^{th} ECOOP*, pages 327–353, 2001.

[13] M. Lippert and C.V. Lopes. A study on exception detection and handling using aspect-oriented programming. In *The 22^{nd} International Conference on Software Engineering*, pages 418–427, June 2000.

[14] B. Meyer. *OO Software Construction*. Prentice Hall, 1988.

[15] T. Mikkonen. Formalizing design patterns. In *The 20^{th}th International Conference on Software Engineering*, pages 115–124, 1998.

[16] N. Noda and T. Kishi. Implementing design patterns using advanced separation of concerns. In *The OOPSLA Workshop on Advanced Separation of Concerns in Object-Oriented Systems*, October 2001.

[17] D.S. Rosenblum. A practical approach to programming with assertions. *IEEE Trans. on Software Engineering*, 21(1):19–31, 1995.

[18] D.C. Schmidt et al. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*. John Wiley & Sons, 2000.

[19] N. Soundarajan and J.O. Hallstrom. Responsibilities and rewards: Specifying design patterns. In *The 26^{th} International Conference on Software Engineering*, pages 666–675, May 2004.

[20] Sun Microsystems, Inc. Java 2 platform SE 5.0 API speci cation. http://java.sun.com/j2se/1.5.0/docs/api/index.html, 2004.

[21] T. Taibi and D.C.L. Ngo. Formal speci cation of design patterns – a balanced approach. *Journal of Object Tech.*, 2(4):127–140, 2003.

[22] B. Tyler et al. Automated generation of monitors for pattern contracts. In *The 21^{st} ACM SAC Conference*, April 2006. (*to appear*).

[23] J. Vlissides. Notation, notation, notation. *C++ Report*, April 1998.

An Empirical Study of the Maintenance Effort

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

Kai Chen
Institute for Software Integrated Systems,
Vanderbilt University
Nashville, TN, USA
kai.chen@vanderbilt.edu

Abstract

Software must be continually maintained to remain useful. One of the major concerns in the management of maintenance is to understand the maintenance effort in order to allocate the resources. This paper proposed a software maintenance effort model in which the maintenance effort is divided into the programmer effort and the supporting effort. The programmer effort contains the isolation, modification and testing effort, while the supporting effort includes the management and service effort. In addition, through an empirical study, we analyze the distributions of different maintenance efforts and the correlations among them.

1. Introduction

Planning is critical to software maintenance management. An effective planning is based on the accurate effort estimation of the maintenance task. Early and accurate maintenance effort estimation can assist cost estimation, support resource allocation, guide decision making, and reduce project risks [1] [2]. However, maintenance effort is affected by many factors such as the maintenance task and the system complexity. An accurate effort estimation model in turn is based on the fully understanding of the maintenance effort. According to our knowledge, in the related studies [3] [4] [5], the maintenance effort spent by programmers was considered as a whole. The researchers did not differentiate maintenance effort according to the difference of the maintenance activity. Furthermore, the management effort and the service effort are largely ignored.

In this paper, the maintenance effort is divided into the programmer effort and the supporting effort. The programmer effort contains the isolation, modification and testing effort, while the supporting effort includes the management and service effort. A software maintenance effort model is proposed based on this categorization. In addition, through the statistical analysis, we analyze the distributions of different maintenance efforts and their correlations.

Data used in this study was provided by NASA SEL. It includes over 22 thousand records of maintenance changes. Some records include complete information about the changes, such as the effort to isolate to the problem, the effort to modify the program, and the effort to test the modifications. Some records contain the data about the effort to manage the maintenance task and the effort to assist the maintenance task.

2. Maintenance effort

From the viewpoint of the purpose of maintenance, there are three types of maintenance tasks: corrective, perfective, and adaptive. Corrective maintenance (or software repair) consists of the removal of residual faults while leaving the specification unchanged. Perfective maintenance and adaptive maintenance are also called enhancement. Perfective maintenance improves the effectiveness of the product, such as security, performance, and usability. Adaptive maintenance contains changes made to the product in order to adapt to a new platform, new software or new hardware.

Maintenance effort includes the effort spent by all involved personnel in a maintenance task, which is either a corrective maintenance, a perfective maintenance, or an adaptive maintenance. The maintenance programmers are the key group of people involved in a maintenance task. We call the maintenance effort spent by the programmers the *programmer effort*. For each maintenance task, the

maintenance programmers need to work on three consecutive activities: isolating the problem, modifying the program, and testing the modification. Accordingly, the programmer effort can be partitioned into three parts: the effort to isolate problems, the effort to make modifications, and the effort to test the modifications.

The effort to isolate problems (*isolation effort*) is the effort spent in determining which modules should be changed. For corrective maintenance, it is the effort to trace the origination of reported errors. For perfective maintenance and adaptive maintenance, it is the effort to target the modules that are related to the enhancement requirements. The effort to make the modifications (*modification effort*) contains the effort spent in determining how the modules should be changed and the effort in making the changes. The effort to test the modifications (*testing effort*) is the effort spent in regression testing to ensure the modifications are correct and do not introduce regression faults to the system.

The maintenance effort spent by people other than the maintenance programmers is called the *supporting effort*. The supporting effort can be subdivided into the effort to manage the maintenance process (*management effort*) and the effort to provide additional service (*service effort*). The management effort is the effort spent by maintenance managers in one maintenance task. It includes the effort to authorize the changes, allocate the resources, and monitor the maintenance process. The service effort is the effort spent by other assisting personnel in one maintenance task. It includes the effort to document the maintenance, update the change log, and refresh the repository.

Figure 1 shows the proposed software maintenance effort model. The arrows indicate the work flow of the maintenance activities and the dependencies between them. In this model, the maintenance effort is divided into the programmer effort and the supporting effort. The programmer effort consists of three consecutive efforts: isolation, modification and testing effort. The supporting effort includes the management effort and service effort, which are involved in the whole maintenance process. This model applies to all three types of maintenance: corrective, perfective and adaptive maintenance.

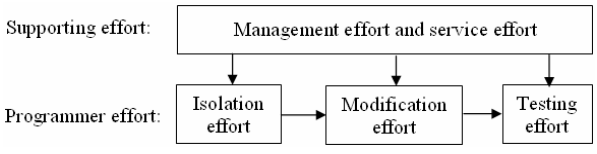


Figure 1. The software maintenance effort model

3. The empirical study

3.1. The distribution of programmer effort

In the data we studied, effort is represented as person-hours to perform a maintenance task. In all, we have 15164 maintenance records that contain the complete effort data. Figure 2 shows the distribution of different programmer efforts. The data is analyzed by summarizing each of the three programmer efforts (isolation effort, modification effort, and testing effort).

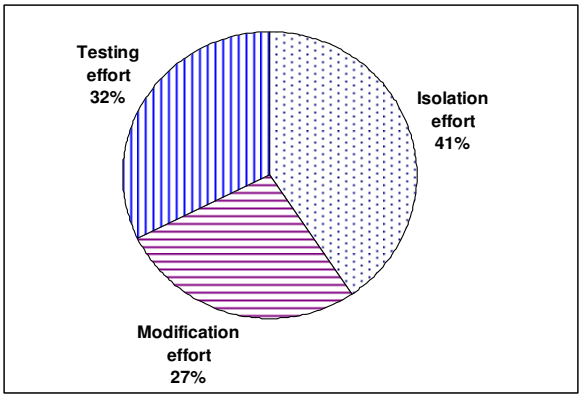


Figure 2. The distribution of programmer effort

To understand how different programmer efforts are distributed in different maintenance types, we divide the effort data into subgroups (adaptive maintenance, corrective maintenance, and perfective maintenance). Each group contains three types of programmer effort (isolation effort, modification effort, and testing effort). Figure 3 shows the average isolation effort, modification effort, and testing effort for different types of maintenance tasks.

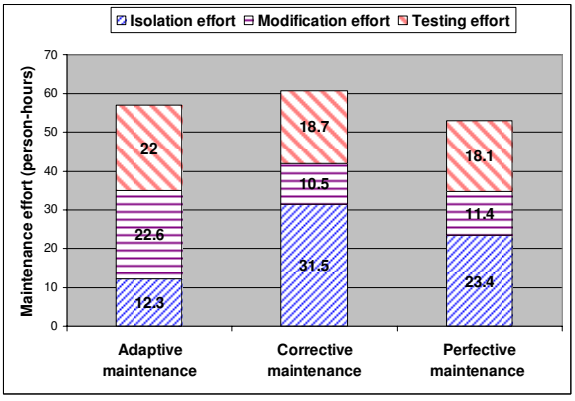


Figure 3. The average maintenance effort

To study the similarities or differences among three distributions statistically, we hypothesize that isolation effort, modification effort, and testing effort have similar distributions in three types of maintenance task:

H01: There is no significant difference between the distribution of the isolation effort, modification effort, and testing effort in adaptive maintenance, corrective maintenance, and perfective maintenance.

The obvious way to test this hypothesis is to apply the chi-square test. We construct a 3 × 3 contingency table based on the data shown in Figure 3. The degree of freedom of this test is 4 and the chi-square is 14.07. The corresponding p-value is 0.007. Therefore, we reject the null hypothesis and conclude that there are significant differences among the three distributions.

Previous related studies [6] [7] showed that different types of maintenance task have different affecting factors and should follow different estimation models [8]. Our study extends these results and shows that the distributions of the programmer effort are different for different types of maintenance tasks. More specifically, it needs more effort to isolate the problem in corrective maintenance than in adaptive maintenance and in perfective maintenance, because it is usually more difficult to trace the origination of the error. On the other hand, more effort is needed to make the modifications for adaptive maintenance than for corrective maintenance and perfective maintenance, because adapting a program to new hardware or new software requires modifications to all related modules.

3.2. The correlations among isolation effort, modification effort, and testing effort

If a system is more complex, more effort is needed to isolate the problem, to modify and test the program. Based on the interdependences between these efforts shown in Figure 1, we would expect to find the modification effort increases as the isolation effort increases, and the testing effort increases as the isolation effort and modification effort increase. In more detail, we tested the following three null hypotheses:

H02: There is no linear relationship between the modification effort and the isolation effort.

H03: There is no linear relationship between the testing effort and the isolation effort.

H04: There is no linear relationship between the testing effort and the modification effort.

We performed Spearman's rank correlation test. Table 1 shows the correlations among the maintenance efforts. Because all the correlation coefficients are significant at 0.01 level, we reject all three null hypotheses and conclude that there is significant positive linear correlation between modification effort and isolation effort; there is significant positive linear correlation between testing effort and isolation effort; there is significant positive linear correlation between testing effort and modification effort.

Table 1. The correlations among the maintenance efforts

	Isolation effort	Modification effort	Testing effort
Isolation effort	1.000	0.619*	0.781*
Modification effort	0.619*	1.000	0.458*
Testing effort	0.781*	0.458*	1.000

* Correlation is significant at the 0.01 level (2-tailed).

Number of data set: 15164

3.3. Management effort and service effort

A successful maintenance task not only depends on the programmers, but also depends on the supporters who provide management and services to the programmers. Figure 4 shows the average programmer effort and supporting effort in 430 recorded projects. It shows the management effort is about 14 percent of the programmer effort, and the service effort is about 9 percent of the programmer effort.

If more effort is spent by a programmer to perform the maintenance task, more management effort and service effort are expected to be needed. To study their relationships statically, we tested the following three null hypotheses:

H05: There is no linear relationship between the management effort and the programmer effort.

H06: There is no linear relationship between the service effort and the programmer effort.

H07: There is no linear relationship between the service effort and management effort.

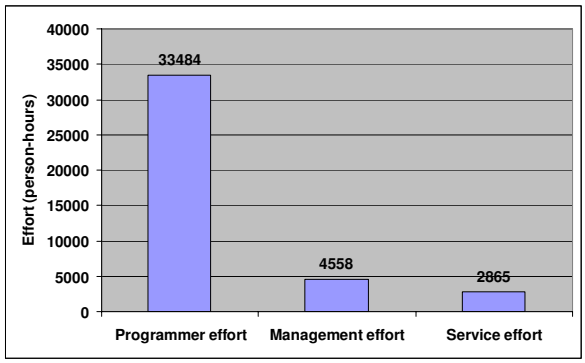


Figure 4. The average programmer effort and supporting effort in 430 projects

Table 2 shows the correlations among the programmer effort, the management effort, and the service effort. Because all the correlation coefficients are significant at 0.01 level, we reject all three null hypotheses and conclude that there is significant positive linear correlation between management effort and programmer effort; there is significant positive linear correlation between service effort and programmer effort; there is significant positive linear correlation between service effort and management effort.

Table 2. The correlations among programmer effort, management effort, and service effort

	Programmer effort	Management effort	Service effort
Programmer effort	1.000	0.847*	0.748*
Management effort	0.847*	1.000	0.681*
Service effort	0.748*	0.681*	1.000

* Correlation is significant at the 0.01 level (2-tailed).
Number of set of data: 430

4. Conclusions

In this paper, we performed a deep analysis on different efforts in the software maintenance process and proposed a software maintenance effort model. In addition, we performed an empirical study on NASA SEL projects and found (1) the distributions of isolation effort, modification effort, and testing effort are different in corrective, adaptive and perfective maintenance; (2) strong correlations exit among isolation effort, modification effort, and testing effort; and (3) strong correlations exist among programmer effort, management effort, and service effort.

These empirical results can help the maintenance managers to estimate the effort, allocate the resource, plan and monitor the project, and improve the maintenance process. On the other hand, this study provides the basis for constructing more accurate and refined maintenance effort models, in which, different models could be built for isolation effort, modification effort, testing effort, and supporting effort (management effort and service effort).

5. References

[1] A.D. Lucia, E. Pompella, and S. Stefanucci, "Effort Estimation for Corrective Software Maintenance", *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, Ischia, Italy, 2002, pp. 409–416.

[2] A.D. Lucia, E. Pompella, and S. Stefanucci, "Assessing Effort Estimation Models for Corrective Software Maintenance through Empirical Studies", *Information and Software Technology*, vol. 47, no. 1, 2005, pp. 3–15.

[3] V. Basili, L. Briand, S. Condon, Y. Kim, W. L., Melo, and J. D. Valett, "Understanding and Predicting the Process of Software Maintenance Releases", *Proceedings of the 18th International Conference on Software Engineering*, Berlin, Germany, 1996, pp. 464–474.

[4] J.H. Hayes, S.C. Patel, and L. Zhao, "A Metrics-Based Software Maintenance Effort Model", *Proceedings of the 8th European Conference on Software Maintenance and Reengineering*, Tampere, Finland, 2004, pp. 254–260.

[5] B. Boehm, E. Horowitz, R. Madachy, D. Reifer, B.K. Clark, B. Steece, A.W. Brown, S. Chulani, and C. Abts. *Software Cost Estimation with Cocomo II*, Prentice-Hall 2000.

[6] V. Basili, L. Briand, and S. Condon, "Understanding and Predicting the Process of Software Maintenance Release", *Proceedings of the 18th International Conference on Software Engineering*, Berlin, Germany, 1996, pp. 464–474.

[7] I. Vessey and R. Weber, "Some Factors Affecting Program Repair Maintenance: An Empirical Study", *Communications of the ACM*, vol. 26, no. 2, 1983, pp. 128–134.

[8] F. Fioravanti and P. Nesi, "Estimation and Prediction Metrics for Adaptive Maintenance Effort of Object-Oriented Systems", *IEEE Transactions on Software Engineering*, vol. 27, no. 12, 2001, pp. 1062–1084.

Experimental Study on the Impact of Team Climate on Software Quality

Silvia T. Acuña
Escuela Politécnica Superior
Universidad Autónoma de Madrid
silvia.acunna@uam.es

Marta Gómez
Escuela Politécnica Superior
Universidad San Pablo-CEU
mgomez.eps@ceu.es

Ramón Rico
Facultad de Psicología
Universidad Autónoma de Madrid
ramon.rico@uam.es

ABSTRACT
This paper describes an empirical study that examines the work climate amongst members of software development teams. The question of interest is whether the fit between team climate preferences and perceptions in software developer teams has any effect on software product quality. The team climate factors examined were West and Anderson's participative safety, support for innovation, team vision and task orientation. The results show that high team vision preferences and high post-development perceptions of participative safety improve the quality of the software. Additionally, it is found that better climate expectations within software development teams for the participative safety and team vision factors have a positive effect on software product quality. This finding should encourage further investigation of what effect better climate expectations within software engineering teams for the support for innovation and task orientation factors has on software product quality.
Keywords: Work climate preferences; Work climate perceptions; Work climate preferences-perceptions fit; Team climate; Team building

1. INTRODUCTION

This article is about teams and the collective effort of individuals working together towards a common goal. A team is defined as "a number of individuals brought together for a certain task, goal or objective, engaged in frequent face-to-face interaction to execute a task, while the individuals are (mutually) interdependent on each other with regard to the outcome of the task and its execution" (Katzenbach & Smith, 2001). More specifically, we are interested in teams that develop software.

Social psychology is now researching team building considering a series of factors that have been found to affect team performance. This research has traditionally analysed team member personality factors and their relationship to task characteristics (Molleman et al., 2004; Barry & Stewart, 1997). However, this relationship involves other factors of team behaviour like interactions between people, including conflict, cohesion, cooperation, communication and climate. It is accepted that team performance cannot be reliably predicted from team personality composition and task characteristics alone, but rather depends on the interactive effects of team climate characteristics (Burch & Anderson, 2004). Teamwork is affected by what preferences and perception the team members have as to how the tasks should be performed within the team, that is, of what the climate is like. Climate is understood as the shared perceptions of working practices. Work climate perception is related to organisational performance (Fay et al., 2004).

There are a few studies in the field of software development on the impact of some team factors —cohesion, conflict, team structure, expertise coordination and administrative coordination— on team performance (Yang & Tang, 2004; Faraj & Sproull, 2000). Climate is one of the least studied group factors in software development. Nevertheless, the consideration of work climate is a catalyst for other team factors and encourages team development (Burch & Anderson, 2004). The work reported in this paper examines the key question of whether team effectiveness depends on the team members' ability to cope with and work in a particular type of climate.

According to West and Anderson (1996), four factors have been identified as being central in determining effective team functioning. These four factors are: participative safety, support for innovation, team vision and task orientation. This article analyses these four work climate factors within 35 software developer teams. Additionally, it presents an experimental study that measures the team climate preferences, the team members' perceptions of climate and the software development climate preferences and perception fit. This study aims to check empirically two key questions for each of the four work climate factors within software development teams: a) whether there is any dependency between either the preferences or the perceptions teamwork and software product quality; and b) whether there is any dependency between the climate preferences and perception fit and the quality of the software produced by the surveyed teams.

This paper is structured as followed. Section 2 describes related work on team building in the software process. Section 3 presents the teamwork climate approach to team building from the field of social psychology and describes the examined teamwork climate factors. Section 4 details the approach and design of the experimental study carried out to relate climate to software product quality. In section 5, we present the analysis of the results. Section 6 discusses the results and states the conclusions of this research.

2. TEAM BEHAVIOUR RELATED WORK

There has been little research into group aspects applied to software development. On the one hand, there are a number of papers that deal with what personal characteristics of the participants influence development success rather than with

the "team" issue (Teague, 1998; Hardiman, 1997). On the other hand, there are a number of papers that deal with personality aspects as related to the team. For example, there are studies that use a standard test, like the Myers-Briggs Type Indicator (Rutherford, 2001; Bostrom & Kaiser, 1981), to determine the guidelines for team success according to software engineer personality types. There is another study that determines the connection between abilities and personality traits and team performance (White & Leifer, 1986).

Additionally, there are a number of papers that focus on team forming. There is only one quantitative team forming method based on a quantitative abilities model (Burdett & Li, 1995). This work does not consider the softer aspects, such as team member personality factors, actual team personality and team climate preferences. Another team forming method is based on analysing required and available skills (Zakarian & Kusiak, 1999). Neither of these two studies considers group factors, such as confidence, conflict and climate.

There is one study that combines individual personality and group factors, but omits climate characteristics. In this paper, Zuser and Grechening (2003) propose the use of a questionnaire based on abilities and personality traits that provides 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 (Tuckman, 1965).

We find then that none of the analysed studies take climate into account, despite this being a factor that is involved in effective and efficient team performance.

Social psychology research has suggested that climate preferences and perceptions are associated with a variety of important outcomes at the individual, group, and organizational levels. These include job satisfaction (Mathieu et al., 1993), individual job performance (Brown & Leigh, 1996), and team performance (Patterson et al., 2004).

Despite its importance, little research has been done on the person-team-climate fit in software development.

3. MEASURING TEAM CLIMATE DIMENSIONS

Team climate is a complex concept and is decomposed into dimensions. There is a range of standard tests for studying team climate dimensions. For example, OCQ (Organizational Climate Questionnaire) (Litwin & Stringer, 1968) comprises 50 items that assess nine climate factors. Schneider's service climate (Schneider et al., 1998) is a climate instrument specific to the services marketing domain. Another commonly used test is the Team Selection Inventory (TSI) (Anderson & Burch, 2003) and the Team Climate Inventory (TCI) (Anderson & West, 1998; Anderson & West, 1999). These last two tests are psychometric inventories for measuring team climate preferences and team climate perceptions, respectively. These tests have been validated, and have been shown to have robust properties, with acceptable levels of reliability and validity (Anderson & West, 1998; Burch & Anderson, 2004). These measures of team climate in organizations are applied in settings such as organizational climate surveys, team building and development, selection of new members for groups, and group development over time (Anderson & West, 1998; Patterson et al., 2004).

Both the TSI and TCI questionnaires are composed of 44 items divided into the four factors considered essential for effective team operation and propensity to innovation:

- Participative safety (11 items): how much trust participating team members sense there is within the group when explaining their opinions and ideas.
- Support for innovation (8 items): support provided by the team for innovative ideas.
- Team vision (11 items): how clearly the team defines goals.
- Task orientation (6 items): how much effort the team puts into achieving excellence in what it does.

These questionnaires include a social desirability index, added to detect response bias and faking (8 items).

All the factors and indexes are measured on a 5-point Likert scale. Respondents are asked to indicate the extent to which they agree and disagree with each item. All respondents are told that there are no right or wrong answers and that they should take care to answer accurately and sincerely. Responses range from 1 "not at all" or "strongly disagree" to 5 "completely" or "strongly agree". A higher TSI and TCI score indicates a greater preference or perception of participative safety, support for innovation, team vision and task orientation by team members.

4. CONCEPTION AND DESIGN OF THE EXPERIMENTAL STUDY

The empirical study run fits the description of a quasi-experiment (Juristo & Moreno, 2002). 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. In other words, the group applies a given treatment, but this treatment is not allocated at random. A quasi-experiment is characterised by being not very intrusive and not very costly. The selected statistical tests are applied to the data collected.

4.1. Participants

The participants were 2nd-year computing students at Madrid Autonomous University's Higher Technical School of Computing during the 2004/2005 academic year. These students were taking the Data Structures and Algorithms subject. A total of 103 subjects participated in this empirical study, of which 81 were male (79%) and 22 were female (21%). Of the students, 75% (77 students) were aged under 21 years and 25% (26 students) were from 21 to 30 years of age.

The group of students was divided into 35 small (3-member) teams working on the development of the same software project. The Extreme Programming (XP) agile methodology (Beck et al., 2001) and the C programming language were used in this project. The teams were formed at random and their members were blind to the quasi-experiment conditions and hypothesis.

4.2. Response Variable

The response variable for this study is software product quality. The quality of the software product was measured by analysing the code and project documents. The weighted grade on a scale from 0 to 10 points was calculated according to the following formula:

Grade = (((Modularization * 2 + Testability * 2 + Functionality * 2 + Reusability * 2 + Style * 2) / 4) * 0.8) + ((Participation * 10 / 4) * 0.2)).

4.3. Data Collection

Information for this study was gathered by means of measurements taken before, during and after project development. *Before* refers to when the course kicked off, when the teams were being set up to do the development project, but no teamwork had yet been done. This was when the TSI team climate preferences test was handed out and collected in. *During* is the period that it takes to complete 45% of the development project (week 11) when the teams were already working on project development. This was when the climate perception questionnaires were handed out and collected in: TCI measured during the project. Finally, *after* is when the project is 95% complete (week 23) just before project development comes to an end. This period coincided with the end of the course and was the last stage after having handed in the project developed during the course. The measurement taken at this point was again climate perception: TCI measured after the project.

5. DATA ANALYSIS AND RESULTS

The following statistical techniques were used to analyse data and get the results:

1. Descriptive analysis of the four teamwork climate factors. This analysis is needed to explore the data and ascertain what characteristics the teams have.
2. Regressions between the pre-project preference and post-project perception and quality of the software product for each of the four climate factors. These analyses will be used to check if these teamwork climate factors lead to a better quality software product.
3. Analysis of mean differences between teamwork climate preferences and perceptions (which can be used to categorize the group as a team with a better, matching, worse climate) to check the dependency between the climate categories and the software product quality for each of the four climate factors.

The results for each analysis are detailed in the following.

5.1. Descriptive Analysis of Climate Factors

As mentioned, we analysed each of the teamwork climate factors both for climate preferences at the start of development and climate perceptions during and after the software system development. Five score categories (P1-P5) for each factor were determined considering the number of questions for each factor and the response values in the range from 1 to 5. The results for each climate factor are discussed below.

5.1.1. Participative Safety

Category P1 is up to 11 points and corresponds to participative safety "strongly disagree". Category P2 ranges from 12 to 22 "disagree". Category P3 ranges from 23 to 33 "undecided". Category P4 ranges from 34 to 44 "agree" and category P5 ranges from 45 to 55 "strongly agree."

Figure 1 shows just how little the overall mean of the teams falls at the different measurement times for the participative safety factor. Therefore, we can conclude that all teams agree or strongly agree about it being better to work in a safe climate, where this safety encourages all team members to be more participative. Additionally, all teams developed a safe climate, which, members being students, is to be expected.

5.1.2. Support for Innovation

Category P1 is up to 8 points and corresponds to "strongly disagree" about support for innovation. Category P2 ranges from 9 to 16 and signifies "disagree". Category P3 ranges from 17 to 24 and means "undecided". Category P4 ranges from 25 to 32, indicating "agree" and category P5 ranges from 33 to 40, that is, "strongly agree."

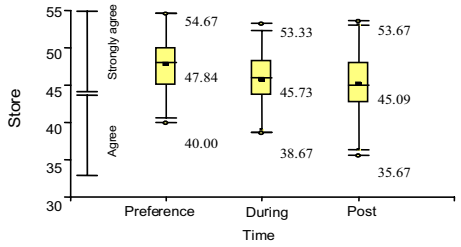


Figure 1. Teams classed by measurement time for the participative safety factor

Looking at the evolution of the overall mean in Figure 2, we can conclude that all the teams agree or strongly agree that it is better to work in a climate in which it is practicable to put forward new ideas, which encourages more innovation by all team members. Nevertheless, all the teams developed a climate that was not fully supportive of innovation. This has a logical explanation in that, being students, the primary aim of all the team members was to comply with minimum requirements, for which they selected the more conservative ideas. Then, if they had time to play with, they included more innovative ideas.

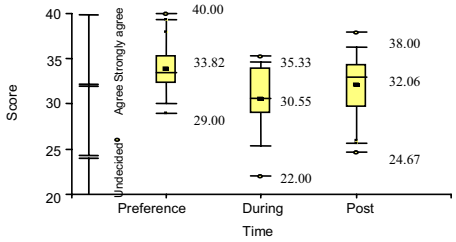


Figure 2. Teams classed by measurement time for the support for innovation factor

5.1.3. Team Vision

Category P1 is up to 11 points and corresponds to "strongly disagree" with team vision. Category P2 ranges from 12 to 22, signifying "disagree". Category P3 ranges from 23 to 33, meaning "undecided". Category P4 ranges from 34 to 44, that is, "agree" and category P5 ranges from 45 to 55, namely, "strongly agree."

Figure 3 shows that there is a slight drop in the overall mean by measurement time during and after the project as compared with before the project. Therefore, we can conclude that all the teams agree or strongly agree to it being better to work in a cohesive climate, where the goals are clearly defined and shared and teamwork is the preferred option for achieving the desired results.

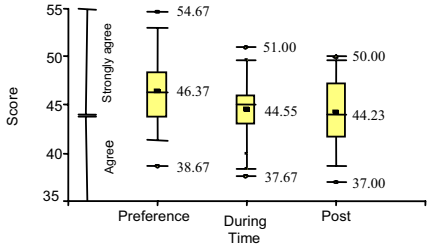


Figure 3. Teams classed by measurement time for the team vision factor

5.1.4. Task Orientation

Category P1 is up to 6 points and corresponds to a task orientation of "strongly disagree". Category P2 ranges from 7 to 12, signifies "disagree". Category P3 ranges from 13 to 18 and means "undecided". Category P4 ranges from 19 to 24, that is, "agree" and category P5 ranges from 25 to 30, namely "strongly agree."

Figure 4 shows that there are hardly any significant variations in the overall mean by time. Therefore, we can conclude that all the teams agree or strongly agree that it is better to work in a task-oriented climate, where such task orientation encourages all the team members to strive for excellence in what they are doing. Similarly, all the teams developed a task-oriented climate. This occurred in a group of students where emphasis was placed on development practices focusing on testing, refactoring, continuous integration and evaluations and checks of how the work was done to achieve excellence.

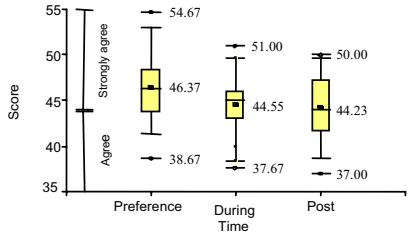


Figure 4. Teams classified by measurement time for task orientation factor

The Tukey test was used to calculate the differences between the mean scores of the factors at each of the times —before, during and after. The difference between the means should be greater than 1.56094 in the event of combinations of factors with times (significance level 0.05; error 3.7876, degrees of freedom (df) 204). We found that there are no statistically significant differences between measurements taken during and after development for all four factors. Consequently, one of these two measurements can be omitted.

To continue with the analysis of the results of this experimental study, we will retain the measurements of preferences before and perceptions after project development in the following tests.

5.2. Influence of Team Climate Preferences and Perceptions on Software Product Quality

Regressions for each of the four climate factors have been run between the teamwork climate preference before development and software product quality, as well as between the perception at the end of development and software product quality. In this technique, r^2_{xy} is the coefficient of determination or proportion of variance shared between x and y. This is a standardised measure whose values range from 0 and 1 and is used to interpret how much dependency there is between climate and product quality. The only statistically significant results are:

a) The preference for team vision before the project explains 13% of the quality of the software produced. For the team vision factor, the linear regression between software quality (dependent variable) and team vision preference before development (independent variable) is $r^2 = 0.134$, $F = 5.125$ ($p = .030$). This ratio is statistically significant at a confidence level of 95%.
b) The perception of participative safety after the project explains about 12% of quality of the software produced. For the participative safety factor, linear regression between software quality (dependent variable) and final perception of participative safety (independent variable) is $r^2 = .115$, $F = 4.295$ ($p=.046$). This ratio is statistically significant at a confidence level of 95%.

From the results, it can be said that most preferences for and perceptions of the teamwork climate factors before and after the project are not good at predicting software product quality. Only the preference for team vision and the perception of participative safety at the end of the project

(teams that have developed a safe climate, which has promoted the participation of each of the team members) can explain software product quality. Nevertheless, these relations do confirm important aspects of work climate.

5.3. Dependency Test between Teamwork Climate Category and Software Product Quality

Studying the fit between the teamwork climate preferences and perceptions involves a set of steps that is detailed below. First, we calculated the differences between the scores at the before and after times for all four factors. For each factor, we calculated the 34th and 67th percentiles to separate the data into three parts containing an identical number of teams, that is, the 34% that has the lowest difference values; the 33% that has the medium values and the 33% that has the highest values, each of these groups being called Better, Matching and Worse climate. These groups are different for each factor, that is, depending on each factor, there are different teams and number of teams whose perceived work climate is better than, matches or is worse than their preferences. Table 1 shows the statistics for each factor according to which they were separated into the three above-mentioned climate categories: better, matching and worse for the before and after times.

Secondly, we have categorized the resulting software product quality level, that is, the grades of the different teams. These grades range from 1.75 to 9.7 out of 10. The categories of grades considered were: C1 up to 6.99 points, C2 from 7.00 to 7.99, C3 from 8.00 to 8.99 and C4 from 9.00 to 9.99. Finally, Pearson's and the MV-G2 chi-squared tests were used to test the association between these grade categories and the category of differences, that is, the climate (better, matching, worse).

Table 2 shows the results for each factor, considering the climate categories for before and after the project.

We find that for the participative safety and team vision factors (Table 2), there is a dependency between the better, matching and worse categorisations and grades. In other words, for these two factors, participative safety and team vision, the teams with a better climate produced a better quality software product as compared with the matching and worse climates. In Table 2, for example, we find that 8 teams (of a total of 12 teams) whose climate preferences were improved were graded from 8.00 and 8.99 (C3), whereas only 5 teams (of a total of 12) for a matching climate attained a C3 grade and only one team (of a total of 11 teams) with a worse climate attained a C3 grade for the participative safety factor. A similar thing applies for the team vision factor (Table 2).

In other words, once the team is consolidated, teams in which the participative safety and team vision climate is better than team members' preferences have produced a better quality software product as compared with teams with a matching and worse climate.

We have concluded from these results that if the climate offers participative safety, the team is more likely to communicate and be more cohesive (Burch & Anderson, 2004). Similarly, cooperation and team cohesion is also encouraged if its members have a team vision. Therefore, if the real climate improves upon preferences in terms of team vision and participative safety, the team will certainly work better, as all the members will feel secure. On the basis of the results, we can say that software product quality will improve if the people working together have a better work climate perception than their preferences for both the participative safety and team vision factors.

Table 1. Statistics of the differences between the before and after scores and climate categories defined for each factor

Before/After Score Differences	n	Mean	Minimum	Maximum	P(34)	P(67)	Climate Category
Participative safety	35	2.11	-6.00	14.67	0.00	3.33	Better: up to 0 // Matching: from 0.01 to 3.33 // Worse: >3.333
Support for innovation	35	3.271	-4.33	11.50	1.66	3.67	Better: up to 1.67 // Matching: from 1.68 to 3.67 // Worse: >3.67
Team vision	35	1.824	-4.33	9.00	-0.33	4.00	Better: up to -0.33 // Matching: from -0.34 to 4.00 // Worse: >4.00
Task orientation	35	0.510	-5.00	6.67	-1.00	1.33	Better: up to -1.00 // Matching: from -1.01 to 1.33 // Worse: >1.33

Table 2. Contingency table by climate categories defined for each factor (before/after)

	Participative Safety Factor					Support for Innovation Factor					Team Vision Factor					Task Orientation Factor				
Grades categ. / Climate categ.	C1	C2	C3	C4	Tot.	C1	C2	C3	C4	Tot.	C1	C2	C3	C4	Tot.	C1	C2	C3	C4	Tot.
Better	1	2	8	1	**12**	4	3	5	1	**13**	1	2	8	1	**12**	3	4	4	1	**12**
Fitted	2	3	5	2	**12**	4	1	4	2	**11**	3	3	5	2	**13**	3	1	4	1	**9**
Worse	9	1	1	0	**11**	4	2	5	0	**11**	8	1	1	0	**10**	6	1	6	1	**14**
Total	**12**	**6**	**14**	**3**	**35**	**12**	**6**	**14**	**3**	**35**	**12**	**6**	**14**	**3**	**35**	**12**	**6**	**14**	**3**	**35**

	Participative Safety Factor			Support for Innovation Factor			Team Vision Factor			Task Orientation Factor		
Statistic	**Value**	**df**	**p**	**Value**	**df**	**p**	**Value**	**df**	**p**	**Value**	**df**	**p**
PearsonChi Square	18.17	6	0.0058	3.03	6	0.8050	15.45	6	0.0170	3.75	6	0.7098
MV-G2 Chi Square	19.05	6	0.0041	3.74	6	0.7117	16.24	6	0.0125	3.63	6	0.7272
Cramer's Conting. Coeff.	0.42			0.17			0.38			0.19		
Pearson's.Conting. Coeff.	0.58			0.28			0.55			0.31		

6. DISCUSSION AND CONCLUSIONS

At the start of development, the TSI is used to question participants about their work climate preferences. The teams in our study were very homogeneous and most of their members preferred teamwork, as, according to the test, they agreed or strongly agreed that it was preferable to work in teams with participative safety, support for innovation, team vision and task orientation. After they had been working together for a time, they were asked about their perceptions of teamwork in their respective teams. It was found that there was a transition between the higher two categories of work climate perceptions within the teams. Therefore, climate was generally favourable for the developed tasks. Although the real climate generally appears to have been slightly worse than preferences. That is, the scores remained high during and at the end of

development, although there was a drop in the scores for participative safety, support for innovation and team vision, especially halfway through development. One possible explanation is that at that point of the project, the teams were in the midst of the development process and were, therefore, still getting acquainted with each other and the tasks that they had been set. Additionally, the highest perceived climate scores are for the participative safety factor, which characterises the teams in which decision making is motivated and strengthened, provided that it occurs in an environment that is perceived to be non-threatening. As the adaptation within groups is so good, it is logical to think that this will have an impact on software product quality. And precisely these teams have shown themselves to product higher quality software products.

From the descriptive analysis of the four factors we found that when the teams were surveyed in the midst of the project (during), participative safety, team vision, support for innovation and task orientation dropped. When they were surveyed at the end of collaboration (after), support for innovation and task orientation recovered. Our hypothesis is that when the teams were surveyed in the midst of the project (during), they felt under too much pressure because they had to do the work and meet the deadlines. At that time, they were still putting together and oiling the group processes and teamwork. Then when they were surveyed at the end of collaboration (after), the teams were working well together and they were clearer about the what work had to be done and how to do it within the deadlines, and support for innovation and task orientation was found to increase, even though it was not significant with respect to the original score.

The results have shown that high team vision preferences and post development participative safety perceptions improve the quality of the software product. Additionally, we have found that better software development climate expectations for participative safety and team vision have a positive effect on software product quality. This point has need of further research on the effect of better software engineering team climate expectations for support innovation and task orientation on software product quality.

REFERENCES

Anderson N, Burch GJ. *The Team Selection Inventory*. ASE, NFER-Nelson, Slough, 2003.

Anderson N, West M. Measuring climate for work group innovation: Development and validation of the team climate inventory. *Journal of Organizational Behaviour* 1998; 19:235-258.

Anderson N, West M. *The Team Climate Inventory: User's Guide*. 2nd ed. ASE, NFER-Nelson, Windsor, 1999.

Barry B, Stewart GL. Composition, process and performance in self-managed groups: The role of personality. *Journal of Applied Psychology* 1997; 82:62-78.

Beck K., Beedle M, Cockburn A, Cunnimgham W, Fowler M et al., *Agile Manifesto*, http://agilemanifesto.org/, 2001.

Bostrom RP, Kaiser KM. Personality differences within systems project teams: Implications for designing solving centers. *Proceedings of the 18th Annual ACM SIGCPR Conference* 1981; 248–285.

Brown S, Leigh TW. A new look at psychological climate and its relationship to job involvement, effort, and performance. *Journal of Applied Psychology* 1996; 81:358-368.

Burch GJ, Anderson N. Measuring person-team fit: Development and validation of the team selection inventory. *Journal of Managerial Psychology* 2004; 19(4):406-426.

Burdett G, Li R-Y. A quantitative approach to the formation of workgroups. *Proceedings of the 32nd Annual ACM SIGCPR Conference* 1995; 202–212.

Faraj S, Sproull L. Coordinating expertise in software development teams. *Management Science* 2000; 46(12):1554-1568.

Fay D, Lührmann H, Kohl C. Proactive climate in a post-reorganization setting: When staff compensate managers' weakness. *European Journal of Work and Organizational Psychology* 2004; 13(2):241–267.

Hardiman LT. Personality types and software engineers. *IEEE Computer* 1997; 301(10):10–10.

Juristo N, Moreno AM. *Basics of Software Engineering Experimentation*. Kluwer, Boston, MA, 2002.

Katzenbach J, Smith D. *The Discipline of Teams: A Mindbook-Workbook for Delivering Small Group Performance.* John Wiley & Sons, 2001.

Litwin GH, Stringer RA. *Motivation and Organizational Climate*. Harvard University Press, Boston, MA, 1968.

Mathieu JE, Hoffman DA, Farr JL. Job perceptions-job satisfaction relations: An empirical comparison of three competing theories. *Organizational Behaviour and Human Decision Processes* 1993; 56:370-387.

Molleman E, Nauta A, Jehn KA. Person-job fit applied to teamwork: A multi-level approach. *Small Group Research* 2004; 35(5):515-539.

Patterson MG, Warr PB, West MA. Organizational climate and company performance: The role of employee affect and employee level. *Journal of Occupational and Organizational Psychology* 2004; 77:193-216.

Rutherfoord RH. Using personality inventories to help form teams for software engineering class projects. *SIGCSE-Bulletin* 2001; 33(3):76–76.

Schneider B, White SS, Paul MC. Linking service climate and customer perceptions of service quality: Tests of a causal model. *Journal of Applied Psychology* 1998; 83:150-163.

Teague J. Personality type, career preference and implications for computer science recruitment and teaching. *Proceedings of the Third Australasian Conference on Computer Science Education* 1998; 155–63.

Tuckman B. Developmental sequence in small groups. *Psychological Bulletin* 1965; 63:384–399.

West MA, Anderson N. Innovation in top management teams. *Journal of Applied Psychology* 1996; 81:680-693.

White K, Leifer R. Information systems development success: Perspectives from project team participants. *MIS Quaterly* 1986; 10(3):215–23.

Yang H-L, Tang J-H. Team structure and team performance in IS development: A social network perspective. *Information & Management* 2004; 41:335-349.

Zakarian A, Kusiak A. forming teams: An analytical approach. *IIE Transactions on Design and Manufacturing* 1999; 31(1):85-97.

Zuser W, Grechening T. 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.

Web Object Cacheability – How Much Do We Know?

[1]Chi-Hung Chi, [2]Jun-Li Yuan, [1]Liu Lin
[1]School of Software, Tsinghua University, China
[2]Institute for Infocomm Research, Singapore
Contact email: chichihung@mail.tsinghua.edu.cn

ABSTRACT -- In this paper, we would like to quantify the cacheability of web objects from the viewpoint of a proxy cache. Reasons behind the non-cacheability decision of objects are investigated and their inter-relationship is studied. Our experimental result shows that non-cachability of web object is mainly due to either the default (inappropriate) setup of web servers or explicit content providers' decision rather than their actual content change. Through this study, we hope to give content providers insight on where proxy caching and bandwidth utilization lose their performance and how they can optimize the delivery service of their content.

1. INTRODUCTION

In web caching, the cacheability of objects, which is defined as the availability and duration that web objects can be kept in a web cache, is controlled by some factors (mainly HTTP headers) which come along in the responses of objects from web servers. To decide whether an object is cacheable or not, web caches typically examine certain factors in a pre-defined order and make the decision based on the first satisfied factor. In reality, the response of an object often contains multiple factors. Therefore, to simply improve one factor may not result in improvement in cacheability because web caches might then meet other non-cacheable factors which also appear in the response and make decision based on them.

In this paper, we would like to investigate the cacheability of web objects and identify reasons behind their non-cacheablilty. We found that very often, web objects are labeled as non-cacheable not because of their content change. Instead, it is due to either the default (inappropriate) setup of web servers or explicit content providers' decision. With a better understanding on the web object cacheability decision, we hope that the latency problem of content and service delivery on Internet can be improved.

2. ALGORITHM AND FACTORS FOR CACHEABILITY DECISION

In general, whether a web object is cacheable is determined by the presence or absence of certain HTTP headers and the different status codes of HTTP responses. Table 1 lists the main HTTP response headers that are related to caching. These headers are selected based on the specification of HTTP/1.0 and HTTP/1.1. The HTTP response status codes can be classified into 4 classes for deciding cacheability of objects. Table 2 gives these classified status codes of HTTP responses.

Table 1: HTTP Headers that Related to Cacheability of Web Objects

Header Name	Usage
`Age`	Specify the age of response entity since the time the response was generated by the origin server
`Authorization`	Pass user's authentication credentials to origin server
`Cache-Control`	Control various aspects of caching
`Content-Length`	Specify length of entity object in bytes
`Content-Type`	Specify media type of the object
`Date`	Indicate date and time at which the message was generated
`Expires`	Specify expiration date and time of object
`Last-Modified`	Specify creation or last modification time of object on origin server
`Pragma`	This header is being phased out in favor of the Cache-Control header
`Vary`	Lists request headers on which document content may vary

If the response of an object contains status code belonging to Class 4, it is deemed as non-cacheable. If the response's status code belongs to Class 3, the object can be negatively cached for some time. For this type of objects, the responses from servers are actually error messages. If a web cache receives new request for such objects in the near future, it will assume the same error will happen and it will simply reply the request with the negatively cached object. For objects with status code belonging to Class 1 and 2, they are possibly cacheable. The final decision whether it is cacheable or not is further determined by if it can be validated. Web cache will not cache an object which cannot be validated at a later time. Whether an object can be validated is also determined by some HTTP headers such as "`Expires`", "`Last-Modified`" and "`Content-Length`" etc. If such headers are not present or their values are inappropriate, the objects will be considered as non-cacheable. (Note here that the "`Content-Length`" header is used in a reverse manner. If the value of this

header is zero, the object will be considered as non-cacheable because there is no use to cache a zero-byte object.)

Table 2: Classified Status Codes of Response

Class	Status Codes
Class 1	200(OK) 203(Non-Authoritative Information) 300(Multiple Choices) 301(Moved Permanently) 410(Gone)
Class 2	302(Moved Temporarily)
Class 3	204(No Content) 305(Use Proxy) 400(Bad Request) 403(Forbidden) 404(Not Found) 405(Method Not Allowed) 414(Request-URI Too Long) 500(Internal Server Error) 501(Not Implemented) 502(Bad Gateway) 503(Service Unavailable) 504(Gateway Timeout)
Class 4	206(Partial Content) 303(See Other) 304(Not Modified) 401(Unauthorized) 407(Proxy Authentication Required) Other codes and Invalid codes

Table 3: Factors for Non-Cacheability Decision of Web Objects

Factors	Description
fn1	There exists the header "Cache-Control: private"
fn2	There exists the header "Cache-Control: no-cache"
fn3	There exists the header "Cache-Control: no-store"
fn4	There exists the header "Vary"
fn5	There exists the header "Pragma: no-cache"
fn6	There exists the header "Content-Type: multipart/x-mixed-replace"
fn7	Status code belongs to Class 4
fn8	Status code belongs to Class 1, and server specified "must-revalidate"
fn9	Status code belongs to Class 1, and the object cannot be revalidated because there is no "Last-Modified" header
fn10	Status code belongs to Class 1, and the object cannot be revalidated because "Content-Length" is 0
fn11	Status code belongs to Class 1 and the object is fresh in 60 seconds or it can be revalidated, but there are no "Date", "Last-Modified" and "Expires" headers
fn12	Status code belongs to Class 2, and there is no "Expires" header

We systematize/standardize the conditions for determining cacheability of objects into two sets of factors. The first set of factors contains 12 factors which will rule that an object is non-cacheable. The second set of factors is for making cacheable decision and there are 4 factors in this set.

The 12 factors for non-cacheable are listed in Table 3. These factors are usually checked in the order as shown in the table. When one factor is found satisfied, the rest of the factors will not be checked (even if there are still more factors in the HTTP response). Note that the response of an object might contain more than one factor, but it is not necessary for all the factors to present simultaneously.

If an object passes the check of the factors listed in Table 3 and no factor is found satisfied, it will possibly be cacheable. The final decision whether it is cacheable is further decided by some other factors which are listed in Table 4. These factors are also often checked in the order as shown in the table. Again, the response of an object might contain more than one factor, but when a factor is found to be satisfied, the rest of the factors will not be checked.

Table 4: Factors for Cacheability Decision of Web Objects

Factors	Description
fc1	Status code belongs to Class 1, and there exits "Date" header
fc2	Status code belongs to Class 1, and there exits "Last-Modified" header
fc3	Status code belongs to Class 1, and there exits "Expires" header
fc4	Status code belongs to Class 2, and there exits "Expires" header

3. EXPERIMENTAL RESULT

We did trace-driven simulations to investigate the effectiveness of the cacheability factors in current web system. The trace we used for our experiments was from the National Laboratory for Applied Network Research (NLANR) [5]. The trace used in our study contains about 1.36 million requests. Our simulation finds that about 38.4% of them are non-cacheable.

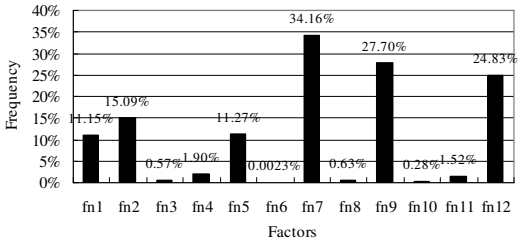


Figure 1: Occurrence Frequencies of Non-Cacheable Factors

Figure 1 plots the occurrence frequencies of non-cacheable factors. From this graph, we see that factors fn7, fn9 and fn12 all occur about 25% and above, while factors fn1, fn2 and fn5 also have considerable contributions. Thus, we see that non-cacheability decisions are mainly due to:

- Content owner's explicit decision (fn1, fn2, fn5), but it might not be related to the actual content change of object (in order words, the explicit decision is found to be too conservative).
- Missing header information (fn9, fn12), which is often due to the inappropriate setting of the web server.
- Nature of the content (fn7).

Figure 2 plots the effectiveness of non-cacheable factors against their respective occurrence frequencies. The effectiveness is obtained by taking the multi-factors co-occurrence effect into consideration. From this graph, we see that factors' effectiveness is generally different from their respective frequencies. This indicates that the effectiveness of the factors is affected by the co-occurrence relationship among them.

In general, the absolute value of the effectiveness of a factor is lower than its frequency. This is because factors often occur in groups. If all factors always occur alone in HTTP responses, their effectiveness would be the same as their respective frequencies. However, if a factor *fnx* often occurs together with other factors, its effectiveness will be lowered. This is because other factors in the factor combination will still make the object non-cacheable when the factor *fnx* is removed. So, a factor occurring in multiple-factor groups would lower its effectiveness. Note that the occurrence frequency of every factor can be as high as 100% while the sum of the effectiveness of all factors can only be 100%.

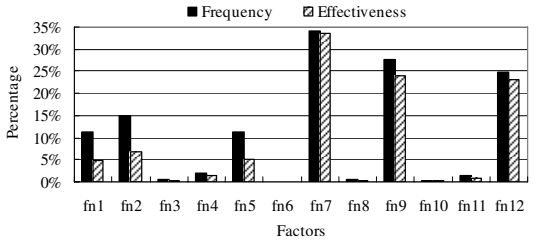


Figure 2: Frequencies and Effectiveness of Non-Cacheable Factors

In Figure 2, we see that the effectiveness of some non-cacheable factors is lower than their respective frequencies more significantly than others. Further study reveals the reason being that different factors occur in different number and different size of factor combinations. If a factor occurs more often in groups or occurs in larger factor groups, its effectiveness will be lowered more significantly.

Factors for cacheable usually take effect after objects pass the check of non-cacheable factors. Objects will be considered cacheable if any factors for cacheable is found satisfied. As we see in Section 2, there are only 4 factors for cacheable decisions. So the multi-factors co-occurrence relationship among those factors is relatively simple.

Figure 3 plots the occurrence frequencies and the effectiveness of cacheable factors. We see that the curve of effectiveness is very similar to the curve of frequency. This is because that the majority of the factors have similar opportunity to occur in groups, so their effectiveness is affected by similar weights.

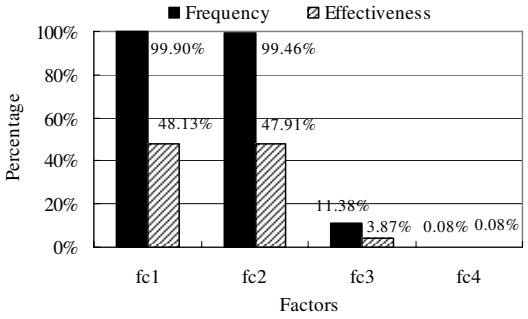


Figure 3: Frequencies and Effectiveness of Cacheable Factors

Table 5 gives all the occurrence combinations of cacheable factors according to their occurrence frequencies. From this table, we can see that the majority of the occurrences of the factors are in groups, only 0.09% of the occurrences have a single factor (i.e. the last two rows). Since the majority of the factors have similar opportunity to occur in groups, the impact of multi-factors co-occurrence on the effectiveness of factors is distributed quite evenly. So the relative distribution of the effectiveness of cacheable factors is quite close to the relative distribution of their respective frequencies.

Table 5: Cacheable Factor Occurrences

Frequency	Factors
88.52%	fc1 fc2
10.92%	fc1 fc2 fc3
0.46%	fc1 fc3
0.08%	fc4
0.01%	fc2

Because the number of factors for cacheable decisions is small and the co-occurrence situation of them is simple, so the relative distribution of the

effectiveness of cacheable factors is quite close to the relative distribution of their respective frequencies. But for cacheable objects, there is another important issue to study, which is the accuracy of the TTLs of them.

4. RELATED WORK

The impacts of caching-related HTTP headers on cacheability decision were investigated in several trace-base studies. [3] collected network traces for modem traffic at UCB in 1996 and analyzed the cacheability of requested web objects. In their study, many caching-related HTTP header fields were examined such as "`Pragma: no-cache`", "`Cache-control`", "`Expires`" and "`Last-Modified`". Ignoring cookies, their experiment results showed that non-cacheable responses were quite low, so did the CGI response. [1] noticed the bias in [3] and considered cookies in their experiments. Their results showed that non-cacheable requests due to cookies could be up to 30%. Later studies on different traces in [2] and [6] showed that the overall rate of non-cacheablility was around 40%.

More complete analysis on content non-cachability can be found in [4, 7, 8]. [8] concluded that the main reasons for non-cacheability included responses from server scripts, responses with cookies and responses without `Last-Modified` header. [4] proposed a complicated method to classify content cacheability using neural network. More recently, [7] analyzed the combinational effects of the causes in content non-cacheability and proposed quantity measure to characterize object cacheability. Content change information has been considered as important factors in all these cacheability models. From previous studies on content cacheability, a large portion of non-cacheable objects is dynamic generated and personalized content. This observation implies the potentials of dynamic web caching.

5. CONCLUSION

In this paper, we systematically study the factors affecting the cacheability of web objects. We dig into the relationship among the co-occurrence factors and reveal the effectiveness of the factors in the multi-factors co-occurrence scenario/situation. Our study reveals the effective factors and proper settings for TTL. By improving them, considerable improvement on the web retrieval performance can be expected.

ACKNOWLEDGEMENT

This research is supported by the funding 2004CB719400 of China.

REFERENCES

[1] R. Caceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich, "Web Proxy Caching: the Devil is in the Details", *Proceedings of the 1998 Workshop on Internet Server Performance,* Madison, WI, June 1998.

[2] A. Feldmann, R. Cáceres, F. Douglis, G. Glass, and M. Rabinovich, "Performance of Web Proxy Caching in Heterogeneous Bandwidth Environments", *Proceedings of IEEE Infocom'99*, March, 1999, pp. 107-116.

[3] S. D. Gribble, E. A. Brewer, "System Design Issues for Internet Middleware Services: Deductions from a Large Client Trace", *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, 1997.

[4] T. Koskela, J. Heikkonen and K. Kaski, "Modeling the Cacheability of HTML Documents", *Proceedings of the 9th World Wide Web Conference*, 2000.

[5] National Lab of Applied Network Research (NLANR) sanitized access log: http://www.ircache.net/

[6] A. Wolman, G. M. Voelker, N. Sharma, N. Cardwell, A. Karlin, H. M. Levy., "On the Scale and Performance of Cooperative Web Proxy Caching", *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, Kiawah Island, SC, December 1999.

[7] L. W. Zhang, "Effective Cacheability", Internal Technical Report, National University of Singapore, 2002.

[8] X. Zhang, "Cacheability of Web Objects", Master Thesis of Computer Science Department, Boston University, USA, 2000.

Bayesian Estimation of Defects based on Defect Decay Model: *BayesED^{3}M*

Syed Waseem Haider
Department of Computer Science
University of Texas at Dallas
waseem@utdallas.edu

João W. Cangussu
Department of Computer Science
University of Texas at Dallas
cangussu@utdallas.edu

Abstract

The estimation of the total number of defects at early stages of the testing process helps managers to make resource allocation and deadline decisions. The use of non-bayesian approaches has proven to be accurate but presents a certain latency to achieve a reasonable accuracy. Here we describe $BayesED^3M$, a bayesian estimator construct upon a existing MLE (Maximum Likelihood Estimator) named as ED^3M. The case studies presented in this paper show that $BayesED^3M$ outperforms ED^3M whenever reasonable historical data is available.

1. Introduction

The major goal of a software testing process is to nd and x, during debugging, as many defects as possible and release a product with a reasonable reliability. Unfortunately, many companies do not use any prediction technique and the release date is solely based on a given deadline and not on the status of the product based on the results of the testing process. Under such circumstances a product with a high number of defects may be released incurring high customer support and dissatisfaction. A trade-off between releasing a product earlier or investing more time on testing is always an issue for the organization. The clear view of the status of the testing process is crucial to compute the pros and cons of possible alternatives. Time to achieve the established goal and percentage of the goal achieved up to the moment are important factors to determine the status of a software testing process. Many techniques, such as software reliability models [9] and coverage analysis [8], can be used to estimate the status of a testing process. Assume the goal of a testing process is to achieve 100% decision coverage. It is easy to see that a product with only 60% of coverage is far from achieving its goal. However, it does not help to determine the time required to achieve such a goal much less what would be the consequences of releasing a product with a lower coverage. Reliability models improve on coverage analysis as time to completion can be estimated as well as side effects of releasing a product with a lower reliability. However, reliability models may be very sensitive to operational pro le which are not easily estimated.

An alternative measure to compute the status of a testing process is the number of remaining defects in a software product which clearly depends on the estimation of the total number of defects. The availability of an accurate estimation of the number of defects at early stages of the testing process allows for proper planning of resource usage, estimation of completion time, and current status of the process. Also, an estimate of the number of defects in the product by the time of release, allows the inference of required customer support. This paper focuses on the prediction of the total number of defects using Bayesian Estimation. The proposed approach is an extension of ED^3M[1] [6, 7] and addresses the limitations discussed in [6, 7]. The limitations were related mainly to the presence of high noise during the initial phase of system testing. The estimator described here is based upon two assumptions. First assumption is that system testing presents an exponential or s-shaped decay in the number of defects. This assumption is con rmed by the data sets used in this study as well as others available in the literature [3, 4]. Second assumption is that prior knowledge about the parameters which represents the total number of defects in the software is available. This assumption mainly addresses the issue of noise in the early phase of system testing.

The remainder of the paper is organized as follows. A description of the approach proposed here is the subject of Section 2. Case studies with industrial data are presented in Section 3. Related work, and conclusion and future work are presented in Sections 4 and 5 respectively.

[1] ED^3M stands for **E**stimation of **D**efects based on the **D**efect **D**ecay **M**odel

2. The $BayesED^3M$ Approach

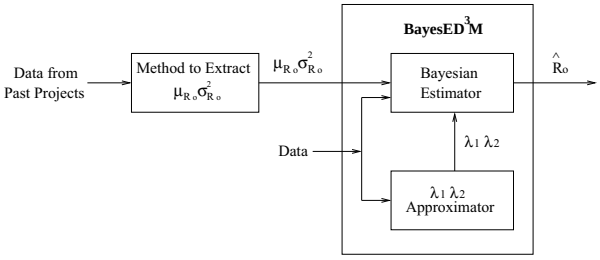


Figure 1. $BayesED^3M$ is composed of Bayesian Estimator and λ_1 and λ_2 Approximator.

We will now describe the **B**ayesian **E**stimation of **D**efects based on **D**efect **D**ecay **M**odel ($BayesED^3M$) [2, 6, 7]. As shown in Figure 1 $BayesED^3M$ is functionally divided in two blocks viz., the "Bayesian Estimator", and the "$\lambda_1\lambda_2$ Approximator". The output is an estimate $\hat{R}_0$ of the total number of defects in the software product under test. Both blocks take current data (the data from the project for which we are interested in nding R_0 or the actual number of defects) as input. The data contains records of defects removed, from the software under test, on either daily basis or weekly basis or any other given xed time unit. We more formally refer to this data as data set. The estimations $\hat{R}_0$ are evaluated incrementally as more data points become available. We provide another input to the "Bayesian Estimator" block called prior knowledge or prior information in the form of prior mean and prior variance of the parameter R_0 which we are estimating. The parameter R_0 represents the initial number of defects present in the software product before the start of testing. The prior knowledge extracted from past similar projects supplements the data set during the initial phase of testing in the estimation of R_0. During the initial phase of the testing the data set does not contain enough information, and the situation is further aggravated by the noise in the testing process. These limitations have been discussed elsewhere [6, 7].

Here, we use a divide and conquer approach by separating the acquisition of prior knowledge from its use and by standardizing it in the form of prior mean and prior variance of R_0. The acquisition of prior knowledge from past similar projects is under study and more is discussed in Section 5. In this section we propose a Bayesian Estimator which uses the prior knowledge. We discuss the behavior of $BayesED^3M$ for various values of prior knowledge in Section 3 for a data set acquired from an industrial project.

As stated in [2, 6, 7], the overall behavior of the decay of defects in the system testing phase is assumed to have a double exponential format as given by Eq. 1.

$$R(n) = R_0\left(\frac{\lambda_2}{\lambda_2-\lambda_1}e^{-\lambda_1 n} - \frac{\lambda_1}{\lambda_2-\lambda_1}e^{-\lambda_2 n}\right) \quad (1)$$

$R(n)$ is the number of remaining defects at day (or any other xed time interval) n, after removing defects at day $n-1$. R_0 is the total or initial number of defects present in the software product. We now de ne in Eq. 2 the random version of Eq. 1 in the form of total number of defects removed from the product under test in n days.

$$\begin{aligned} D[n] &= R_0\left(1 - \frac{\lambda_2}{\lambda_2-\lambda_1}e^{-\lambda_1 n} + \frac{\lambda_1}{\lambda_2-\lambda_1}e^{-\lambda_2 n}\right) \\ &+ w[n] \end{aligned} \quad (2)$$

$D[n]$ is total number of defects removed from the product in n days. Square brackets in $D[n]$ emphasizes that its a random quantity as compared to $R(n)$ which is a deterministic quantity. $w[n]$ is the nth noise component, distributed as Gaussian with zero mean and σ^2 variance, $w[n] \sim \mathcal{N}[0, \sigma^2]$.

The motivation behind the use of a stochastic model is that a deterministic model as de ned by Eq. 1 does not account for random variations which are inbred in the testing process. Nonetheless, noise and unforeseen perturbations are common place in the software testing process, there are multiple sources for these variations such as work force relocation, noise in the data collection process, test of "complex" parts of the system, multiple reports of the same error, among others. An effort to individually account for each such factor, will result in a large number of random variables and an inherently very complex mathematical model. The resulting model would get even more complex when the interference of these random variables with each other is incorporated. And still more complicated when samples taken at different instants of time (dataset) interfere with each other as well. A random variable that consists of sum of a large number of small random variables (various sources of noise), under general conditions can be approximated by a Gaussian random variable [6, 7]. Because of this very reason Gaussian distribution nds its use in a wide variety of man made and natural phenomena. Hence we simplify the model by rst assuming that, a Gaussian noise identically affects the whole testing process. Moreover, the samples taken at different instants are independent to each other [6, 7].

The derivation of an estimator for Eq. 2 would incur the computation of three parameters R_0, λ_1, and λ_2. The complexity of the solution has led us to simplify the problem by rst computing an approximation for λ_1 and λ_2 in the "$\lambda_1\lambda_2$ Approximator" block in Figure 1. We summarize here the description of this block, a detailed account is given in [6, 7]. The block approximates the values of λ_1 and λ_2

in a two-step process. The first step is to find the initial values λ_{1i} and λ_{2i} using a technique called Exponential Peeling [6, 7]. The initial values are then input to a nonlinear regression method in the second step. The nonlinear regression, which is implemented using Gauss–Newton method with Levenberg–Marquardt modification [6, 7], results in the values of λ_1 and λ_2 which are fed to the "Estimator" block. The approximated values are then used to compute $h(n)$ as given in Eq. 3.

$$h(n) = 1 - \frac{\lambda_2}{\lambda_2 - \lambda_1} e^{-\lambda_1 n} + \frac{\lambda_1}{\lambda_2 - \lambda_1} e^{-\lambda_2 n} \quad (3)$$

After substituting the value of $h(n)$ in Eq. 2, we write it in the vector form where **D**, **h** and **w** are vectors of dimensions $N \times 1$

$$\mathbf{D} = R_0 \mathbf{h} + \mathbf{w} \quad (4)$$

Now if we consider R_0 to be deterministic or in other words we do not have any prior knowledge about R_0, then we get the MLE (Maximum Likelihood Estimator) version of $\hat{R}_0$ or simply ED^3M, as given by Eq. 5. For more details on ED^3M refer to [6, 7].

$$\hat{R}_0 = (\mathbf{h^T h})^{-1} \mathbf{h^T D} \quad (5)$$

On the other hand if we do have some prior knowledge about R_0 then we can define a Bayesian version of $\hat{R}_0$ or simply $BayesED^3M$. We now name the two pieces of prior knowledge, i.e. prior mean and prior variance as μ_{R_0} and $\sigma^2_{R_0}$. Assuming that R_0 is also Gaussian distributed we can write more formally as $R_0 \sim \mathcal{N}[\mu_{R_0}, \sigma^2_{R_0}]$. The Bayesian Estimator of $\hat{R}_0$ is given by Eq. 6

$$\hat{R}_0 = \mu_{R_0} + \sigma^2_{R_0} \mathbf{h}^T (\sigma^2_{R_0} \mathbf{h}\mathbf{h}^T + \sigma^2_n \mathbf{I})^{-1} (\mathbf{D} - \mu_{R_0} \mathbf{h}) \quad (6)$$

where $\sigma^2_n = \frac{\sigma^2}{n}$.

3. Case Studies

In this section we explain how the Bayesian estimator behaves and its dependency on the prior knowledge of the parameter to be estimated. Note that prior knowledge is required in the form of prior mean μ_{R_0} and prior variance $\sigma^2_{R_0}$. In the following paragraphs we will discuss the relationship of prior mean μ_{R_0} and prior variance $\sigma^2_{R_0}$, we will analyze the behavior of $BayesED^3M$ for different values of μ_{R_0} over a large fixed interval of $\sigma^2_{R_0}$, such that the minimum value of $\sigma^2_{R_0} << \sigma^2_N$ and maximum value of $\sigma^2_{R_0} >> \sigma^2_N$. Note that σ^2_N is $\frac{\sigma^2}{n}$ at $n = N$, where N is the total size of the data set. The data set used in this section has been acquired from a large industrial project and due to proprietary reasons the actual number of defects has been normalized to 100.

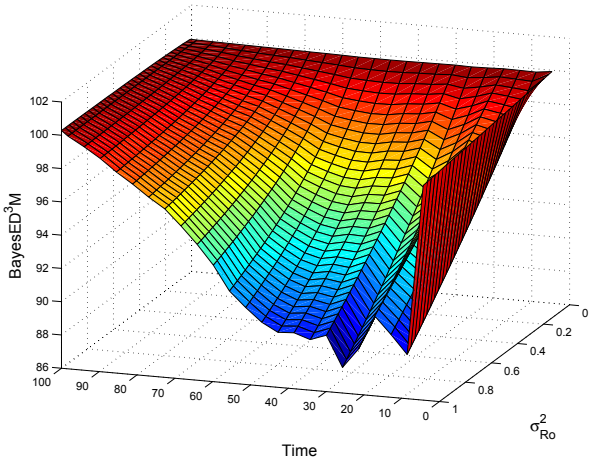


Figure 2. Results of applying $BayesED^3M$ with prior mean μ_{R_0} set to actual number of defects.

Figure 2 shows that if the prior knowledge or the prior mean μ_{R_0} is very close to the actual number of total defects R_0, then it means that the historical data is gathered from past projects using a very efficient method. Naturally our confidence on such method, historical data and the resulting knowledge shall be very high. This confidence is shown by the low prior variance $\sigma^2_{R_0}$. In Figure 2 we have kept the prior mean μ_{R_0} constant and plotted the behavior of $BayesED^3M$ for a range of $\sigma^2_{R_0}$. The surface on the far side which is constant or does not change over the whole course of time is due to the smallest $\sigma^2_{R_0}$ hence the highest degree of confidence. This behavior is natural. When this project, by all means, has very similar characteristics (same development team, similar problem, etc.) as one or more past projects from which the prior knowledge is acquired, then a proportional number (based, for example on average defect density for the projects under consideration) of defects is expected. As we increase $\sigma^2_{R_0}$, the dependency of the estimator on the current data increases and hence the resultant curve drops below the level of prior mean μ_{R_0}. But soon picks up back because $\sigma^2_{R_0}$ is not much smaller than variance of the data σ^2_n. The more we increase the $\sigma^2_{R_0}$ the more the value of the estimator drops and the longer it takes to come back to the actual R_0. This behavior depicts the interplay of the two variances $\sigma^2_{R_0}$ and σ^2_n for a constant μ_{R_0}. Note that because of the high accuracy of prior knowledge and consequently the high degree of confidence, even at the highest value of $\sigma^2_{R_0}$ the results of the estimator does not drop below 86% of actual R_0, which is an optimal lower bound. Hence in this case $BayesED^3M$ is operating in

optimal range and out performs other contemporary estimators as discussed in Section 4.

Now, let us assume a scenario where prior knowledge is not as accurate as before. In Figure 3 we provide $BayesED^3M$ with prior mean μ_{R_0} which is 80% of the actual number of total defects R_0. In the case of very low prior variance $\sigma^2_{R_0}$ $BayesED^3M$ will simply output the input through out the length of time, by over shadowing the data set from the current project. Which means that we had over con dence (very low prior variance $\sigma^2_{R_0}$) on not so accurate prior knowledge (μ_{R_0}) and $BayesED^3M$ is simply showing this contradiction by not converging to the actual R_0. This result also depicts the inef ciency of the method used to collect such prior data. Nonetheless, the balance between the accuracy of the prior knowledge and the con - dence in the historical data and the method is achieved when we increase the $\sigma^2_{R_0}$. In this case $BayesED^3M$ starts with reasonable estimate and then quickly adapts to the current data to produce the accurate estimate.

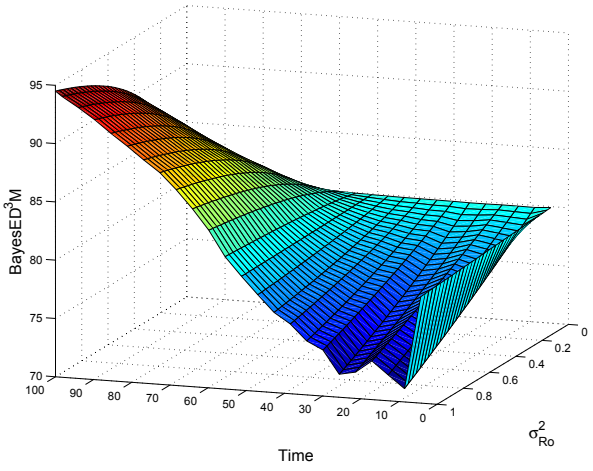


Figure 3. Results of applying $BayesED^3M$ with prior mean μ_{R_0} set to 80% of the actual number of defects.

In general we can conclude that when an initial estimate of μ_{R_0} is within the range of $\pm 20\%$ of the actual number of defects, $BayesED^3M$ converges to a reasonable estimation in a fast manner. However, the value of the variance should be inversely proportional to the con dence in the initial estimate. That is, the higher the con dence the smaller should be the variance. Con icting data will produce misleading results. That is, having a high con dence and consequently a small variance on prior knowledge that is not accurate, results in over shadowing the data from the current project and consequently producing innacurate results.

4. Related Work

Estimation theory and other techniques have already been applied to predict the number of software defects present in a software product. We will provide a quantitative comparison between $BayesED^3M$ and ED^3M, other techniques [10, 11] have already been quantitatively compared with ED^3M in [6, 7] therefore a transitive comparison can be drawn between $BayesED^3M$ and these other techniques. Fenton and Neil [5] have also used Bayesian approach to predict the number of defects. However, since their technique make use of data that is not available to us, we restrict the comparison between $BayesED^3M$ and their approach to a qualitative level.

ED^3M is a Maximum Likelihood Estimator (MLE) also based on Defect Decay Model of system testing process proposed in [2]. The main advantage of ED^3M is that it converges fast to the actual R_0 using information only from the current project (the project for which we are interested in nding R_0 or the actual number of defects). Whereas other estimators, discussed in [10, 11], use more information such as historical knowledge from past projects. It is shown in [6, 7] that ED^3M performs better or as good as these estimators. In many cases where no prior knowledge is available, and so the other estimators cannot be applied, ED^3M produces effective results. As with any other approach ED^3M also has limitation, as discussed in [6, 7]. The limitation of ED^3M is that when the system testing process is poor it causes high corruption of the data (the addition of very high noise to data) which is collected from the ongoing project. Therefore such data severely degrades the performance of ED^3M. To overcome this limitation other sources of information are necessary. Assuming good prior knowledge is available, we need an ef cient estimator to use this knowledge in conjunction with the data available from the current project to estimate the actual number of defects R_0 in the software. This goal is achieved using the $BayesED^3M$ estimator described in Section 2.

We now quantitatively compare $BayesED^3M$ and ED^3M using data acquired from an industrial project. We have successfully run $BayesED^3M$ on other data sets from other industrial projects of varying sizes, but we will use one to show a quantitative comparison due to brevity of space. The results of applying both ED^3M (drawn with '– x –' line) and $BayesED^3M$ are plotted in Figure 4. We have drawn a thick constant solid line of actual R_0 for reference. Note that the graph is normalized along both axes for the protection of the privacy of data. There are two sets of $BayesED^3M$ curves on two ends of reasonably accurate prior knowledge ($\pm 10\%$ of actual R_0) range. Each set has three $BayesED^3M$ curves of varying prior variances $\sigma^2_{R_0}$ for xed prior mean μ_{R_0} to show how prior knowledge affects the performance of $BayesED^3M$ when compared

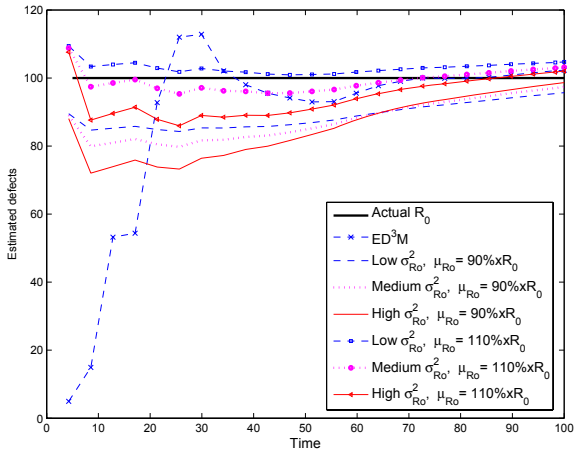


Figure 4. Comparison of $BayesED^3M$ and ED^3M. $BayesED^3M$ is evaluated with $\mu_{R_0} = 100 \pm 10\%R_0$ and also with low, medium, and high values for $\sigma^2_{R_0}$.

to ED^3M. The three variances low, medium and high are all greater than that of the variance of the data σ^2_N and they re ect different levels of con dence in prior mean μ_{R_0} from high to low respectively. The reason for keeping $\sigma^2_{R_0} > \sigma^2_N$ is that when the prior variance $\sigma^2_{R_0}$ is lower $BayesED^3M$ will produce a constant line of prior mean over shadowing the effect of current data as discussed in Section 2. Hence this case does not contribute much to the behavioral analysis and comparison of $BayesED^3M$. Note that the value of σ^2_n for initial data points will be greater than $\sigma^2_{R_0}$. The value of σ^2_n will decrease as the data set will grow and it will become smaller than $\sigma^2_{R_0}$.

Let us rst discuss the set with prior mean $\mu_{R_0} = 110\%R_0$ as seen in Figure 4. In this case low prior variance $\sigma^2_{R_0}$ or high con dence curve given by '–□–' and medium variance or medium con dence curve given by '..○..' performs very well compared to ED^3M. Which is again a valid expected behavior when we have reasonably correct informtion and we are con dent on it. In this case $BayesED^3M$ should take this prior knowledge more into account than the data from the current project and produce estimate accordingly. Even the weak con dence curve given by '–◁–' for the same prior mean μ_{R_0} is almost as good as ED^3M. Strong and medium con dence curves (low and medium $\sigma^2_{R_0}$ respectively) in the second set of curves with prior mean $\mu_{R_0} = 90\%R_0$ gives acceptable estimates. The high prior variance $\sigma^2_{R_0}$ or weak con ndence curve ('—') does not perform as well as ED^3M but de nitely performs better than ED^3M without correction (as shown by curve '-.-.-' in Figure 5) by providing better initial estimates. Figure 5 shows plots of ED^3M for the same data set which is used in Figure 4. The curve ('-.-.-') in Figure 5 is ED^3M without correction. The curve ('-.-.-') depicts the expected exponential behavior but it is not converging fast enough. In order to compensate for the convergence rate we developed a correction technique de- ned in [6, 7]. The other curve ('—') in Figure 5 is ED^3M with correction. Whenever we mention ED^3M we mean ED^3M with correction, otherwise we will state explicitly.

Now we will brie y compare $BayesED^3M$ with the other techniques [10, 11]. Note that since these techniques have already been compared in detail with ED^3M in [6, 7], we will draw conclusions based on transitive comparison. Two of these techniques, Padberg's approach [10] and Software Reliability Growth Model (SRGM) based on Gompertz curve [11], have been quantitatively compared with ED^3M using the data set available in the literature [10]. ED^3M has demonstrated better than Padberg's approach [10]. More over, Padberg's approach is dependent on a number of parameters, as discussed in [6, 7], which are not readily available in the testing process. Hence they limit the applicability of the approach. Note that instead of directly incorporating such parameters which give speci c attributes or pieces of knowledge of data from projects (regardless of current or past projects) into the model, $BayesED^3M$ simply relies on the number of defects found in the current project and $\sigma^2_{R_0}$ and μ_{R_0} from past projects as shown in Figure 1. We provide a clean interface between $BayesED^3M$ and the data from past projects in the form of a Method which will extract $\sigma^2_{R_0}$ and μ_{R_0}. Similar issue exists with Gompertz curve model [11]. The initial values of the three parameters R_o, b and k which characterizes Gompertz curve are not easily estimated.

Fenton and Neil [5] have critiqued software defect prediction models based on different techniques. They have proposed an alternate approach using Bayesian Belief Network (BBN). We would like to address some of the issues that Fenton and Neil have raised about techniques of defect prediction in [5]. They have made an argument based on the work of Adams [1] that only 20% of total defects cause 80% of faults or failures (Pareto Principle). They argue that predicting the count of defects in a product may not contribute to improved reliability. However, reliability is just one of many dimensions of software quality that also includes number of defects. Therefore, predicting the number of defects helps to estimate the quality of the product released and also to infer effort needed for customer support. In addition to that, remaining number of defects is a good measurement of the status of the testing process and helps a testing manager in making decisions related to resource allocation and scheduling issues. Another bene t is that when we do have reliable historical knowledge μ_{R_0} and if the estimate of the total number of defects differs substantially than μ_{R_0}, then it strongly raises suspicion towards the quality of

testing process and the quality (which implies reliability) of the product. Another issue that Fenton and Neil [5] pointed out is that a very accurate residual defect density prediction may be a very poor predictor of operational reliability. We know that defect data re ects the operational pro le used for system testing and when the operational pro le is incorrect defect data is inaccurate as well. We have discussed this issue in [6, 7] for ED^3M, we suggested that with the use of historical knowledge this problem can be alleviated. So work on $BayesED^3M$ addresses this issue.

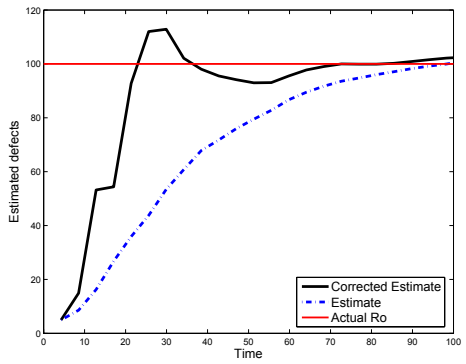


Figure 5. ED^3M with and without correction.

5. Conclusion and Future Work

Results in [6, 7] have shown that ED^3M, a Maximum Likelihood Estimator, in general outperforms other approaches. The same study has shown that high noise in data in the beginning of the testing process greatly impact the outcomes of ED^3M. The higher the noise level, the larger the latency to converge to a reasonable estimation. In order to overcome this problem a Bayesian version ($BayesED^3M$) of the estimator was described here. $BayesED^3M$ uses data from previous similar projects in the form of μ_{R_0} and $\sigma^2_{R_0}$ to compute the estimations. The value of $\sigma^2_{R_0}$ when compared to the variance of the current data dictates the in uence of prior knowledge on the estimation.

It is clear from the case studies that $BayesED^3M$ performs better than other techniques whenever reliable and accurate data is available. However, con icting information such as low variance (high con dence) for inaccurate μ_{R_0} can result in misleading estimations. Also, initial overestimation appear to produce better results than initial underestimation.

Currently we are investigating the use of Clustering techniques to select similar past project. We are studying different attributes of knowledge in projects which can be used to classify projects. These attributes of Projects form a class of similar projects which can be used to extract $\sigma^2_{R_0}$ and μ_{R_0}. We are also studying relationship of attributes of knowledge to the CMMI levels. For example if an organization is on a certain level then which metrics can be collected from this organization. Our conjecture is that the value of $\sigma^2_{R_0}$ can be associated with the level of capability/maturity in terms of con dence level. In simple words the higher the ranking of the company in CMMI levels, the higher is the con dence (smaller value of $\sigma^2_{R_0}$) in μ_{R_0}.

References

[1] E. Adams. Optimizing preventive service of software products. *IBM Research Journal*, 28(1):2–14, 1984.

[2] J. W. Cangussu, R. A. DeCarlo, and A. P. Mathur. A formal model for the software test process. *IEEE Transactions on Software Engineering*, 28(8):782–796, August 2002.

[3] S. R. Dalal and C. L. Mallows. Some graphical aids for deciding when to stop testing software. *IEEE Journal on Selected Areas in Communications*, 8(2):169–175, February 1990.

[4] W. K. Ehrlich, J. P. Stampfel, and J. R. Wu. Application of software reliability modeling to product quality and test process. In *12th International Conference on Software Engineering (ICSE'90)*, pages 108–116, Nice, France, March 1990. IEEE.

[5] N. E. Fenton and M. Neil. A critique of software defect prediction models. *IEEE Transactions on Software Engineering*, 25(5), September-October 1999.

[6] S. W. Haider and J. W. Cangussu. Estimating defects based on defect decay model:ed^3m. *Submitted to IEEE Transactions on Software Engineering*.

[7] S. W. Haider and J. W. Cangussu. A novel approach for defect estimation. Technical Report UTDCS-30-05, The University of Texas at Dallas, August 2005.

[8] B. Marick. *The Craft of Software Testing*. Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[9] J. Musa. *Software Reliability Engineering*. McGraw-Hill, 1999.

[10] F. Padberg. A fast algorithm to compute maximum likelihood estimates for the hypergeometric software reliability model. In *Second Asia-Pacific Conference on Quality Software*, pages 40–49. IEEE, December 2001.

[11] D. Satoh and S. Yamada. Discrete equations and software reliability growth models. In *12th International Symposium on Software Reliability Engineering, (ISSRE)*, pages 176–184. IEEE, November 2001.

A Component Model to Support Dynamic Unanticipated Software Evolution

Hyggo Almeida, Angelo Perkusich, Glauber Ferreira, Emerson Loureiro, and Evandro Costa
Embedded Systems and Pervasive Computing Lab,
Center of Electrical Engineering and Informatics, Federal University of Campina Grande
Postal Code 10.105 - 58109-970, Campina Grande, PB, Brazil
{hyggo,perkusic,glauber,emerson}@dee.ufcg.edu.br

Abstract

This paper presents a component model to support dynamic unanticipated software evolution. Such a model provides mechanisms for managing unpredicted software changes on the fly. A Java implementation of the proposed model is also presented. Finally, an application of the proposed infrastructure in the context of pervasive computing is described.

1. Introduction

Software evolution is a set of activities that occurs after the initial delivery of the software and typically deals with bug x es and the addition, change or removal of functionalities. When the evolution cannot be anticipated and the software is not prepared for changes, the impact over the existing design and code increases [4]. These unanticipated changes have been pointed as the main cause for most technical problems and related costs of software evolution [8].

Recent advances in Software Engineering, such as application frameworks [5], component based systems [3], service oriented architectures [10], and plugin based development [9], have not been conceived to support unanticipated changes. In fact, some approaches provide e xibility for changes, even at runtime. But, it is only valid for speci c parts of the software, which are predicted to change - framework hot spots, services and plug-in interfaces, etc.

The great problem arises when a part of the software which was thought to be x ed has to change. The unanticipated changes force the developers to extensively modify the existing software architecture, design, and code. A change is considered "unanticipated" if its implementation is not dependent of hooks encoded in previous versions of the changed software [8].

By de nition, "unanticipated software evolution is not something for which we can prepare during the design of a software system" [1]. Therefore, a support for unanticipated software evolution must not require from the developer a speci cation of which part of the software could evolve. The software must inherently support evolution, in any part, and the developer should not be aware of the mechanisms that allow this evolution. Moreover, in the case of systems with frequent requirement changes or when the execution cannot be interrupted, this evolution must be managed at runtime.

In this paper we propose a component model to support dynamic unanticipated software evolution, named COMPOR COMPONENT MODEL SPECIFICATION (CMS). Such a model allows to change any part of the constructed software, removing and adding components, even at runtime. The CMS promotes unanticipated software evolution through a simple and lightweight design model. Component based concepts, such as containers and components, are used to build applications based on a hierarchical composition. We present a Java implementation of the CMS, called JAVA COMPONENT FRAMEWORK (JCF). By using the JCF, it is possible to develop Java applications that can be changed during runtime, even for unpredicted changes. Finally, an application of the proposed infrastructure for developing a pervasive computing middleware is presented.

The remainder of this paper is organized as follows. Section 2 describes the CMS. Section 3 introduces the JCF. An application of our approach is presented in Section 4. Related works are discussed in Section 5. Finally, the concluding remarks are presented in Section 6.

2. Component Model Specification

According to the COMPOR COMPONENT MODEL SPECIFICATION (CMS), a component based system must be described as a composition of two kinds of entities: functional components and containers. Functional components are software entities that implement the application-speci c functionalities, making them available by means of services and events. A functional component is not composed of other components, that is, it has no child components. Functional components represent the atomic architectural elements of the software application according to the CMS.

Containers are software entities that implement no application-speci c functionalities. A container manages the access to the services and events provided by its child components, which may be functional components or other

[1] FUSE Workshop - ETAPS 2004

containers. Functional components are made available by inserting them into containers. It is necessary to have at least one container, named *root container*, in order to insert functional components and run the application. Each container has a list of provided services as well as a list of events of interest for its child components.

2.1. Component Deployment Model

As mentioned before, each container has a list of provided services as well as a list of events of interest for its child components. Therefore, after inserting a new component in a given container, the list of services and events for each container up to the root of the hierarchy must be updated accordingly. This is necessary in order to make available the services provided by the inserted component as well as to allow the noti cation of the occurrence of events that are of interest of the component. Figure 1 depicts the component deployment process and its steps are detailed as follows.

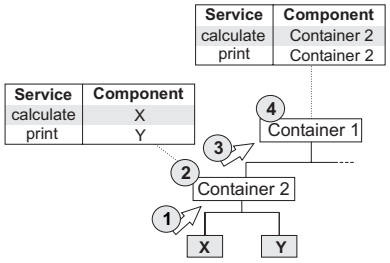


Figure 1. Component deployment - updating service and event tables for the child components.

1. A component *X* implementing the "calculate" service is added to the *Container 2*.
2. *Container 2* updates the service table adding the services provided by the component *X* (the same occurs for the event table).
3. *Container 2* asks its parent container, *Container 1* in this case, to update its service table, which adds the services provided by the *Container 2*.
4. *Container 1* updates its service table.

After the execution of these steps the services provided by the component *X* can be accessed from any component in the hierarchy without an explicit reference to it. Note that a component has only the reference to its parent container. In the case of two or more components having services or events with the same identi er , an *alias* is used. Thus, it is possible to register a nickname for each service, allowing services providers to coexist within the same application and be diversi ed in terms of non-functional features.

The remove operation is similar to the deployment, but after removing a component the next invocations for its services will not work.

2.2. Interaction Model

The interaction model is based on services and events. In the rst case any component may invoke a service of another component, even when the component belongs to an other container. The interaction based on events focuses on the announcement of a state change in a given component to all the interested components. In both cases there is no explicit reference among components.

2.2.1 Service based interaction

As said before, after inserting a component in a given container, its services are made available to any other components in the application. Therefore, assuming the existence of a service named "save" implemented by a component *K* of the application, the execution of this service can be requested by a component *X*, without an explicit reference to *K*. In Figure 2 such an interaction process is illustrated and detailed as follows.

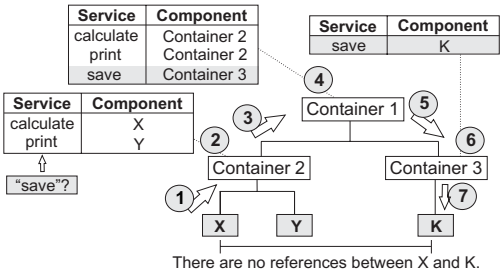


Figure 2. Service based interaction - localization and execution without explicit references.

1. Component *X* requests the execution of the "save" service to its parent container.
2. Based on its service table, *Container 2* veri es that no child component implements the "save" service.
3. *Container 2* forwards the request to its parent container, in this case *Container 1*.
4. *Container 1*, according to its service table, veri es that one of its children implements the "save" service, *Container 3* in this case. The *Container 1* sees *Container 3* as the component that implements the requested service.
5. *Container 1* then forwards the service request to the *Container 3*.

6. *Container 3* does not implement the service but has a reference to the component that implements the requested service, and forwards the request to it, in this case component *K*.

7. Component *K* executes the "save" service and returns the result following the reverse path, back to the requester.

It is important to point out that there are no references between the component requesting the service (*X*) and the component that provides it (*K*). Thus, it is possible to change the component that provides the "save" service without modifying the rest of the structure.

2.2.2 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 noti ed. 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 veri es if any of its child components have to be noti ed 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*.

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*). Thus, the component that announces the event can be changed without modifying the rest of the structure.

2.3. Overriding Services: the Black-Box Inheritance Mechanism

Due to the container-mediated deployment and interaction models, CMS provides mechanisms to override services via "black-box inheritance". In other words, it is possible to reuse component services without extending the component code, even at runtime. This is possible because a component can require a service implemented by itself. Since the services provided by functional components are accessed via the container, it is only necessary to publish internal component functionalities as external services and access them via a container. Figure 4.a illustrates an example of this process. Consider the "readFile" service, which is implemented by the sequence of functionalities: "buffering" and "IOoperation". If the internal functionalities are published as services, *Component1* can access them via container. Then, the "readFile" service is decoupled of "buffering" and "IOoperation" functionalities.

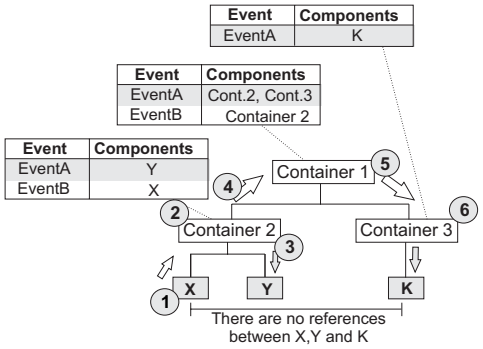


Figure 3. Event based interaction - event notification without explicit references.

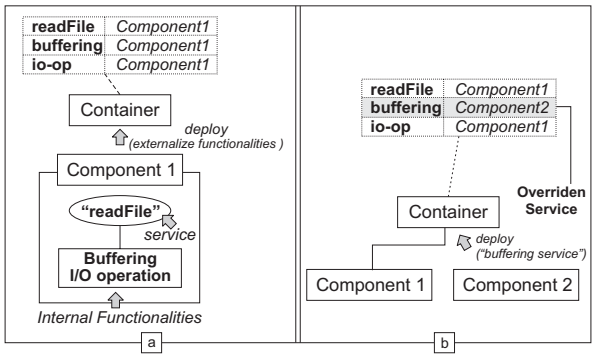


Figure 4. Service overriding - internal functionalities as component services.

To override the "buffering" internal functionality, for example, it is only necessary to deploy a component that provides a service with the same name. Figure 4.b presents the service overriding process. In this gure, *Component2* overrides the service "buffering" of the *Component1*. Since there are no references to the *Component1*, this process, which is based on the *Template Method* and *Strategy* design patterns [6], can be performed at runtime. After deploy-

ing the *Component2*, the "readFile" service of *Component1* becomes based on the "buffering" service implemented by the *Component2*. It is important to note that *Component2* does not have to extend *Component1*, it is only necessary to know the provided and required services.

2.4. Recursive Composition: Applications as Components

One can think that a at architecture could be always better. It could be more interesting in terms of performance and will not require the usage of containers. Also, it will work similarly to registry based systems, event based systems and service oriented architectures, like Jini [13]. However, the main advantage of a hierarchy of containers is that it allows to maintain cohesion of the functionalities provided by their child components. It makes possible to reuse entire containers without needing to understand the internal components or other containers. Also, it allows recursive composition of applications, since root containers can be viewed as components for other containers. Therefore, an application can be built by integrating containers of various CMS based applications (Figure 5). This feature is not straightforward for architectures based on events, registry or services.

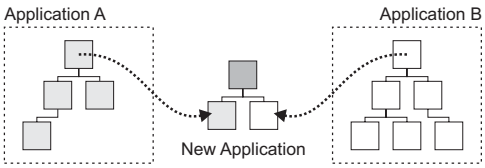


Figure 5. Composition of applications.

Based on the service and event models, black-box inheritance and recursive composition, the CMS supports all kinds of evolution scenarios: component change, addition, and removal; service and event changes; and architectural changes. In face of unanticipated evolution, an entire application could be dynamically changed. It is only necessary: i) to identify which components will change; ii) to identify which are their dependencies (services and events) and, if necessary, change them; iii) and nally change the identi ed components. The effort needed to perform this evolution depend on the functional cohesion and complexity of the application. In fact, it could be hard but using the CMS it is always possible.

3. Java Component Framework

The JAVA COMPONENT FRAMEWORK (JCF) is a Java implementation of the CMS. The JCF design is based on the *Composite* design pattern [6], which can be applied to hierarchical architectures. Figure 6 shows a simpli ed version of the JCF class diagram, describing its main methods. `Container` and `FunctionalComponent` classes are instantiated in containers and functional components, respectively. The abstract class `AbstractComponent` insures the recursive composition [6]. Thus, containers are not aware if their children are functional components or other containers. Additionally, it implements the accessor methods. The methods declared in the class `AbstractComponent` are differently implemented by `FunctionalComponent` and `Container` classes, for both the service and the event interaction models.

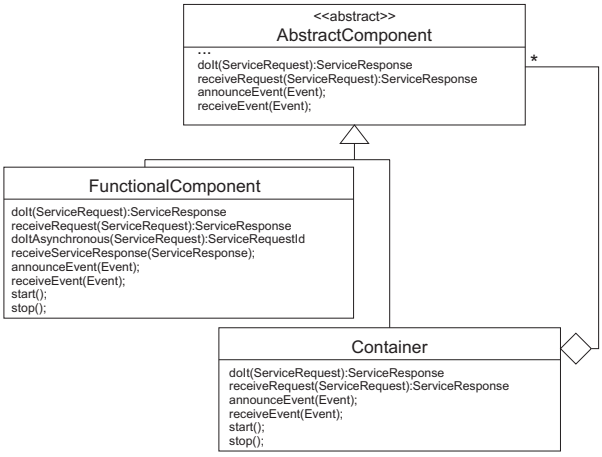


Figure 6. Simplified JCF class diagram.

The service interaction model is implemented through iterative invocations of the `doIt` and `receiveRequest` methods. Such methods are invoked by the components and containers of the hierarchy until the service provider component is located. The function of the `doIt` method is to forward the service request, in a bottom-up way, until reaching the container that contains the reference to the provider. When this occurs, the `receiveRequest` method is invoked, in a top-down way, until reaching the functional component that implements the service (Figure 7). The syntax for the service invocation methods are `doIt(ServiceRequest)` and `receiveRequest(ServiceRequest)`, where `ServiceRequest` is an object that encapsulates a service name and the parameters needed to execute the service. The result of those methods is a `ServiceResponse`, which encapsulates the service execution result or the exception, if it occurs.

The JCF also implements an asynchronous version of the service based model. The asynchronous interaction implementation is based on the *ActiveObject* design pattern [12]. A component asynchronously invokes a service through the method `doItAsynchronous` and receives a request identi er (`ServiceRequestId`). Then, a new thread is started to request the service through the method `doIt`. When the return from `doIt` occurs, it invokes the method `receive ServiceResponse` for the service requester component, forwarding the service request and the service identi er . Based on the service identi er , the requester component identi es to which invocation the re-

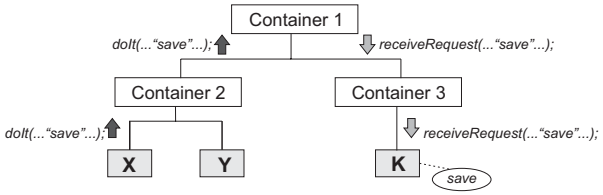


Figure 7. Execution of the `doIt` and `receiveRequest` methods.

ply refers to.

The implementation of the event based interaction model is based on the *Observer* [6] and *ActiveObject* [12] design patterns. The functionality is implemented through the asynchronous invocation of the method `announceEvent` to announce events to the parent containers (bottom-up). On the other hand, the invocation of the method `receiveEvent` noti es the events to the interested components (top-down), as occurs with services.

Besides the interaction models speci ed by the CMS, the JCF implements initialization properties for components. Moreover, JCF provides a mechanism for starting and stopping the execution of the components. The initialization properties are stored in a table for each functional component and can be accessed through the `getInitialization Parameter(String)` method, whose argument is the name of the required initialization parameter. The component initialization is implemented by the `start` and `stop` methods. For containers, these methods start/stop all of its components through the invocation of their respective `start` and `stop` methods. For functional components, these methods invoke template methods [6] that are implemented by the component developer, which initialize/interrupt the execution according to the component needs.

4. Application: Wings Pervasive Middleware

Wings is a middleware for pervasive computing that is guided by three issues: context-sensitivity; networking support e xibility; and interoperability both in terms of networking protocol stack and programming language. The basis of the middleware lies on the concepts of *resource*, *context* and *peer*. We de ne a *resource* as an entity with a description, through which it can be shared, discovered and downloaded, such as an audio le. A *context* encapsulates information about a local peer and the environment on which it is immersed. Finally, *peers* are de ned as network nodes having the following set of capabilities: search and sense other peers; share, discover, and download resources, as well as deliver context information.

Due to the sensing capability, Wings has been designed for "infrastructureless" environments. Therefore, the communication between peers must be performed in an ad-hoc way. This characteristic enhances the applicability of Wings in the world of pervasive computing, where the infrastructure is something we cannot always count on. This approach provides the necessary tools to develop applications (*Winglets*) for ad-hoc like pervasive environments such as mobile virtual communities and mobile le sharing. Using Wings, pervasive applications could take advantage of multiple con gurations by performing host discovery over different network infrastructures, possibly at the same time. Based on this approach, an application could, for example, discover hosts through *UPnP* (`http://www.upnp.org`), *JXTA* (`http://www.jxta.org`) and *Zeroconf* (`http://www.zeroconf.org`) protocols. This improves the acquisition of context information, since more hosts can be discovered by the applications.

However, it is very dif cult to predict which of such protocols will supply the needs of different applications. Moreover, mobile devices are still very limited concerning memory and storage capacities. Therefore, it would not be reasonable to embed in such devices all the existing network infrastructure protocols for each of their wireless interfaces. It becomes necessary a mechanism for inserting and removing such implementations from a device, whenever needed. For that, we use the CMS to encapsulate the peer discovery algorithm and context information mechanisms in software components. Such components, namely Network Infrastructure Components (NICs), may be plugged in and out from the middleware even at runtime.

The CMS successfully provides an effective way of changing *NICs* at runtime, without affecting the rest of the middleware. This is an important feature in a pervasive environment, where the networking protocols involved may change, but users do not want to stop their tasks for replacing one protocol for another. In other words, this process should be performed transparently.

5. Related Work

Different kinds of component models have been proposed. Some examples are *Sun JavaBeans* and *Enterprise Java Beans* (`http://java.sun.com`), and the *CORBA Component Model* (`http://www.omg.org`). Such models have been successfully applied for constructing corporative applications and their middleware implementations provide many interesting services for enterprise software development. However, these models were not conceived to support dynamic unanticipated software evolution. In some cases, their middlewares provide mechanisms and services to perform dynamic changes, but this is not de ned in a component model level. The lack of this feature makes dif cult the construction of systems supporting unanticipated evolution.

CMS has also some similarities with service oriented architectures: service publishing and provision, transparency of the service provider, exibility for changes, among others. For instance, Jini [13] is a Java-based technology for the provisioning of services among network nodes. In Jini,

services are advertised and discovered in central repositories, like a distributed CMS single (root) container. The work presented in [7] describes a Java middleware for providing services in ad-hoc networks. Such middleware, implemented in Java, is based on a distributed service registry, where each node of the network is able to provide and use services. Other example is the OpenWings (`http://www.openwings.org`) framework, which aims at providing service provisioning features targeted to dynamic networks. In these works, dynamic evolution is not supported. In the work of Piccinelli et al [11] the main focus is the dynamic composition of services, and recursive composition of services is also allowed. However, dynamic features are only related to service loading, unanticipated changes are not tackled.

In the context of dynamic evolution models, we highlight HADAS [1]. In HADAS, the focus is the interoperability between distributed components. The re ection concept is used to de ne the dynamic composition. The developer should de ne a set of *fixed* behaviors and another set of *extensible* behaviors for the application. Only *extensible* behaviors can be dynamically composed and changed. Considering that the changes cannot be predicted, the definition of x ed components imposes many dif culties and in some cases makes no sense, since any component can be eventually changed. Another work is Gravity [2] which puts concepts from service and component orientation together for de ning a model that supports the adding and removal of components at runtime. For these works, there are no mechanisms to provide recursive composition and the dynamic composition is only allowed for explicitly de ned non x ed parts.

6. Concluding Remarks

This paper presented a component model to support dynamic unanticipated software evolution. Such a model, named CMS, provides mechanisms for managing dynamic changes on the software without predicting them, on the y . A Java implementation of the CMS which allows constructing Java applications supporting unanticipated software evolution was also presented.

Due to space restrictions, some efforts related to this work were omitted. For example, a performance evaluation model has been conceived and code pro ling has been performed. Based on performance evaluation results, it can be observed that the hierarchical infrastructure of the CMS architecture does not impact the system performance signi - cantly.

Besides, to make possible a large scale development, an Eclipse-based tool to support the composition activities, called CCT, has been developed. Also, an infrastructure for developing enterprise systems has been built to deal with distribution, security, web, and persistence issues. Persistence is an important feature to allow construct stateful components. Finally, the CMS has been implemented in Python and C++. Multi-language implementations are very important to consolidate the CMS, and also to apply it to different contexts and platforms. The simplicity of the CMS makes possible to implement dynamic evolution without requiring object oriented languages complex mechanisms.

The infrastructure presented in this paper represents a novel engineering support for constructing applications with support to unanticipated evolution. Using CMS, JCF, and CCT, applications can be developed based on reuse, besides improving e xibility and reducing maintenance and evolution time and costs.

References

[1] I. Ben-Shaul, O. Holder, and B. Lavva. Dynamic Adaptation and Deployment of Distributed Components In Hadas. *IEEE Trans. Softw. Eng.*, 27(9):769–787, 2001.

[2] H. Cervantes and R. S. Hall. Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model. In *Proceedings of the International Conference on Software Engineering (ICSE)*, page 614623. IEEE Computer Society, May 2004.

[3] I. Crnkovic. Component-based Software Engineering - New Challenges in Software Development. In *Software Focus*, volume 4, pages 127–133. Wiley, 2001.

[4] P. Ebraert, Y. Vandewoude, T. D'Hondt, and Y. Berbers. Pitfalls in unanticipated dynamic software evolution. In *Proceedings of the Workshop on Reflection, AOP and Meta-Data for Software Evolution(RAM-SE'05)*, 2005.

[5] M. Fayad, R. Johnson, and D. Schmidt. *Building Application Frameworks*. Wiley, 2000.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, 1995.

[7] R. Handorean and G.-C. Roman. Secure Service Provision in Ad Hoc Networks. In *Proc. of the First International Conference on Service Oriented Computing*, volume 2910 of *Lecture Notes in Computer Science*, pages 367–383, Trento, Italy, 2003. Springer Verlag.

[8] G. Kniesel, J. Noppen, T. Mens, and J. Buckley. First International Workshop on Unanticipated Software Evolution. In *ECOOP2002 Workshop Reader*, volume 2548 of *LNCS*. Springer Verlag, 2002.

[9] J. Mayer, I. Melzer, and F. Schweiggert. Lightweight Plug-In-Based Application Development. In *NODe '02: Revised Papers from the International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World*, pages 87–102. Springer-Verlag, 2003.

[10] M. P. Papazoglou. Service-Oriented Computing: Concepts, Characteristics and Directions. In *Proc. of Fourth International Conference on Web Information Systems Engineering*, pages 3–12, Rome, Italy, December 2003. IEEE.

[11] G. Piccinelli, C. Zirpins, and W. Lamersdorf. The FRESCO Framework: An Overview. In *Proceedings of the 2003 Symposium on Applications and the Internet Workshops*, pages 120–123, Orlando, USA, January 2003. IEEE Computer Society.

[12] J. Vlissides, J. Coplien, and N. Kerth. *Pattern Languages of Program Design 2*. Addison-Wesley, 1996.

[13] J. Waldo. The Jini Architecture for Network-centric Computing. *Communications of the ACM*, 42(7):76–82, 1999.

Abstract Logic Tree based Framework for Component Based Solution Composition Design and Execution

Wei SUN; Xin ZHANG; Ying LIU; Zhong TIAN
IBM China Research Lab
{weisun;zxin;aliceliu;tianz}@cn.ibm.com

Abstract

Component based development is gaining increasing focus for agile system development through reuse existing asset. To make this approach effective enough to implement, the paradigm of "solution = components + composition logic" should be successfully realized. The current component composition approaches are either not flexible enough, less visibility of the composition logic or demanding heavy platform and technology dependencies. Abstract Logic Tree is a mechanism of using task decomposition approach based on divide-and-conquer strategy to support composition design and execution. In this paper, the design of Abstract Logic Tree and related methodology, its composition logic modeling capability and a supporting framework called DynaTree are presented. And the travel plan case is used to demonstrate the solution development process through the Abstract Logic Tree method.

1. Introduction

Given the economic pressures, software development and maintenance are being demanded to conduct in an adaptive and cost effective approach. The amount of existing software in organizations and COTS (Commercial On The Shelf) [1] on the market that can be reused is increasing. Together with the development of component integration and composition technology, Component Based Development (CBD) [2] has become an emerging method in computing practice [8].

The foundation of CBD approach is that new software solution will be composed through reusing components [9] rather than developing new solution from scratch which can be summarized as a paradigm, "solutions = components + composition logic". This paradigm means that two development steps should be covered while developing a solution: (i) the specification and implementation of components and (ii) the composition approach of components into composites or solutions [7]. While considerable experience in component technology and many resources have been spent in defining component models such as CORBA, COM, EJB, and Web Service [4][5], much less effort is spent in investigating appropriate composition languages. Component composition using object-oriented languages offers some features supporting component-based programming, such as encapsulation of state and behavior, inheritance, and late binding. But, unfortunately, they are not powerful enough to provide flexible and type-safe component composition and evolution mechanisms, respectively. BPEL (Business Process Execution Language) is a standard composition language that choreographs service with control flow [6]. But it is still too complex to be understood by business people, and in practice, that brings some issues in both solution development and maintenance. Naiyana and Kajal have developed XCompose, an XML based component composition framework. The primitive composition operators and language have been defined by XCompose to describe the composition logic [3].

In this paper, we come up with a brand-new component composition model, the Abstract Logic Tree (ALT). The ALT model is designed to follow man's basic problem-solving pattern for task decomposition. By adopting the divide-and-conquer principle, instead of the control flow method in BPEL, the component composition problem is solved by task decomposition in a tree structure with execution logic attached to it. And the runtime support traverses the tree according to the associated logic design to fulfill the composition work. The structure and presentation of ALT model not only bring advantages in component-based solution design, but also bring more benefits such as transaction support, and easy maintenance. Abstract Logic Tree is platform and component's implementation technology independent, and also provides an intuitive means for the composition design.

The rest of this paper is structured as following: The design of Abstract Logic Tree is described in Section 2; and Section 3 introduces the framework, DynaTree, to support the ALT design and execution; in Section 4, we demonstrate the effectiveness of ALT through a case study; finally, the conclusions given.

2. Design of Abstract Logic Tree

When handling a complex business task in real world, the most fundamental approach we often using is the divide-and-conquer approach: to divide the complex business task into a certain sequence of sub tasks and have them done in a certain order. And through iteratively applying the approach, the high-level, complex business task can be decomposed into a set of doable work items. Then by traversal all these work items in certain logic, the business task can be fulfilled. The approach not only enables a clear-cut way for problem solving, but also featured for adaptive for change and easy to manage.

The ALT adopts the same principle by using a tree structure to compose components into business process. A business process is modeled by ALT as an ordered tree that organizes the component invocation (the leaf nodes of the tree, vertex of degree 1) through the composition logic (the internal nodes of the tree, node other than leaf node and root node). Based on the tree skeleton, detailed processing logic is attached to make the tree executable. And through certain traversal method, the business process can be automated.

In the following sub-sections, we will firstly introduce the model elements of Abstract Logic Tree, then we will describe the composition logic, in the last part, we will discuss the traversal method and some other issues related to component composition.

2.1 The Structure of Abstract Logic Tree

An Abstract Logic Tree contains a series of nodes and links that chain from parent node to child node. Node represents the work to be done to achieve a certain goal.

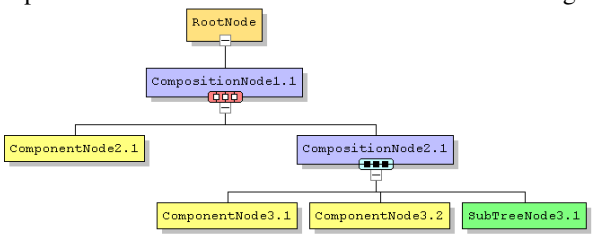


Figure 1. Abstract Logic Tree Sample

As illustrated in the Figure 1, there are four different types of node have been defined on the Abstract Logic Tree. RootNode represents the overall composition objective of the tree; CompositionNode represents the composition objective of all its underlined branches; ComponentNode represents the reference of a ready-to-use component; SubTreeNode represents the reference of another abstract logic tree to fulfill a certain complex composition task. The links connecting the nodes show the decomposing procedure as a path from root to all the leaves. The leaf nodes on the tree should be executable which could either be ComponentNode or SubTreeNode. As depicted in the following UML model of Abstract Logic Tree, a RootNode should be composed by Node, which can be CompositionNode, ComponentNode or SubTreeNode as implementations. A CompositionNode can have a series of CompositionNode, ComponentNode and SubTreeNode as children nodes.

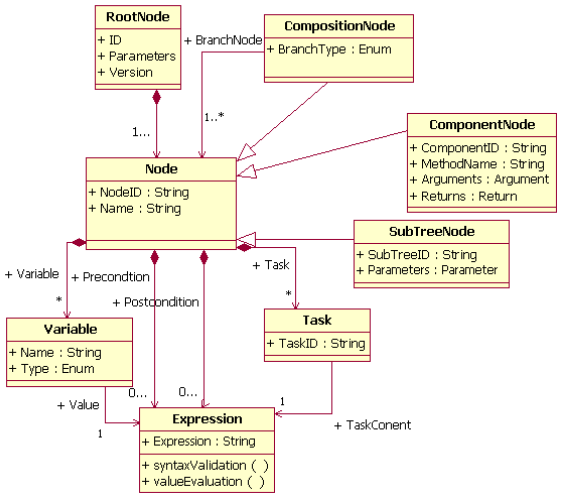


Figure 2. UML Model of Abstract Logic Tree

Every types of Node have the following key properties:

Precondition: The node entry condition. The node will not be processed if this condition is not satisfied.

Postcondition: The node exit condition. The node processing will not complete until this condition is satisfied.

Variable: The data that can be defined on a node, and manipulated and referred by itself and its children nodes. Variables can be manipulated by Tasks and can be referred by conditions and Arguments or Parameters to be used by component and sub-tree invocation.

Task: Variable manipulation on any node processing.

The value of all the above properties can be defined by expression according to certain syntax rules (for example: XPath), which can be processed by corresponding expression processor (for example: XPath engine).

CompositionNode, ComponentNode and SubTreeNode have their unique properties beside those defined by Node. CompositionNode has a property called "BranchType", which defines its children nodes' processing behavior, either in sequence or paralleled; if the "BranchType" value is "*Sequence*", the children nodes have a sequence processing order from left to right. If the "BranchType" value is "Parallel-Sync", the child nodes will be processed in parallel and joined synchronized upon processing completion of any branch; if the "BranchType" value is "Parallel-Async", the paralleled processing of

child nodes are not necessary to be synchronized (all the branches are finished) before marking the completeness of their parent node. ComponentNode is responsible to invoke the external component to achieve certain well-defined functionality. The argument is the input of the ComponentNode and return is the output of the ComponentNode. SubTreeNode is responsible to invoke another abstract logic tree by feeding corresponding parameters. The approach of variable parameters is used.

2.2 Composition Logic Modeling Capability

Abstract Logic Tree is capable to satisfy most of the composition logic requirements. Naiyana has defined a set of primitive composition operators to describe the composition logic according to the principals of "minimal semantics", "complete" and "correct". Subsequently five composition operators have been defined in the framework of XCompose, namely *conjunction*, *sequence*, *choice*, *pipe* and *loop* to enable different compositions of methods from components. In the following table, we have listed the corresponding Abstract Logic Tree representation to describe the five types of composition logic.

Beside the above five primitive composition logic, abstract logic tree can also support more advanced workflow patterns ([10] and [11]):

Sync-Conjunction: Set BranchType as "Parallel-Sync", the paralleled processing should be synchronized.

Async-Conjunction: Set BranchType as "Parallel-ASync", the paralleled processing does not need to be synchronized.

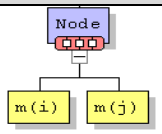
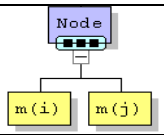
OR-Choice: Set un-exclusive preconditions for each branches, therefore multi branches can be processed. These branches can also be synchronized joined or asynchronous according to the BranchType definition.

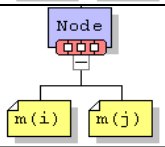
XOR-Choice: Set exclusive preconditions for each branches, therefore only one branch can be processed.

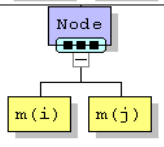
Nesting: Abstract logic tree can be encapsulated into a SubTreeNode; it can be reused in other abstract logic tree.

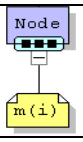
Based on the above analysis, Abstract Logic Tree is capable to describe most of the component composition logic requirements.

Table 1. Composition Logic Modeling Using Abstract Logic Tree

Composition Operators Defined in XCompose	Abstract Logic Tree Presentation	
Conjunction represented as $m_i \wedge m_j$ Denotes the execution of the two methods (m_i and m_j) simultaneously.		Node.BranchType = "Parallel-Sync" or "Parallel-Async" ; m(i).precondition and m(j).precondition are simultaneously satisfied;
Sequence represented as $m_i ; m_j$ Denotes the execution of the two methods (m_i and m_j) in sequence.		Node.type = "Sequence"; m(j).precondition = completion of m(i)

Choice represented as $m_i \vee m_j$
Denotes that the composition method consists of the semantics of either the method m_i or m_j (not both).
		Node.type = "Parallel";
		m(i).precondition = con1;
		m(j).precondition = con2;
		con1 AND con2 = false;
		con1 OR con2 = true;

Pipe represented as $m_i \mid m_j$ Denotes the execution of the two methods m_i and m_j in sequence, where the output of m_i is the input of the m_j .		Node.type = "Sequence"; m(j).argument = m(i).return;

Loop represented as m_i^*
Denotes the repeated consecutive execution of the method m_i .
		m(i).precondition = con1;
		m(i).postcondition = con2;
		m(i) will be iteratively processed until con2 is satisfied.

2.3 The Traversal Method

The ALT traversal method extends the pre-order algorithm of tree structure to support more powerful processing logic, e.g., parallel processing, loop, etc. In the pre-order algorithm of ordered tree, the parent node is visited first, and then the child nodes are visited in order. The ALT traversal method take the similar algorithm but different in two points: firstly, the execution of postcondition of the parent node is delayed to the time after the child nodes executed, and secondly, the traversal of child nodes is done according to the composition operation. The first extension enables the loop logic to be expressed as when the postcondition of the parent node isn't meet, the

execution of child nodes will be repeated. And the later can enable the flexibility of execution sequence of child notes.

3. Composition Design and Execution Framework

Based on the Abstract Logic Tree concept, we have designed the framework called DynaTree for software component composition. As illustrated in the Figure 3, DynaTree consists of a design tool and an engine, therefore provides end-to-end capabilities from design, simulation, deployment to runtime execution and administration.

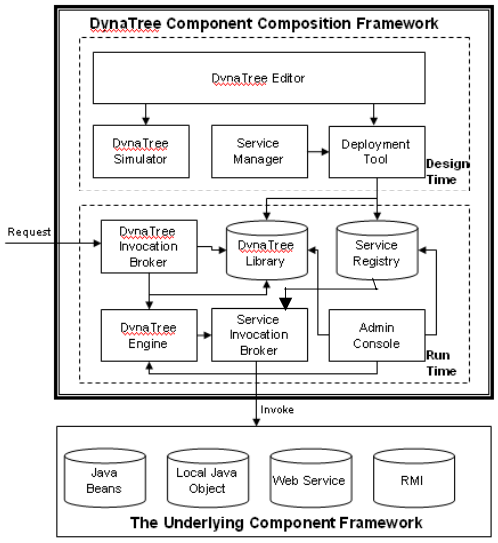


Figure 3. DynaTree Framework

DynaTree Design tool is based on eclipse platform. It consists of the DynaTree Editor, the DynaTree Simulator, the Service Manager and the Deployment tool. As shown in Figure 4, DynaTree Editor provides the visual environment for user to design DynaTree script, which will be stored as XML and can be manipulated as file in the Navigator window. The tree structure can be easily edited in the Canvas window to illustrate how a relatively complex task is decomposed into sub-tasks. The specifications of each node on the tree can be defined in the Property window, which include node type, variables, conditions, service invocation parameters, exception handling rules, etc. While the leaves nodes of the tree are the final "atomic" actions to compose the complex task, which subsequently will be fulfilled by corresponding software components registered as services in the Service Registry. DynaTree currently supports JavaBean, local Java Object, Web Service and RMI call as service types, but can be expanded accordingly.

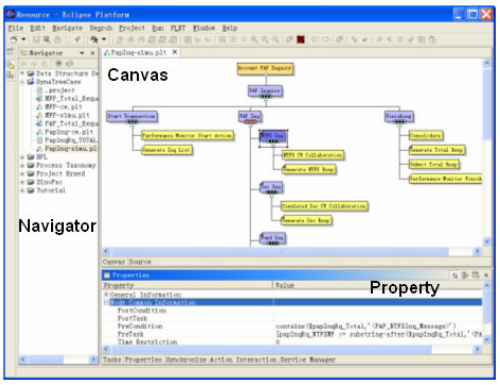


Figure 4. DynaTree Editor

As depicted in Figure 5, Service Manager maintains all the registered services. The detailed specifications of service can be defined through Service Manger related functions, which include: service name, service type, carrier package and class, method and corresponding parameters, etc.

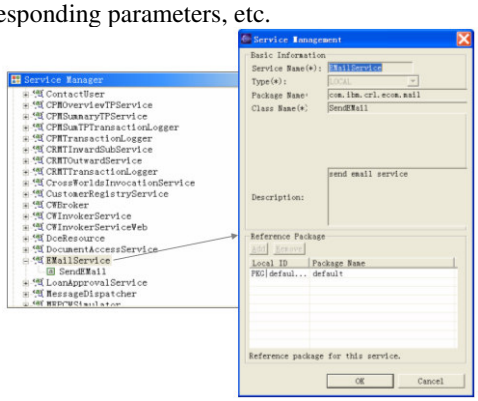


Figure 5. Service Manager for Component Registration

DynaTree Simulator provides user the verification capability for the designed DynaTree script before deploying it onto runtime engine. The DynaTree simulator imitates DynaTree engine in the Eclipse debugging environment to validate logic correctness.

Deployment Tool enables user to deploy designed assets into the engine. The asset of DynaTree script and its associated invocation message type will be deployed to DynaTree Library; the asset of service definition and implementation package will be deployed to Service Registry on the runtime environment.

After a DynaTree script and related services have been successfully deployed to the runtime environment, it can be invoked by a corresponding request message. As illustrated in Figure 3, the Invocation Broker will pickup the message and query the DynaTree Library to bind the message to a specific DynaTree script, then the script would be loaded on the DynaTree Engine. The engine instantiates a script instance by feeding the

request message. The DynaTree Engine will then navigate the tree according to defined traverse algorithm, perform variable manipulating, condition processing, and service invoking so as to achieve the component composition objectives. The software components hosted in the Underlined Components Framework can be invoked through Service Invocation Broker according to service definition registered in the Service Registry.

Admin Console provides the user interface to administrate DynaTree runtime environment. DynaTree script instance, request message queue can be monitored and managed; the detailed information in DynaTree Library and Service Registry can also be queried and listed.

4. The Case Study

In this section, we will demonstrate the design process by using the ALT model, and further discuss and validate the benefit that the method brings. We use the Travel Planning solution as the case.

Generally, there are several steps for a travel preparation: submit travel request, purchase the plan ticket, reserve hotel room and reserve car. Usually the individual components (e.g., airline ticket purchase system) have already been available through the Internet or intranet. So the key requirement is to enable the automation of the travel preparation by compositing those functional components. And based on that, more advanced issues will take into consideration.

4.1 The Design Process

(1) Business Customer (or Analyzer) creates a process called TravelPlan process. And in the Travel Plan process the business task is "make travel preparation", so the Customer added the first level of node and name it as the "Make Travel Preparation"

(2) The Business Customer goes on the decompose the "Make Travel Preparation" into a sequence of four sub tasks, "Request for travel"(request internal approval), "Purchase Plane Ticket", "Reserve Hotel", "Reserve Car". And follow the same approach, the Business Customer breakdown the big task into a set of "atomic" business tasks he can understand (e.g., "Query Ticket Price", "Query Room Charge").

(3) IT Developer continue the decomposition process by growing each "atomic" business tasks into a sub-tree of executable tasks, each task on the leaf node is an operation to a component, e.g., "query AE Airline Price" can be an query of the ticket price from the AE Airlines' web services.

(4) IT Developer further defines and revises the composition logic of the whole process.

(5) IT Developer further binds the leaf nodes with components, e.g., the "submit Travel Request" may be an EJB invocation to the "Travel Management System", and the "Book Room" may be an SOAP invocation to the Hotel's online reservation portal. And also the IT Developer will add more execution artifact, e.g., variable, expression, etc.

(6) IT Developer validates the designed abstract logic tree, verify with Business Customer on the requirement, and make necessary simulation, then deploy it for runtime automation.

4.2 Benefit Gain from the ALT Model

From the depiction of the design process, we can see that the ALT model enables both the Business people and IT people collaboratively design composition solution by following the same design principle. And they gradually decompose a complex business tasks into a set of component operations with composition logic. The ALT model simplifies the design process and makes it very easy for both business and IT people because it matches the common problem solving thinking. The design result, as shown in Figure 7, is clear-cut and very easy to understand for both business and IT people. In another words, it is an excellent view as the requirement communication/verification platform for business people and IT staff.

Beside the design aspect depicted, the ALT model brings more benefit on:

(1) Flexible process navigation. The tree-based presentation mode enables the separation-of-concern navigation. Business people can navigate the high-level process of the solution by shrinking the detailed node, E.g., the "Reserve Car" in Figure 7; IT architect can expand all node to have global view of composition logic; and IT developer responsible for travel management system need only focus on the "Request for Travel" sub-tree.

(2) Ease maintenance. For example, when the customer want to adapt the process to reserve the car before reserve hotel, it only takes only a single step to drag the "Reserve Car" node ahead to the "Reserve Hotel" node, instead of fussy and fallible steps in other composition model.

(3) Potential advantage in business transaction support. Though not implemented, the ALT model has advantage in identifying business transaction scope, even for sophisticated ones: a single tree node can indicate a standalone transaction boundary. And if both "Request for Travel" and "Make Travel Preparation" is marked as transactions, nested transaction is identified.

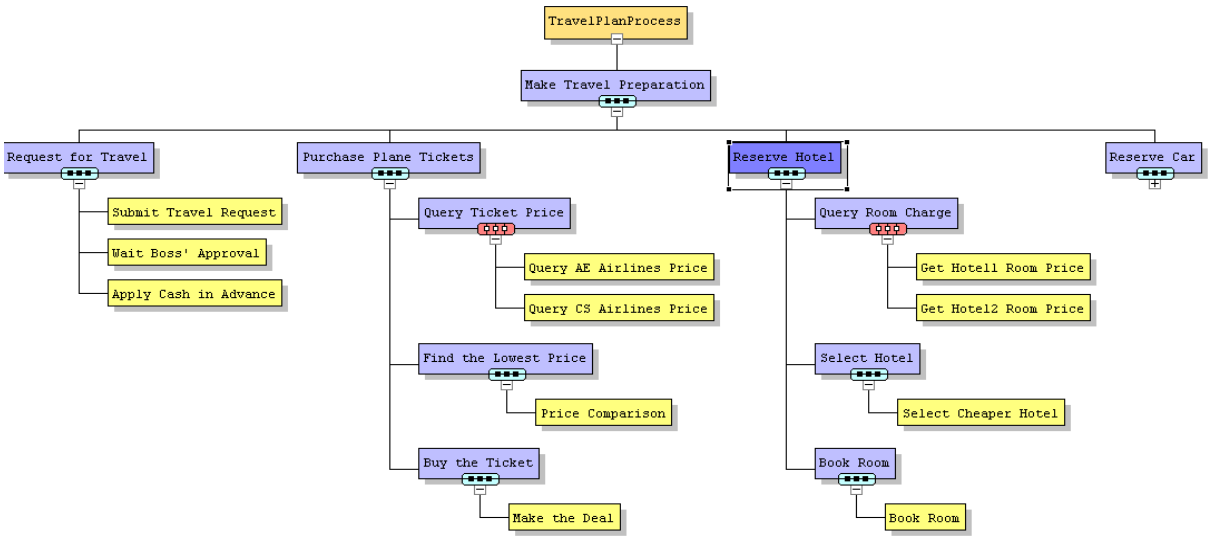


Figure 7. Travel Plan Solution Composition Design

5. Summary

ALT is devised based on divide-and-conquer strategy to solve the component composition problem. In this paper, we introduce the composition logic modeling capability of ALT and its detailed design. DynaTree framework has been developed to support the ALT design, simulation, automation, and management. A travel plan solution case is presented to demonstrate its advantage. And the technology approach has been successfully verified by some real customer engagements. ALT is designed as a platform independent mechanism. A pluggable and flexible framework, DynaTree, is provided to support different component implementation technologies, including web service, EJB, Java. All of these characteristics make Abstract Logic Tree and DynaTree framework can be widely used in many different component composition scenarios.

As the Abstract Logic Tree gives a good presentation in a tree structure for the composition logic of solution components, additional presentations are usually needed to illustrate the solution architecture based on a set of components, such as the collaboration and interaction relationships diagrams solution components, we will further explore the linkage and transformation between ALT and solution architecture related diagrams in our future work.

References

[1]Brownsword, L. Oberndorf, T. "Developing new processes for COTS-Based System". CA Sledge-Software, IEEE,2000.

[2]B Meyer, C Mingins, "Component-based development: from buzz to spark". Computer Publication Date: Jul 1999 On page(s): 35-37 Volume: 32, Issue: 7.

[3]Naiyana Tansalarak, Kajal T.Claypool, "XCompose:An XML-Based Component Composition Framework". Third Int. Workshop on Composition Languages, 17th European, 2003.

[4]Roel Wuyts, Stephane Ducasse, "Composition Languages for Black-Box Components". Language Mechanisms for Programming Software Components, 2001

[5]Ludger Martin, "Visual Composition of Components". The 6th IASTED International Conference Software Engineering, 2002.

[6]IBM Developerworks, "Business Process Execution Language for Web Services". http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[7] Tuyet Le-Ann, Jorge Villalobos, "Multi-Level Composition for Software Federations". Electronic Notes in Theoretical Computer Science, 2003

[8] M. Aoyama, Component-Based Software Engineering: Can It Change the Way of Software Development? In Proceedings of the International Conference on Software Engineering, Volume 11, April. 1998.

[9]H. Mili, F. Mili, and A. Mili. Reusing Software: Issues and Research Directions, IEEE Transactions on Software Engineering, Vol. 21, No. 6, pp528-561, June. 1995.

[10] W.M.P. van der Aalst, A.H.M. ter Hofstede, A. Kiepuszewski, and A.P. Barros. Advanced workflow patterns. In 7th International Conference on CooperativeInformation Systems (CoopIS 2000), volume 1901 of Lecture Notes in Computer Science, pages 18–29. Springer-Verlag, Berlin, 2000.

[11] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. Distributed and Parallel Databases 14(3), pages 5–51, 2003.

Container-Based Component Deployment: A Case-Study

Nigamanth Sridhar, Cleveland State University, n.sridhar1@csuohio.edu
Jason O. Hallstrom, Clemson University, jasonoh@cs.clemson.edu
Paolo A.G. Sivilotti, Ohio State University, paolo@cse.ohio-state.edu

Abstract

A component container*, similar to those used to host Enterprise Java Beans, is a runtime environment that manages the execution of components. Component containers support a separation of application concerns from architectural and maintenance concerns. In this paper, we examine the deployment models supported by* DRSS*, a container architecture for Microsoft's .NET Framework that provides support for dynamic component deployment. We illustrate the principal advantages of these deployment models in the context of a case study. We describe the evolution of a distributed conflict resolution protocol, with an emphasis on evolving network performance, failure-locality, and visualization support. All the components of the system described have been implemented in C# for the .NET architecture.*

1 Introduction

A component container is a runtime environment that manages the execution of components. Container-hosted components are subject to behavioral transformations that extend the functionality provided by the components in isolation. While a container may impose architectural constraints on the components it hosts, the functionality extensions provided by the container can often be achieved without the hosted components having been explicitly designed to support them. For example, a component can be imbued with persistence simply by being deployed in its hosting container.

Perhaps the most well-known example of a component container is the J2EE container from Sun Microsystems that is used as a hosting environment for Enterprise JavaBeans components [11]. An EJB container is just one example of a more abstract concept. Another implementation of this concept is described in [7], which presents the design and implementation of the *Dynamic Reconfiguration Sub-System (DRSS)* container for Microsoft's .NET Framework. While similar in spirit to an EJB container, DRSS provides greater e xibility with respect to component deployment.

In this paper we describe the DRSS deployment model, and illustrate the utility of the model in the context of a case study documenting a resource allocation component that we have built for DRSS. We focus on the following deployment advantages offered by DRSS:

- Dynamic component deployment
- Dynamic service deployment
- Flexible scoping of behavioral transformations

The rest of this paper is organized as follows. We give a brief overview of component containers and some background in the area, along with an introduction to container services in Section 2. In Section 3, we outline the deployment model used by DRSS, and some of the advantages of using this deployment model. We also illustrate each of these advantages by way of a case study. We conclude in Section 4.

2 Component Containers

A component container behaves like a barrier, protecting the enclosed contents from potentially damaging interactions with a harsh environment. In this case, the harsh environment consists of client components that depend on their container-hosted counterparts to implement their operations. The container monitors interactions between hosted components and their clients, and only allows "safe" interactions. The container also presents a natural solution to the problem of cross-cutting concerns — such concerns are implemented as *container services*.

Consider, for example, a set of components that require authentication and access control. If the component designers had to implement these services at a per-component level, service evolution would be dif cult. And in the context of continually evolving attack models, security services need to evolve continuously. However, if these services are implemented in the container, maintenance and evolution is more localized. Containers thus provide a scalable approach to *separation of concerns* [12] in component-based software.

All access to hosted components is through the container. When a client requires access to a component within the container, it must request a reference to the desired component using the container interface. In response, the container returns a proxy [6] that appears to the client as the component itself. From this point on, the client interacts only with the proxy. The initiating client is unaware of the additional level of indirection, which is maintained throughout the lifetime of the component. The invocation model (similar to that used by commercial middleware platforms like Java RMI and CORBA) is shown in Figure 1.

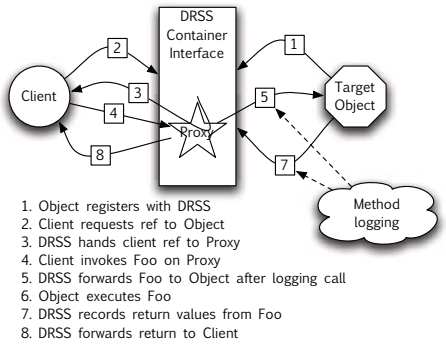


Figure 1. DRSS Invocation Model

The additional level of indirection introduced by a component container is not incidental to the architecture. In fact, the additional level of indirection is precisely what enables the container to transparently transform component functionality. When a component container receives a method call, it need not deliver the call immediately. Instead, the container can inject additional services in the path of the invocation before it is *eventually* delivered to the target. It might also inject additional services on the return-trip. The additional services supplied by a container are often modularized as reusable "*interceptor*" components.

The canonical example of a container-supplied service is invocation logging. By simply recording the method calls it receives, a container can transform the behavior of its hosted components to include support for invocation logging. The implementations of the hosted components are unaffected. From a client's perspective, the view of the component is altered by the behavior that the container provides. Another example is discussed in [13], where container mediation is used to implement invariant-based state monitoring for J2EE applications.

Regardless of the functionality it provides, each DRSS interceptor supplies an identical interface. Although the actual methods vary from one implementation to another, conceptually, each interceptor provides methods for processing method calls, and method responses. This interface equivalence makes them easily composable as *interceptor chains*. An interceptor chain is composed of individual interceptors arranged in sequence. By using such chains, the container can inject cross-cutting services in the path of component collaborations without compromising encapsulation.

3 DRSS-Based Component Deployment

We restrict our attention to the deployment details; see [7] for a full treatment of the DRSS design. The deployment options that DRSS supports are presented in the context of a case study.

Roadmap. We present a *utility component* (named Philosopher) that DRSS-hosted applications can use to manage their resource allocation needs. The component uses a solution to the dining philosophers problem [3] as the con ict resolution policy.

We demonstrate dynamic component deployment by rst starting an application with one implementation of the Philosopher component — the Asynchronous Doorway implementation [5]. This algorithm has good *failure locality* (a measure of the robustness of an algorithm in the presence of faults), but performs poorly in terms of the number of messages exchanged, and time required for a process to begin eating. We use *dynamic module substitution* to replace this implementation with a new one — the Hygienic implementation [3]. Next, we show how the failure locality of this algorithm can be improved by dynamically deploying a *fault localization service*. Finally, we illustrate how we can dynamically control the scope of services injected into the container architecture by deploying a *visualization service* to monitor the Philosopher component exclusively, without affecting the application layer that makes use of the Philosopher component.

3.1 Dynamic Component Deployment

As part of its interface, the DRSS container provides management methods for controlling the dynamic deployment and removal of hosted components. This interface is exposed by the container to remote processes, enabling third-party management of the deployment process.

To deploy a component, the component must be created by the container. The client passes the type information corresponding to the component to be deployed, as well as the arguments to be passed to the component's constructor. The container constructs an instance of the type requested, and registers the instance under a unique ID that is returned to the client. This ID can later be used to remove the instance from the container, assuming there are no outstanding proxies associated with the instance.

DRSS additionally provides support for *dynamic module substitution*. That is, DRSS supports substituting new component implementations for existing implementations dynamically, without disrupting the services supplied to component clients. The container implements this functionality by temporarily blocking access to the instance to be replaced. Such blocking is possible because of the permanent level of indirection maintained by the container. It is important to note, however, that the client is unable to distinguish this delay from the normal delay introduced during method execution.

If the component being replaced is stateful, the container will then transfer the abstract state of the existing implementation to the new implementation, using state transfer operations provided by the components. Module replacement in DRSS leverages work described in [1, 10].

Case study example. The substitution of the Philosopher implementation involves the following stages [10]:

1. **Initiation.** The replacement must be initiated. DRSS supports remote message delivery to components running inside the container; an external application can prompt the recon guration.
2. **Module Rebinding.** The target module is bound in place of the existing module. DRSS supports dynamic binding and re-binding of module implementations.
3. **Instance Rebinding.** For every object that the old module created in the application layer, a new object from the new module must be created. DRSS uses the .NET runtime to dynamically create new objects.
4. **State Migration.** The state of the application must be preserved across the recon guration process. DRSS can perform automatic state migration if the module being replaced supports externalization and internalization of state [10].

Throughout the module replacement process, *module integrity* must be preserved. To ensure that application behavior is not altered, we use an adaptation of the algorithm described in [10]. During the recon guration, all requests to eat are deferred by DRSS. Once the module has been properly replaced, all deferred requests are handled appropriately.

These steps can be used to "*hot-swap*" the Philosopher implementation without having to shutdown the application layer. The application layer is unaffected with respect to correctness. The application may (and likely will) be able to observe a degradation of performance, but this degradation is transient, and is indistinguishable from a scenario where the other clients eat for a long time.

3.2 Dynamic Service Deployment

In commercial architectures like J2EE, the set of services supplied to container-hosted components is x ed at container initialization. This static deployment precludes the dynamic adaptation of the services supplied by the container. So, for example, if increased security risks are detected, it is impossible to deploy additional container-based security services without bringing the container down for maintenance. This is, of course, an inappropriate strategy for systems with high-availability requirements.

To overcome this problem, the DRSS container provides methods as part of its management interface for dynamically deploying and removing container-supplied services. This is made possible by the fact that DRSS relies on interceptors to modularize the services that it supplies to hosted components. These interceptors are maintained as chains, and the container interface allows remote processes to manage the interceptors contained in these chains. In particular, given a particular chain, the management interface provides methods for inserting and removing interceptors.

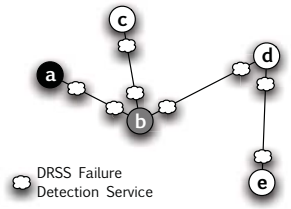


Figure 2. Fault Localization Using DRSS

Every method invocation from a client to a hosted component is processed through the interceptor chains within the hosting container. Thus, dynamically modifying the set of interceptor components in a chain modi es the services injected by the container.

Case study example. How does the failure locality of the Hygienic algorithm compare to the Doorway algorithm? The Hygienic algorithm has a failure locality of n, where n is the size of the con ict resolution graph. This is significantly worse than the optimal gure of 2 provided by the Doorway algorithm. How do we address this issue?

In [8], Pike and Sivilotti propose a transformation of the Hygienic algorithm that provides failure locality 1. This transformation uses an *eventually perfect* ($\diamond\mathcal{P}$) failure detector [2]. We implement this $\diamond\mathcal{P}$-based fault-containment service in DRSS. The service is implemented using a set of interceptors, one at each dining philosopher node, that collectively perform the function of the $\diamond\mathcal{P}$ failure detector. Any message that a philosopher node sends or receives is monitored by the interceptor associated with it. Based on the observations that it makes about whether neighbors are alive or crashed, the interceptor may make modi cations to the messages. These modi cations reduce the failure locality of the Hygienic algorithm to 1.

See Figure 2. At each node, the failure detector service is deployed as an interceptor; each neighbor is monitored. If a process b "suspects" its neighbor a has failed, it sacri ces local progress, and yields forks to all other neighbors regardless of their priority. This way, the effects of the failure are restricted to the *1-neighborhood* of the failed node. See [8] for details of the transformation.

3.3 Flexible Scoping

Commercial container architectures such as J2EE, and academic container architectures such as E"Speak [9], provide support for interceptor composition via interceptor chains. However, these containers support only a single interceptor chain that is shared among all container-hosted components. As a consequence, these containers inject the same services into every hosted component. They do not provide a mechanism for scoping the effects of interceptor-based transformations. Any change to the global interceptor

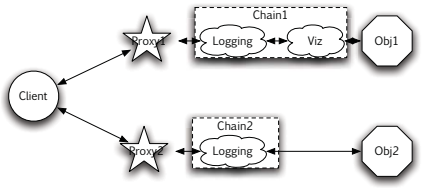


Figure 3. Flexible Scoping in DRSS

chain results in a behavioral transformation that affects all hosted components.

DRSS also supports interceptor composition via interceptor chains. However, the container provides methods for managing *multiple* chains. Using the management interface, it is possible, for example, to bind one set of container-hosted components to chain A, and to bind another set of container-hosted components to chain B. When chain A is modi ed, the resulting behavioral effects in uence only components in the rst group. Similarly, changes to chain B effect only components in the second group. For example, in Figure 3, method calls to Obj1 are processed through Chain1 (*i.e.,*logging and visualization), while those to Obj2 are sent through Chain2 (*i.e.,*logging only).

Case study example. The visualization service for the Philosopher component monitors the entire con ict resolution graph and shows the movement of messages and forks among the philosopher nodes. The visualization service is dynamically deployed to one of the containers hosting a Philosopher node; this serves as a bootstrap mechanism. The service then *diffuses* through the network until it is running at all of the nodes. In effect, the service is an *eventually stabilizing visualization service*.

Once all of the nodes have been discovered, the service keeps track of the local state of each Philosopher node — *thinking, hungry, eating*. Any change in (*i*) the local state of a node, (*ii*) the location of the forks in the con ict graph, and (*iii*) the partial ordering among nodes imposed by their priorities, is re ected within the visualization (after a short delay).

As with the fault localization service, the visualization service is deployed as a set of interceptors. Each Philosopher node is monitored by an interceptor, and each interceptor is responsible for updating the state of its associated Philosopher object within the visualization. The interceptor also examines messages passing through its respective node, and updates the status of the forks and the directions of the arrows on the edges between neighboring nodes in the con ict graph.

The visualization service only applies to the Philosopher component, not to the application layer using the component. So although the two may be running within the same container (and sharing some services), the visualization service is only included within the interceptor chain to which the Philosopher is subscribed.

4 Conclusions

In this paper, we described the deployment model supported by the Dynamic Recon guration Sub-System (DRSS) — a dynamic container architecture for the Microsoft .NET Framework. DRSS affords a e xible and extensible deployment model. The three main advantages of the deployment model are:

- Dynamic component deployment
- Dynamic service deployment
- Flexible scoping of deployed services

We illustrated these advantages in the context of a small case study involving a utility component that provides resource allocation services to DRSS-hosted applications. We demonstrated the capability of dynamic module substitution, replacing one dining philosophers algorithm with another based on application performance needs. We also demonstrated dynamic deployment of behavioral transformations by way of container services, strengthening the fault containment capabilities of the resource allocation component. Finally, we demonstrated support for selective scoping of container services to speci c components hosted within a single container.

References

[1] Castro M. and Liskov B. Practical byzantine fault tolerance and proactive recovery. *ACM ToCS*, 20(4):398–461, 2002.

[2] Chandra T. D. and Toueg S. Unreliable failure detectors for reliable distributed systems. *JACM*, 43(2):225–267, 1996.

[3] Chandy K. M. and Misra J. The drinking philosophers problem. *ACM TOPLAS*, 6(4):632–646, 1984.

[4] Chandy K. M. and Misra J. *Parallel Program Design: A Foundation*. Addison-Wesley, Reading, MA, USA, 1988.

[5] Choy M and Singh A. K. Localizing failures in distributed synchronization:. *IEEE TPDS*, 7(7):705–716, 1996.

[6] Gamma E., et al. *Design Patterns*. Addison Wesley, 1995.

[7] Hallstrom J. O., Leal W. M., and Arora A. Scalable evolution of highly-available systems. *IEICE/IEEE Joint Special Issue on Assurance Systems and Networks*, E86-D(10):2154–2166, 2003.

[8] Pike S. M. and Sivilotti P. A. G. Dining philosophers with crash locality 1. In *ICDCS 2004*, pages 22–29. IEEE, 2004.

[9] Pruyne J. Enabling QoS via interception in middleware. Tech Report HPL-2000-29, HP Laboratories, February 2000.

[10] Sridhar N., Pike S. M., and Weide B. W. Dynamic module replacement in distributed protocols. In *ICDCS03*, p 620–627, May 2003.

[11] Sun Microsystems. J2EE 1.3 speci cation. July 2001.

[12] Tarr P., et al. N degrees of separation: multi-dimensional separation of concerns. In *ICSE 1999*, pages 107–119. 1999.

[13] Vecellio G.J. and Thomas W. M. Infrastructure support for predictable policy enforcement. In *CBSE-6*, May 2003.

Interaction Partnering Criteria for COTS Components

M. Kelkar M. Smith R. Gamble
Department of Mathematical and Computer Sciences
University of Tulsa
Tulsa, OK 74104 USA
gamble@utulsa.edu

Abstract

Commercial-off-the-Shelf (COTS) software provides a choice of products to streamline enterprise applications. COTS software integration can introduce security vulnerabilities due to mismatches between security constraints coupled with inadequate knowledge of interaction requirements. Though a component can be validated against its stand-alone functional and security requirements, two aspects of the validation for its integration are missing. First, no straightforward process exists to guide the developer in identifying integration-induced security risks. Second, interaction properties contributing security risks are not part of COTS product evaluation. In the former case, a process is needed to take advantage of selection criteria. In the latter case, interaction partnering criteria - criteria indicating how closely related the security constraints of two potentially communicating components are - must be defined. We examine these issues by defining initial interaction partnering criteria and exploring there use in a security profile for COTS components.

1. Introduction

Composing COTS products into larger systems provides a cost-effective and streamlined solution to increase functionality. However, COTS components do not interact seamlessly. Their insertion into application integrations leads to both functional (communication, data, and control exchange) [1] and non-functional (performance, reliability, and security) [2] interoperability problems.

COTS component security characterization provides an initial foundation for expressing necessary security properties whose mismatched values can cause security-based interoperability conflicts [3]. Template development for uniform characteristic representation is still needed if comparisons are to yield understandable, deterministic results. Moreover, a secure composition of components can be guaranteed only if the template covers a broad range of security properties related to many kinds of threats.

Defining security properties and mechanisms of COTS components in a uniform manner is a necessary step toward building a set of criteria to evaluate individual components interacting within an integrated system [4, 5]. Research on evaluating security properties deals mainly with product offerings and has not yet addressed evaluating products for interaction capabilities in an enterprise system while complying with security constraints. To do this, characteristics should be acquired from reliable sources, assessed for relevance, and organized for comparison.

Our objective is to develop a methodology to determine the best interaction partners among COTS components. We define *interaction partnering* among COTS components with respect to security as *the uninhibited exchange of data and control information between two components that complies with defined security constraints and requires minimal intervention of external processing code.* Interaction partnering analysis helps integrators limit adaptation mechanisms required for component integration.

The interaction partnering concept reduces security vulnerabilities by bringing direct and induced security property mismatches to the forefront of the design. This is achieved by generating a *security profile* - a uniform representation of component security properties whose comparison among interacting components yields informal metrics related to the amount of external processing needed to adapt each component to maintain security compliance in an integrated system. Through these metrics, assessments can be made to determine which components make better interaction partners. If interaction partnering is conditional, the security mismatches due to conflicting policies are

addressed both locally at component level and globally at integrated system level.

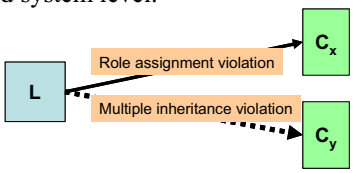


Figure 1: Interaction Partnering Challenge

For example, assume a company has a legacy component, L (Figure 1). They want a COTS component that complies with specific security criteria. The choice is between C_x and C_y. Closer scrutiny of C_x reveals that it establishes an inter-component role inheritance which contradicts the security policy of L, which for a user with role R, disallows access to a role hierarchically superior to role R. Thus, C_x allows a role assignment violation when interacting with legacy component L. C_y offers rule inheritance which allows a user to inherit access control inappropriately from two parent roles having conflicting privileges. This leads to multiple inheritance security violations. Though both policies conflict with L, C_x is a better interaction partner because minimal external processing will be needed to restrict the inter-component role inheritance.

2. Related Work

Component-based software engineering approaches provide definitions of component properties at the architectural level [1, 6]. The defined interface properties mainly consider the functional aspects of component interactions. However, conflicts are only minimally addressed when interacting component definitions do not match.

Component interface signatures incorporate properties, operations, and events as elements observable from outside the component along with the constraints and relationships between these elements [7, 8]. Direct replacement can occur by components with the same signature. Such approaches are based on functional properties, and although non-functional properties like security are addressed, the full depth and breadth of non-functional property expression is still difficult to achieve.

Software safety research examines COTS products with respect to integration, certification, and maintenance, given a defined interaction between components within an application [9]. This approach yields a COTS component selection process involving system criticality analysis to detect hazards and identify COTS component failure modes. A *safety contract* – a safety agreement between selected COTS components and the system – is used for system safety certification. Safety contracts have analogous goals to security profiling for COTS.

Certain security specifications for COTS components are defined uniformly to evaluate individual COTS components [4, 5, 10]. However, security properties for an integrated system with COTS software are not considered. Where approaches provide samples of security mismatches by two interacting components, no definitive process for detecting mismatches is uniformly applied.

For security certification of an integrated COTS system, observable security properties can be defined at the atomic, composition, and global levels using logic programming, developing contracts between components for compatibility checking [3]. This provides a framework for composers to test the impact of security on component functionality by exposing trust-related attributes of one component to be captured by another component. A *Compositional security Contract* (CsC) performs the checks. The objectives of CsC research closely relates to ours. The difference is that CsCs are based solely on Common Criteria [11] whereas our approach takes a broader view of security properties. If CsC confirms that the properties are matched, a viable solution exists. However, when mismatch occurs, the CsC and candidate component are discarded instead of trying to find an integration solution that can be certified.

3. Distillation Process

Because of the number and range of security properties (e.g., confidentiality, integrity, availability, and non-repudiation), we approach the creation of security profiles for COTS components by defining a reusable process with which we accumulate, categorize, and assess a single property group. In this case, we examine properties related to access control conflicts. The reusable process, DIPC (Distilling Interaction Partnering Criteria) is shown in Figure 2. The goal for DIPC is to command a level of abstraction among properties that (1) have direct impact on design information, (2) can be applied to and valued by COTS products, and (3) maintain a high degree of reusability across different products and security policies.

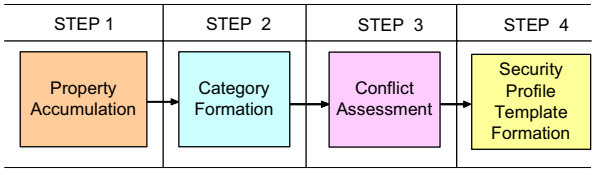


Figure 2: Distilling Interaction Partnering Criteria

The remainder of Section 3 focuses on the first two steps of DIPC. We discuss evaluation and conflict assessment of Step 3 in Section 4 and the security profile template formation of Step 4 in Section 5.

3.1. Property Accumulation

The easiest, but perhaps most tedious, way to acquire properties is through the perusal of published sources that address the understanding, coordinating, certifying, verifying, and assessing of security properties. Some predominant sources include government and military documents such as Common Criteria [11], DITSCAP [12], and the NIST Information System's certification document [13], all of which mention the need to assess critical security features for correct and complete implementation of an integrated system. These documents are primarily directives for the integrator and certifier to review the interfaces between components and determine their security compliance. Unfortunately, they lack explanation for compliance checking, ascertaining contributing properties, and identifying the conflicts that cause non-compliance.

The best sources for property accumulation are published accounts of security property expression, use, and analysis. Limiting our initial DIPC accumulation needs to access control properties, we researched documents that directly address policy conflicts (e.g., [14]), emerging vulnerabilities from system interactions (e.g., [15]), mismatches of access control use and intent (e.g., [16]), and references to COTS components and their access control constraints (e.g., [17]). A few publications directly consider access control as defined for a component [3, 18, 19].

Another DIPC task is to filter property information so that definitions are at the same level of abstraction. The key to filtering is to immediately establish a classification scheme starting with the first property and grouping properties as they are examined. This scheme may be refined multiple times. However, each refinement usually induces a smaller set of properties that are more tightly coupled within the same classification or are more distinct across multiple classifications. For example, early passes at classifying access control properties put them in a specific access control model (e.g., RBAC). We then refined that classification according to the formal language representation used (e.g., role-based) [3, 20-23]

Successive refinements resolve generalized definitions to accommodate various access control models and their respective languages. For example, authentication tokens can be defined differently on a component level than on the access control model level. A Web Service provider component, may call the information required from the client a token (identity-based) [16] while access control models may refer to authentication tokens as the actual user name and password pair (identity-based) or associated encryption key (credential-based). We resolve the semantic problem by defining authentication token as authorization data and use identity-based and credential-based as values for more specific information (Table 1).

Many property statements are straightforward. Exceptions require matching statements to known properties or waiting until a new property emerges for further match. For instance, *"requires agreement on protocol, syntax and semantics for each piece of shared authorization data* [14]" brings authorization data to the forefront as an entity to consider. Less obvious is the authorization data entity in the following statement: "*There might be a potential conflict regarding the identification a Web service provider requires to provide the service, and the identification a Web service client is willing to reveal* [17]. *"*

Redundancy can be problematic unless only standard definitions are used, if they exist [1]. Redundancy indicates importance, even if there is some conflict among the property definitions. The DIPC process regards and eliminates redundancy by (a) taking the union of definitions and values and assigning them to a single property or (b) partitioning the conflicting pieces of the definitions across multiple, non-redundant properties. For example, temporal security properties have overlapping definitions across publications, some of which are called session based properties. These we group under the property *temporal* (Table 2).

The granularity issue requires rephrasing low-level properties to fit with more coarse-grained descriptions. For example, XML based access control research defines access as subjects having various rights on different parts of an XML document [14, 15, 24]. Some Web service papers define access as the ability to discover and invoke a web service and have its WSDL document be based on WS-security standards [16, 17, 24, 25]. Since a privilege level must be defined to see which domain is given priority, these properties are represented by *Privilege Level* (Table 3).

Once a set of properties is established, we proceed to place them in more concrete categories. We describe these categories in the next section.

3.2 Category Formation

The categorization of accumulated properties provides more substance to relationships among properties. For a property to be placed into a category, it must have distinct values that can be expressed at

relatively the same granularity as the values of other properties in that category. This better organizes them into a security profile, determine dependencies in and among categories, and facilitate conflict resolution. It is also the first step in reusing DIPC because as more properties become relevant, their placement in a category will inherit the organizational distinctions that have already been analyzed.

Because we used an opportunistic classification scheme in the first DIPC accumulation step, we must not rely solely on that for categorization. Rather, we ground DIPC in the application framework of integrated systems relying on COTS components. In this context, access control is generally defined as allowing or disallowing components to perform a specified operation on other components based on the security attributes of those components [14, 16]. This objective strategy, along with the examination of the resulting properties, yields an initial three categories: (1) exchanging authorization data, (2) controlling access to the components, and (3) strategically combining the policies which define access data and control in a system. We validate the worthiness of the categories by checking that the majority of accumulated properties can align with only one. Different valuations of certain properties may be needed for complete alignment. In one case, a property is further divided across two categories.

The first category is directly related to data issues with the properties that fall under *authorization data* are shown in Table 1. The second category focuses on control issues and is appropriately called *access control.* These properties are related to functional restrictions to information. Thus, in this category we have the access inheritance, separation of duty (SOD), cardinality, and temporal constraints (Table 2). This category is formed by grouping the properties which are based on user-based and role-based access control models. Properties arising due to implementation of other access control models that are either independent or extensions of these basic models are not in this table.

The properties under *access control policy combining strategies* (Table 3) are based on the precedence of action defined in the policy, comprehensiveness of the access policy definition, configuration of the component, order of the access policy, or privilege level defined in the policy. Usually components can have multiple policy combining strategies which are stated according to priority. The precedence type of policy combining strategy is usually given first preference. Thus, if one of the interacting components allows access and the other one denies access, the decision of which policy overrides depends on the precedence rule defined. If both components allow or deny access but one defines policy in more detailed form with some exceptions in it, then the strategy next-in-line is applied. The decision application for this strategy is based on the type of policies that need to be combined.

4. Evaluation and Conflict Assessment

Once the categories are established, the goal for evaluation and assessment is two-fold. First, we interpret the issues that arise from mismatched values in a single category. Then we extend the comparisons across tables. Methodological advancement of this goal needs further empirical testing to validate the true presence of the conflicts that emerge. However, the initial categories form only a foundation for three major types of conflicts between interacting components. These conflicts are interoperability conflicts, security conflicts, and discrepancy conflicts.

4.1 Interoperability Conflicts

Interoperability conflicts are considered to be high-risk and occur when access can be defined for the individual component but not in a global context. Components are unable to interact if mismatches occur for properties within the Authorization Data Category (Table 1), which produces a high level of security vulnerability. Resolving these conflicts requires an extensive amount of glue code and wrappers to be evaluated and inserted into the system. For example, two components using identity-based and credential-based access control respectively have no way to communicate authentication tokens correctly without using some form of escrow service. Without resolving this issue, access remains undefined in the global context. Because access remains undefined, the security properties in this category carry more weight than properties under other categories.

4.2 Security Conflicts

A security conflict occurs when access is allowed or denied in a single domain and is similarly not allowed or denied across all domains in the system. This type of conflict is more likely to happen when one or more access control properties exist (Table 2). These values are not evaluated on a comparison basis like the interoperability conflicts in Table 1. The existence of these properties shows the need for more in-depth evaluation techniques. Security conflicts are considered less severe than interoperability conflicts because the resolution strategies are generally less extensive. However, the level of possible security risk involved remains the same.

Table 1: Authorization Data Category

PROPERTY	Authorization Data Type	Authorization Data Interactivity	Interpretation of Authorization Data
DEFINITION	Format used to produce or accept authorization token	Number of times a component produces and/or accepts authorization data	How a component responds to the received authorization data and what it expects
POSSIBLE VALUES	**None**	**None**	**None**
	Identity-based: Something the component *knows* (passwords, mother's maiden name, etc)	**Single Sign-On:** Credentials are accepted or produced only one time - sign on once for multiple systems	**Abduction** : Replies back and expects reply if the credentials are valid and access is allowed
	Credential-based: Something the component *has* (third party item, encryption keys, etc)	**Multi-Step Sign On:** Challenge-Response - additional credentials may be required during authentication process	**Deduction**: Replies back and expects a reply if the access is denied or failed

Table 2: Access Control Category

PROPERTY	Inheritance	Separation of Duty	Cardinality	Temporal
DEFINITION	Ordering of access control policy elements.	Two conflicting entities can not be simultaneously assigned to a particular entity	Numerical restrictions on access control entity assignment	Time for which access is defined (session based, time dependent)
POSSIBLE VALUES	**None**	**None**	**None**	**None**
	User-based: Seniority levels are defined for users in an access control policy	**User Based:** Two conflicting users cannot access a role simultaneously	**User-based:** Number of users that can be assigned to a certain role/rule	**User-based:** Time that a user can be assigned or activate a particular role or rule
	Role-based: Roles defined in an access control policy are hierarchical	**Role Based:** A user can not access two conflicting roles simultaneously	**Role-based**: Number of roles which can be assigned to a particular user	**Role-based**: Time for which a role can be assigned to a particular user

Table 3: Access Policy Combining Category

PROPERTY	Precedence	Granularity	Configuration	Applicability	Privilege Level
DEFINITION	What action defined in a set of policies override	Policies are given priority based on completeness or comprehensiveness	Master or slave component policies override the other	Access control statements are evaluated in the order they are found	The policy having least or most privilege is given priority
POSSIBLE VALUES	**None**	**None**	**None**	**None**	**None**
	Deny: If any statement denies access, access is denied	**Fine-grained**: More detailed policy takes priority over general policy	**Master:** Master component policies override slave	**First Applicable:** Evaluation in First come first serve basis	**Least Privilege:** The policy having least privilege is given priority
	Permit: If any statement permits access, access is permitted	**Course-grained**: Properties with less detailed information gets priority	**Slave:** Slave component policies override master	**Only One applicable:** If more than one policy is found to be applicable, then the result is indeterminate	**Most Privilege:** The policy having most privilege is given priority

Proper mapping between users and roles can alleviate the need to use bulky glue code. Such accurate mapping between components becomes more challenging with each additional access control property used by the component. All of the properties in this category allow a component to have multiple implementations of the property, leading to an added layer of possible conflict evaluation complexity.

4.3 Discrepancy Conflicts

Discrepancy conflicts encapsulate conflicts arising from the Access Policy Combining Category (Table 3). Even if mismatches occur, components will still be able to interact with each other, provided they have proper access definitions. Some access policies may remain unaddressed if conflicts arise.

For example, say a server (master) component is interacting with a client (slave) component where the client uses a least-privilege policy combining strategy and the server uses a most-privilege policy combining strategy. A discrepancy remains as to which strategy should be followed for combining policies—master/slave or most-/least- privilege. By redefining individual component policy statements or declaring a global combining strategy that trumps the component policies, the conflict might be resolved.

5. Security Profile Template Formation

Each security property discussed in the category tables forms initial set of interaction partnering criteria for a component. These criteria build the security profile for a component where the corresponding value for each property is stated. Security profiles of components can then be compared to detect mismatches in the partnering criteria, which lead to the above-mentioned conflicts. This helps integrators identify the type of conflicts caused by a pair of components attempting to interact with each other. Further refinement to the profile is part of ongoing research.

6. Conclusion

In this paper, we describe the initial phases of DIPC to determine relevant security interaction partnering criteria for COTS components. Through this process, we elevate access control properties to a common granularity of expression whose values can be easily established via the documented interface and security mechanisms of a COTS product. This helps bridge the gap between ad hoc implementation of integration solutions that may cause added security problems and the design of a security compliant integrated system. This is because the property definitions and values facilitate an evaluation of component interactions using a security profile to detect possible mismatches.

Acknowledgement: This work is supported in part by a US AFOSR award FA9550-05-1-0374.

7. References

[1] L. Davis, R. Gamble, and J. Payton, "The impact of Component Architectures on Interoperability," *Journal of Systems and Software*, 2002.

[2] C. M. Geisterfer and S. Ghosh, "Software Component Specification: A Study in Perspective of Component Selection and Reuse," Int'l Conf. on COTS based Software Systems, Florida, 2006.

[3] K. M. Khan and J. Han, "Deriving Systems Level Security Properties of Component Based Composite Systems," Australian Software Engineering Conf., 2005.

[4] W. J. Lloyd, "A Common Criteria Based Approach for COTs Component Selection," *J. Object Technology*, vol. 4, pp. 27-34, 2005.

[5] Q. Zhong and N. Edwards, "Security Control for COTS Components," *IEEE Computer*, 31(6):67-73, June 1998.

[6] L. Davis, et al., "A Notation for Problematic Architecture Interactions," European Software Eng. and ACM Foundations of Software Eng., Austria, 2001.

[7] J. Han, "A Comprehensive Interface Definition Framework for Software Components," Asia-Pacific Software Eng. Conf., 1998.

[8] Y. Myers and I. Exman, "Software Codons for Fast Program Reassembly from Components," IEEE Int'l Conf. on Software-Science, Technology & Eng., 2003.

[9] F. Ye and T. Kelly, "Use of COTS Components in Safety Critical Applications – A Defensible Approach," Int'l Workshop on Models and Processes for the Evaluation of COTS Components, 2004.

[10] S. A. Hissam, D. Carney, and D. Plakosh, "DoD Security Needs and COTs-Based Systems," *SEI Monographs on the Use of Commercial Software in Govt. Systems*, 1998.

[11] "Common Criteria for Information Technology Security Evaluation," http://www.commoncriteriaportal.org/public/files/ccpart2v2.3.pdf, June 2005.

[12] "Department of Defense Information Technology Security Certification and Accreditation Process," http://www.dtic.mil/whs/directives/corres/pdf/85101m_0700/p85101m.pdf, July 2000.

[13] R. Ross, et al., "Guide for the Security Certification and Accreditation of Federal Information Systems," NIST Special Publication 800-37, May 2004.

[14] M. Lorch, et al., "First Experiences using XACML for Access Control in Distributed Systems," ACM Workshop on XML Security, Fairfax, VA, 2003.

[15] A. Gummadi, J. Yoon, B. Shah, and V. Raghavan, "A Bitmap-Based Access Control for Restricted Views of XML Documents," ACM Workshop on XML Security, Fairfax, VA, 2003.

[16] F. Koshutanski and F. Massacci, "An Access Control Framework for Business Processes for Web Services," ACM Workshop on XML Security, Fairfax, VA, 2003.

[17] R. Kraft, "Designing a Distributed Access Control Processor for Network Services on the Web," ACM Workshop on XML Security, 2002.

[18] R. Bhatti, et al., "X-GTRBAC Admin: A Decentralized Administration Model for Enterprise-Wide Access Control," *ACM Trans. On Information and System Security*, vol. 8, pp. 388-423, 2005.

[19] M. J. Covington, et al., "Securing Context-Aware Applications Using Environment Roles," ACM Symp. on Access Control Models and Technologies, 2001.

[20] D. F. Ferraiolo, S. Gavrila, V. Hu, and D. R. Kuhn, "Composing and Combining Policies under the Policy Machine," ACM Symp. on Access Control Models and Technologies, Sweden, 2005.

[21] R. J. Hulsebosch, et al., "Context Sensitive Access Control," ACM Symp. on Access Control Models and Technologies, Sweden, 2005.

[22] I. Ray, N. Li, R. France, and D. Kim, "Using UML to Visualize Role-Based Access Control constraints," ACM Symp. on Access Control Models and Technology, New York, 2004.

[23] X. Zhang, F. Parisi-Presicce, R. Sandhu, and J. Park, "Formal Model and Policy Specification of Usage Control," *ACM Trans. on Information and System Security*, vol. 8, pp. 351-387, 2005.

[24] S. D. Di Vimercati, S. Marrara, and P. Samarati, "An Access Control Model for Querying XML Data," Workshop on Secure Web Services, Fairfax, VA, 2005.

[25] XACML, "eXtensible Access Control Markup Language (XACML) version 2.0," T. Moses, Ed. 2005.

Ranking Component Retrieval Results by Leveraging User History Information

Yan Li, Ying Pan, Lu Zhang, Bing Xie+, Jiasu Sun

Software Institute, School of Electronics Engineering and Computer Science
Peking University, 100871, Beijing, P.R.China
{liyan,pany,zhanglu,xiebing,sjs}@sei.pku.edu.cn

ABSTRACT

In the literature, many research efforts have been put into the improvement of component retrieval in reuse repositories. However, when retrieving components from a large reuse repository, it is usually unavoidable for a retriever to browse the retrieved component set to locate the appropriate ones. Therefore, the strategy for ranking the retrieved results may be essential for facilitating this task. In this paper, we propose a ranking strategy by considering user preference calculated from the user's history information. Furthermore, we also demonstrate that our strategy can be easily combined with other strategies. To evaluate the effectiveness of our strategy, we performed an experimental study. The experimental results demonstrate that our approach is effective and the combination of our strategy and a previous strategy may be more effective than either of the two.

1. INTRODUCTION

It is widely accepted that software reuse may be an effective way to improve software development productivity and quality [2]. To facilitate software reuse, many reuse repositories have been set up and can be accessed via Internet and/or Intranet [7]. To achieve this goal, the component retrieval mechanism in a reuse repository has to be both efficient and convenient [6] [18]. In the literature, many approaches to component retrieval have been proposed [17].

In a reuse scenario, typically, queries generated by the retriever are matched against each component according to the retrieval mechanism in the repository. Those matched components are returned as candidates and ranked in a descending order of the similarity between the query and the component descriptions. The retriever then has to browse and comprehend those candidate components to identify the appropriate ones [12].

However, for a repository containing a very large number of components (such as SourceForge.net [10] that has a collection of over 100,000 components), a typical candidate set of components for a query may become so large that the so-called "information overflow" problem become quite severe [3]. Thus, the process of identifying appropriate components will become much more time-consuming when the candidate set becomes larger. Furthermore, retrievers with larger candidate sets may have further cognitive difficulties, as the extra information may confuse or even intimidate them. To alleviate this problem, one way is to require the retriever to provide more sophisticated queries, but this may become a heavy burden as the formation of queries is usually a very difficult task [18]. Another way is to re-organize the retrieval results using some other information sources, and thus components that are more likely to be appropriate can be browsed earlier. A typical approach is the one proposed by Inoue etc. (see [13] and [14]), whose basic idea is to rank the components in the retrieved set using the use relations among components so that components widely used and with high quality will be in the front of the result list. In their experiments, it is demonstrated that ranking by use relations should be more effective than ranking by similarity between the query and the component descriptions.

In this paper, we propose a method to rank the candidate components using the information of user preference calculated from the user's history information. Intuitively, the aim of our method is to bring components that are likely to be preferred by the particular retriever to the front of the result list. As our approach utilizes information different to the approach in [13] and [14], the two approaches can be complementary to each other in practice. Furthermore, we also performed an experiment. The experimental results show that our approach is an effective one.

In the next section we present our motivation. In the third section we propose our method in detail. In section 4, we demonstrate how our approach can be combined with the approach using use relations. In section 5, we report an experimental study and analyze the results. Related research is discussed in section 6. We further discuss the future work and conclude this paper in section 7.

2. MOTIVATION

A fundamental goal of reuse repositories is to provide developers with components that satisfy their particular

+ Corresponding Author

needs [27]. As developers may have diverse background knowledge and tasks, their concerns for components may also be different.

Firstly, the domain in which developers are working may influence their choices of components. That is to say, developers may have diverse requirements for components according to the domain characteristics. For example, when needing a logging component, developers of real-time software may be primarily concerned with its performance. In the case of e-business software implementation, developers may need a component that is easy to modify to fit into all kinds of business processes.

Secondly, when facing the same result set, developers having different levels of skills and experiences may also make different choices. For example, an experienced developer may like to choose components conforming to some mature design patterns, whereas a novice may prefer components that are well-documented and easy to understand.

However, for current reuse repositories, the component retrieval mechanisms in them usually ignore these differences among retrievers and present the same retrieval results for different retrievers. That is to say, retrievers have to formalize their preference as part of the query to make their preferred components in the front of the result list. Obviously, this is an extra burden.

3. RANKING COMPONENTS BY USER PREFERENCE

3.1 Overview

The basic idea of our approach is to consider the user preference when retrieving components. To achieve this purpose, we propose a method that takes two steps.

- Firstly, for each specific retriever, we measure his or her preference for each component by leveraging his or her history of retrieval.
- Secondly, when a retriever is retrieving components, the set of retrieved components are ranked in accordance with the values calculated in the above measurement.

3.2 Measuring User Preference

The basis of our measurement of user preference is the historical retrieval information for each retriever. With the historical information, the main task of our approach is to figure out which kind of components each retriever may be particularly interested in recently.

There are many related researches based on analyzing user models. According to [4], these research efforts can be classified into two categories: memory-based methods and model-based methods.

The memory-based methods [4] [24] operate over the entire user history data to carry out further actions. In contrast, the model-based methods employ the user data to get a model first [1] [15] [25], such as the Bayesian network, which is then used to guide future operations.

To solve the ranking problem, we currently choose the memory-based method as it is easier to understand and implement. Specifically, we use the vector-formed history information to represent user preference, which is a typical memory-based method.

Meanwhile, the use or download times can naturally be a good indicator for component popularity, which has been applied in many widely-used online repositories such as ComponentSource.com [9] and SourceForge.net [10] for a long time. Therefore, we can make use of the download times in the vector form to represent user preference.

As a retrieved component may not be downloaded again by the same retriever (the user can use the local version when needed), we divide the entire set of components in the repository into several mutually excluded categories, and measure user preference on the basis of the categories of components. That is to say, if components in a particular category are frequently downloaded recently, the whole category becomes more preferred and thus all the components in this category become more preferred. Formally, according to the retrieval history for a particular period, the preference of a user for all the categories can be expressed in the form of a vector denoted as $V = < m_1, m_2, ... m_n >$, in which, n stands for the number of component categories, and m_i is the number of times the user has recently downloaded components in the i-th category ($1 \le i \le n$). In other words, the user preference of a particular component can be measured by the value in the vector for the category that the component belongs to.

In the above measurement, the categorization is similar to component classification that has been discussed in the literature [6]. In particular, for a hierarchical classification method, we can select a specific level to get a categorization of the components.

3.3 Ranking Retrieval Results

As mentioned above, each user has a vector calculated from his or her retrieval history, and this vector can be used to calculate his or her preference for each component. When the user submits a query to retrieve components, our approach can use the above measurement of user preference to rank the retrieved components, instead of using the similarity between the query and the component descriptions. The larger the preference value is, the higher the rank of the component is.

4. COMBINING USE RELATIONS AND USER PREFERENCE

In previous research [13][14], it has been demonstrated that use relations among the component set can be used to rank retrieved components and this kind of ranking is beneficial for users to identify appropriate ones. In this

section, we demonstrate that the ranking by user preference can be easily combined with the ranking by use relations.

As the two different ranking strategies both rely on a measurement value of each retrieved component, a straightforward idea for combining the two approaches is to multiply the two different metrics when determining the components' positions. For a component *c* in the retrieved set, the calculation of its measurement value for the ranking can be summarized in formula (1), where *R(c)* denotes its measurement value for ranking, *P(c)* denotes the measurement value calculated for user preference, and *I(c)* denotes the measurement value calculated for use relations.

$$R(c) = P(c) * I(c) \quad (1)$$

Actually, this strategy can be easily generalized to support ranking based on multiple measurement values. Supposing there are *m* metrics for component *c*, denoted as $R_i(c)$ $(1 \le i \le m)$, the final measurement value for ranking, denoted as *R(c)*, can be depicted in formula (2).

$$R(c) = \prod_{i=1}^{m} R_i(c) \quad (2)$$

5. AN EXPERIMENAL STUDY

5.1 The Experiment

To evaluate our approach, we performed an experimental study by applying our approach for retrieving classes in the JDK 1.4.2. Similar to previous experimental studies in [13] and [14], in our experiment, each class in JDK 1.4.2 was viewed as a component and the entire JDK 1.4.2 was viewed as the whole repository. As JDK 1.4.2 is organized as packages, we naturally treated each package as a category. Thus, the entire repository was divided into 135 categories.

Figure 1 illustrates the main step of our experiment. Firstly, we utilize JavaDoc to get the component description information. Specifically, we extract token from the JavaDoc and build index. Then, whenever the user submits a query, the search engine will get the candidate components according to calculated similarity between user's query and component description information. After that, we will leverage different criteria to reorder the candidate component list and then returned to the user.

In the experiment, we employed the well-known BM25 algorithm [22] [23], which is widely used in the information retrieval field [26], to calculate the similarity between each component description and the query.

$$S = \sum_{i \in q} \frac{(k_1+1)tf_i}{k_1((1-b)+b\frac{dl}{avdl})+tf_i} \log \frac{N-df_i+0.5}{df_i+0.5}$$

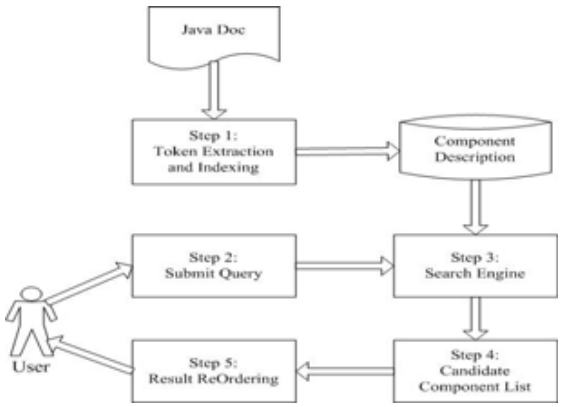


Figure 1. The Main Step in Experiment

In the formula above, *i* denotes for a word in the query *q*. tf_i and df_i stand for the term frequency and document frequency of the word *i* respectively [3]. The dl is the length of the component description. And the $avdl$ is the average length of all component descriptions. Here, k_1 and b are parameters, we set $k_1 = 1.1$ and $b = 0.7$.

Six graduate students majoring in computer science with experience of programming with Java for at least 2 years participated in our experiment as volunteer retrievers. Before the experiment, we parsed the java files in their recent development and calculated the use frequency of the classes in JDK 1.4.2. Then we got the statistics at the package level and formed their preference vectors like following (the corresponding package name is given for explanation):

< 203 (java.lang), 198 (java.util), ….. , 85(java.sql), ….>

In the experiment, each student was requested to generate 10 queries according to his or her needs for development for retrieving classes from JDK 1.4.2.

As our experiment aimed at evaluating whether our idea of ranking components by user preference is beneficial and whether the combination of use relations and user preference is beneficial, we used four different methods to organize the candidate component list returned by search engine in step 5. We compared the following results: the results ranked by similarity calculated by the BM25 (denoted as A), the results ranked by use relations (denoted as B), the results ranked by our vector-formed user preference (denoted as C), and the results ranked by combining user preference and use relations according to formula 2 (denoted as D). The four different ranking strategies were evaluated against two criteria: 1) precision and 2) user satisfaction.

To simulate the situation that retrievers usually only browse the top components, only the top ten components for each ranking were considered in our experiment.

5.2 Experimental Results and Analysis

5.2.1 Precision

Precision is the ratio of the relevant components retrieved by the query over the total number of components retrieved.

Table 1 shows the average precision for each student and the total average result. As only the top ten components are considered, it is no surprise that the precisions are relatively low. Besides this, we can also see that the average precision of ranking by user preference (C) is higher than ranking by similarity (A). Actually, ranking by user preference outperforms ranking by similarity for most retrievers. This indicates that user preference is a positive factor for ranking. Similarly, ranking by use relations (B) also seems superior to ranking by similarity. This confirms the findings in [13] and [14]. Furthermore, the combined method (D) yields the highest average precision and performs best for most retrievers. In particular, the combined method can outperform both ranking by use relations and ranking by user preference for most retrievers.

Table 1. Average Precision of the Four Methods

User	A	B	C	D
U_1	15%	13%	15%	19%
U_2	12%	13%	14%	13%
U_3	12%	17%	20%	18%
U_4	8%	12%	13%	14%
U_5	18%	24%	27%	29%
U_6	7%	7%	11%	12%
Avg.	12.0%	14.3%	16.7%	17.5%

Here, we employ the Paired Wilcoxon Test [11] to evaluate the differences of average precision between different methods. Paired Wilcoxon Test compares the magnitude of differences to determine whether the average precision between methods is significant different.

Firstly, let X_i and Y_i be the precision of the two methods compared in the i-th case. We define $D_i = X_i - Y_i$. Secondly, we ignore all the cases $D_i=0$, and sort all the remaining D_i to form a list L according to its absolute value $|D_i|$. So, after the sorting procedure, each case $D_i \neq 0$ gets a rank in the list L. We use R_i to denote the rank of D_i in L. Here, we define two sets:

$$V_+ = \{D_i \mid D_i \in L, D_i > 0\}$$

$$V_- = \{D_i \mid D_i \in L, D_i < 0\}$$

And let $S_+ = \sum_{D_i \in V_+} R_i$, and $S_- = \sum_{D_i \in V_-} R_i$.

We give the null hypothesis that the difference between average precisions does not exist. If the hypothesis holds, the chance that S_+ and S_- is small should be tiny. Let v_a be the maximal integer that satisfies

$$P(v_a) \leq \alpha$$

If $\min(S_+, S_-) \leq v_\alpha$, the testing illustrate that we can reject the hypothesis at α level.

Table 2. Paired Wilcoxon Test of Average Precision

Methods	S_+	S_-	$v_\alpha(\alpha = 0.1)$
A VS. B	2	13	0(accept)
A VS. C	0	15	0(reject)
B VS. D	0	15	0(reject)
C VS. D	6.5	14.5	2(accept)

The result of comparison is showed in Table 2. From the table, we can see that the average precision of ranking by use relation (B) and ranking by user preference (C) are both better than ranking by similarity (A). Besides, ranking by user preference can reject the hypothesis. As for the combined method (D), it has a superior average precision to ranking by use relation and ranking by user preference. And the difference between ranking by use relation and the combined method is significant.

5.2.2 User Satisfaction

In the experiment, students were also asked to give their opinion of the four ranking results for each query according to their satisfaction.

Table 3 shows the result of this comparison of user satisfaction. From the table, we can see that for over 60% times, students consider ranking by use relations (B) and ranking by user preference (C) are better than ranking by similarity (A). This can further demonstrate the effectiveness of our strategy of ranking by user preference and the strategy of ranking by use relations. This is in accordance with results reported in [13] [14]. Furthermore, the table also demonstrates that the combined strategy (D) is more satisfactory than both ranking by use relations and ranking by user preference in most cases. In other words, user preference and use relations can be complimentary to each other in ranking components.

We also make an analysis of the user satisfaction. As there is no quantitive data, we employ Sign Test [11] here, which is a non-parameter alternative of the t-test.

The null hypothesis is that the difference in user satisfaction between methods does not exist. We define the statistic n_+ and n_- as the number of 'Win' cases and 'Lose' cases respectively. Let $N = n_+ + n_-$. All the 'Draw'

cases are ignored. Let v_α be the maximal integer that satisfies

$$\sum_{i=0}^{V_\alpha} C_N^i \left(\frac{1}{2}\right)^N \le \frac{\alpha}{2}$$

If $\min(n_+, n_-) \le v_\alpha$, the testing demonstrate the hypothesis is rejected as α level. Thus, we believe there is significant difference between those methods.

Table 3. Comparison of User Satisfaction

A VS. B	Win	Draw	Lose
Times	18	4	38
Percentage	30	7	63
A VS. C	Win	Draw	Lose
Times	13	6	41
Percentage	22	10	68
B VS. D	Win	Draw	Lose
Times	11	12	37
Percentage	18	20	62
C VS. D	Win	Draw	Lose
Times	23	5	32
Percentage	38	8	54

Table 4. Sign Test of User Satisfaction

Methods	n_+	n_-	N	$v_\alpha(\alpha = 0.05)$
A VS. B	18	38	56	19(reject)
A VS. C	13	41	54	19(reject)
B VS. D	11	37	48	16(reject)
C VS. D	25	32	57	20(accept)

Table 4 shows the result of the sign test. From the table, we can see that all the hypotheses can be rejected at α level except for the case between ranking by use preference (C) and the combined method (D). Thus, we can conclude that both ranking by use relation (B) and ranking by user preference are superior to ranking by similarity (A). And the combined method is significant better than ranking by use relations. Besides, although the combined method seems better than ranking by user preference, the difference is not significant.

6. RELATED WORK

Many research efforts have been focused on how to retrieve components effectively and conveniently. In case of retrieval in single repository, some previous works employ mature techniques in other fields, such as IR [8] [16] etc. Drummond [5] provides an agent to infer users' intentions from their normal browsing actions and give advice to accelerate the search. *CodeBroker* [27] employs the information delivery technique to search components based on task-relevant information and the user model. Others [19] rely on formal methods to achieve high precision. To access resources from multiple reuse repositories via a single representation view, Pan [20] [21] etc. use the Bayesian network to transform queries for one repository to queries for another repository.

Besides the above research, there are efforts focusing on component ranking, whose aim is to re-organize the retrieved candidates by exploring other information sources. Typically, the approach in [13] [14] employs use relations to rank components using an algorithm similar to the one used in Google to rank web pages. This paper further investigates the benefit of ranking by user preference.

7. CONCLUSION AND FUTURE WORK

As a retriever may usually face a large set of retrieved components, a helpful aid for locating his or her desired components in this set is to rank the retrieved components. In previous research, use relations among components have been used for this ranking. In this paper, we propose to rank components according to user preference and describe how to combine our strategy with the ranking by use relations. In our experiment, we find that the strategy of ranking by user preference is effective and the combination of the two strategies can be superior to either strategy.

Utilizing the retrieval history of each user stored in the repository is the basis of our approach. However, as this information is changing continuously, we need to update the vector for each user periodically. Thus, there is a tradeoff between the precision and the workload of the system. In our experiment, the up-to-date information is used. In the future, we will further investigate how to determine the optimal period for updating automatically.

In our approach, user preference is represented in a vector, which is based on the categorization of the components. Although it is natural to use packages as categories for JDK 1.4.2 in our experiment, in real world reuse repositories, how to categorize the components may still be a problem, for which, it seems that the classification information is helpful. Besides, Java classes are cohesive, as the use of them is strongly related, while this may not be the case for components, and this makes another risk of our study. In the future, we will continue to pursue these issues. Furthermore, although we employ a straightforward strategy to calculate the vector using the retrieval history, we think a more sophisticated one might be more effective. Besides, other related methods to represent user preference, such as the model-based methods, will also be tried.

In the experiment, we have noticed that there are cases that ranking by similarity is superior to other methods. Thus, we plan to investigate whether and how the similarity can be taken into consideration when ranking the components.

Besides the retrieval history, feedback information collected from users who have downloaded and used some components may also be useful. How to utilize this information will also be one of our research topics in future.

8. ACKNOWLEDGEMENTS

This research was sponsored by the National Grand Fundamental Research 973 Program (SN: 2002CB312003) and the National Nature Science Foundation (SN: 60473059) in China.

9. REFERENCES

[1] Agrawal, R., Imielinski, T., and Swami, A., Mining Association Rules between Sets of Items in Large Databases, In *Proceedings of 1993 ACM SIGMOD International Conference on Management of Data*, 207-216, 1993.

[2] Basili, V., Briand, L., and Melo, W. How reuse influences productivity in object-oriented systems. *Comm. of the ACM*, 39, 10 (Oct. 1996), 104-116.

[3] Baeza-Yates, R. and Ribeiro-Neto, B. *Modern information retrieval*. Addison-Wesley/ACM Press, Reading, 1999.

[4] Breese, J.; Heckerman, D. and Kadie, C. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, 43-52, 1998.

[5] Drummond, C. G., Ionescu, D., and Holte, R.C. A learning agent that assists the browsing of software libraries. *IEEE Trans. on Software Engineering*, 26, 12 (Dec. 2000), 1179-1196.

[6] Frakes, W. B., and Pole, T. P. An empirical study of representation methods for reuseable software components. *IEEE Trans. on Software Engineering*, 20, 8 (Aug.1994), 617-630.

[7] Guo, J., and Luqi. A survey of software reuse repositories. In *Proceedings of the 7th International Conference and Workshop on the Engineering of Computer Based Systems*, 92-100, 2000.

[8] Henninger, S. An evolutionary approach to constructing effective software reuse repositories. *ACM Trans. on Software Engineering and Methodology*, 6, 2 (Apr. 1997), 111-140.

[9] "http://www.componentsource.com".

[10] "http://www.sourceforge.net".

[11] Hull, D. Using statistical testing in the evaluation of retrieval experiments. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information*, 329-338, 1993.

[12] Isakowitz, T., and Kauffman, R. J. Supporting search for reusable software objects. *IEEE Trans. on Software Engineering*, 22, 6 (Jun. 1996), 407-423.

[13] Inoue, K., Yokomori, R., Fujiwara, H., Yamamoto, T., Matsushita, M., and Kusumoto, S. Component rank: relative significance rank for software component search. In *Proceedings of the 25th ICSE*, 14-24, 2003.

[14] Inoue, K., Yokomori, R., Yamamoto, T., Matsushita, M., and Kusumoto, S. Ranking significance of software components based on use relations. *IEEE Trans. on Software Engineering*, 31, 3 (Mar. 2005), 213-225.

[15] Joachims, T. A Probabilistic Analysis of the Rocchio Algorithm with TF-IDF for Text Categorization. In *Proceedings of the 14th International Conference on Machine Learning,* 143-151, 1997.

[16] Li, G., Pan, Y., Zhang, L., Xie, B., and Shao W. Z. Attribute ranking: An entropy-based approach to accelerating browsing-based component retrieval. In *Proceedings of the 8th ICSR*, 232-241, 2004.

[17] Lucredio, D., Prado, A. F., and de Almeida, E. S. A survey on software components search and retrieval. In *Proceedings of the 30th Euromicro Conference,* 152-159, 2004.

[18] Mili, H., Mili, F., and Mili, A. Reusing software: issues and research directions. *IEEE Trans. on Software Engineering*, 21, 6 (Jun. 1995), 528-562.

[19] Morel, B., and Alexander, P. SPARTACAS: automating component reuse and adaptation. *IEEE Trans. on Software Engineering*, 30, 9 (Sept. 2004), 587-600.

[20] Pan, Y., Wang, L., Zhang, L., Xie, B., and Yang, F. Q. Relevancy based semantic interoperation of reuse repositories. In *Proceedings of the 12th FSE,* 211-220, 2004.

[21] Pan, Y., Wang, L., Zhang, L., Xie, B., and Yang, F. Q. An extended approach to improving the semantic interoperation among reuse repositories. In *Proceedings of the 29th COMPSAC*, 89-94, 2005.

[22] Robertson, S.E., Walker, S., Jones S., Hancock-Beaulieu, M.M. and Gatford, M. Okapi at TREC 3. In *Proceedings of the Third Text REtrieval Conference (TREC-3)*, 109-126, 1994.

[23] Robertson, S., Zaragoza, H. and Taylor M. Simple BM25 extension to multiple weighted fields. In *Proceedings of the 13th International Conference on Information and Knowledge Management*, 42-49, 2004.

[24] Schwab, I., and Pohl, W. Learning User Profiles from Positive Examples. In *Proceedings of the 1999 International Conference of Machine Learning & Applications*, 15-20, 1999.

[25] Silverstein, C., Brin, S. and Motwani, R. Beyond Market Baskets: Generalizing Association Rules to Dependence Rules. *Data Mining and Knowledge Discovery*, 2, 1 (Jan. 1998), 39-68.

[26] Trotman, A. Learning to Rank. *Information Retrieval*, 8, 3(May 2005), 359-381.

[27] Ye, Y., and Fischer, G. Supporting reuse by delivering task-relevant and personalized information. In *Proceedings of the 24th ICSE*, 513-523, 2002.

A Framework for Component-based System Modeling

Zhijiang Dong Yujian Fu Xudong He
School of Computing and Information Sciences
Florida International University
E-mail: {yfu002, zdong01, hex}@cs.fiu.edu

Abstract

This paper presents a framework to model component-based systems made up of a set of components interacting with each other through message exchange. Component behavior is specified by algebraic high-level nets. The idea of "nets as tokens" is explored to integrate behavioral model with component communication mechanisms. Component interactions are modeled by transformation rules based on the category of algebraic high-level nets. Component models are synthesized into a valid system model by applying transformation rules according to predefined consistent conditions. In addition, transformation rules can also be explored to refine component models. The main contribution of the framework is the explicit separation and seamless synthesis between component models and their interaction models based on different techniques. Another contribution is the application of "net as token" to algebraic high-level nets to model more complex components.

1. Introduction

Currently the most popular support in industry for component-based frameworks appears to be COM+ and CORBA. Unfortunately, components in these frameworks lack a precise semantics probably due to their focus on system implementation, which makes it dif cult to reason about this kind of systems. Many formal methods have been proposed to model and analyze component based systems, such as Piccola Calculus [14], Abstract Behavior Types [1], Eiffel Language [7] etc. In this paper, we use Petri nets as the underlying formal method, and present a component modeling framework.

One particular concern in component-based systems is the component modeling. The generic component modeling, presented in this paper, has been mainly motivated by the ideas in [22] for "tiered component framework", and by the concepts of "nets and rules as tokens" for Petri nets [8, 23, 24]. In [22], component frameworks are organized into multiple layers, and two layers often suf ce in most cases. Blackbox frameworks accept "plug-in" components without modi cations to the framework. The architecture can be extended further: a component framework itself can be slotted into a higher tier framework that regulates interactions. Such an idea is adopted in our work to separate component functionalities from interactions and required properties such as responsiveness, scaleability, security, and reliability. More speci cally , the internal behavior is captured by a function net whereas interactions and required properties are modeled by component nets in which function nets serve as tokens.

Although components have been the predominant focus of research, they address only one aspect of component-based software development. Another important aspect is the interactions among components, i.e. connectors. Connectors are sometimes deliberately modeled as components (connection components in Rapide [11]). In this paper, in order to make the distinction clearer, we use a different technique – transformation rules [18] to model connectors. Although the main purpose of adopting transformation rules is to model connectors, they can also be explored to re ne component nets in multiple ways to add additional functionalities such as creation and destruction of components. Fig. 1 shows an overview of the framework proposed in this paper. Components' internal behaviors, captured by function nets, are wrapped by component nets, which not only deal with interactions with other components, but also model non-functional component requirements. A set of component nets are composed into a (sub)system model by applying some transformation rules. Such an approach is e xible, and makes the reuse and maintenance of components and connectors easier since connectors and components are independent from each other in the framework.

The ability to compose Petri nets is fundamental to component-based system modeling. In literatures, there are other ways to compose Petri nets to form a system model. One of them is to construct an algebra of Petri nets over constants and compositional operators as in [13,19]. In their work, labeled Petri nets are extended with interfaces (public

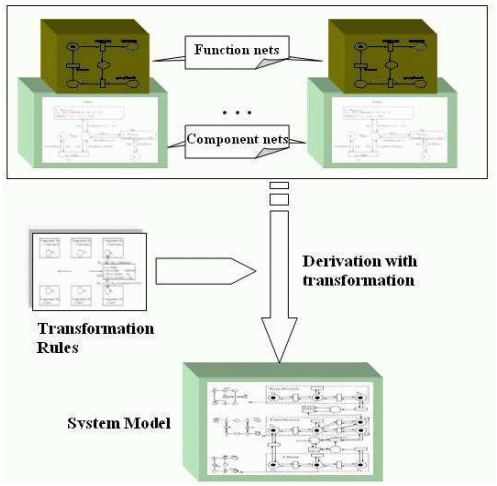


Figure 1. An Overview of the Framework

places and transitions) through which the component communicates with the external environment. Another way to compose Petri net is through place fusion [4, 9], transition fusion [3], or both [5]. However, place fusion and transition fusion are very tightly coupled, which cannot decide the enabling of a transition locally, and even worse violates the modular principle of incremental system development. The last way to compose Petri nets is based on category theory [10] [2]. Unlike our work, there is no explicit separation among component models and their interaction models, which violates reusability and maintainability.

The rest of the paper is organized as follows: Section 2 explains the framework informally. For the formal de nition of related concepts, please refer to [6]. An example is given in section 3 to illustrate the application of the framework. Finally, we give the discussion and conclusion.

2. The Framework

2.1. Function Nets

Function nets are used to model components' functionalities without the concern of interactions and non-functional requirements such as responsiveness, scaleability, security, and reliability etc. More speci cally , function nets specify the responses of components to a message from the external environment.

In the framework, function nets are a special kind of algebraic high-level nets [18] by classifying places into three categories: input places, output places and internal places. The input places contain messages received from the external environment. The output places contain messages to the external environment as responses, while the internal places indicate component status. The input, output, and internal places are supposed to be disjoint, otherwise the meaning of a message in such places is ambiguous. Upon reception of a message in an input place, a function net can be executed until reaching a stable status, in which no transition is enabled and the component is waiting for the next message from the external environment. As a result of handling a received message, in most cases, at least one message in an output place is generated as a response to the external environment.

In most cases, the sets of input and output places are not empty. However, if both sets are empty, we say the component is a closed system that does not interact with other components or systems. If only the set of input places is empty, we say the component is a message generator, which affects its environment. If only the set of output places is empty, the component is called recorder, which only records environment's in uence on itself without feedback.

Given a component, we can either model its behavior as a function net from scratch , or make little modi cations to the available Petri net behavioral model to meet the de nition of function nets. However, it is in general impossible to model or obtain the behavioral model of commercial-off-the-shelf (COTS) components. What we have known about these blackbox components is the well-de ned relationship among interfaces (input and output places). Fortunately, we can either construct a behavioral model from such relationships or use algebraic speci cations to represent such interface relationships of blackbox components . In either way, component nets work correctly since a function net is actually treated as an algebraic speci cation due to the fact that Petri nets are monoids [12].

2.2. Component Nets

A component not only has its own behavior, which is modeled by function nets, but also needs to communicate with other components through message exchange, and may have some non-functional requirements such as responsiveness, scaleability, security, and reliability etc. Therefore, we adopt the idea "nets as tokens" to synthesize function nets with communication mechanism. The paradigm "nets as tokens" was introduced by Valk in order to allow nets as tokens, called object nets, within a net, called system net [23, 24]. The object nets may not only change its marking, but also modify its net structure in the context of system nets. Such characteristics, together with the communication complexity between objects nets and system nets, con ne the research of object nets on low level Petri nets.

Fortunately, net structures of function nets in our work are supposed to be unchangeable. Therefore, an algebraic high level net may be represented by an algebra, which can be adopted by another algebraic high level net as part of its

speci cation, which is viable since Petri nets are monoids [12]. Therefore, a signature and an algebra are constructed in the framework for a given function net.

In the framework, a generic component net as shown in Fig. 2, is an extension of algebraic high-level nets based on "nets as tokens" to model a set of components sharing the same kind of function nets. The places P_i and P_o contain a token respectively, which represents an input/output message queue as message pool. Although we choose the term queue in this paper, it actually can be any other data structures such as sets, lists, or stacks. A token in the place P_{object} is actually a function net. The transition t_{in} controls message movement from the input message pool into an input place of the function net, while the transition t_{out} indicates the time to move a message from an output place of the function net into the output message pool, the messages in which are visible to the external environment. The transition $t_{response}$ models the action of executing the function net.

2.3. Model Synthesis

After obtaining component nets for components in a system, we need to provide an approach to model interactions in the form of message exchange among component nets as well as a methodology to integrate component nets into a system model in a modular and incremental way.

In the framework, transformation rules (or productions) in HLR-category (**AHLNET**, M_{AHLN}) [18] are adopted to tackle the above problems. Due to page limits, we cannot give a brief introduction to transformation rules and HLR-category. Please refer to [18] for details. By exploring transformation rules, the framework has the following advantages as well as e xibility and powerful expressiveness:

- Transformation rules have formal semantics. Since Petri nets are also a formal graphic modeling language, our methodology of system modeling has a strong theory basis.
- By adopting transformation rules, we not only separate component modeling from channel/connector modeling, but also distinguish dynamic component creation and destruction from component modeling.
- Transformation rules can also be explored to re ne/construct component nets.
- By adopting transformation rules, system can be modeled in a modular and incremental way.
- It is easy to model similar but different systems by choosing different sets of productions.

Multiple types of transformation rules are de ned for various purposes in the framework. Table 1 gives a summary of production types. Creation/destruction message passing productions are distinguished from interaction productions since such messages are passed from a component to a component net, not from a component to another component just like interaction productions. Re nement productions are used to re ne component nets, especially the relationship between message pools and function nets to support complicated behavior such as run-to-completion assumption in UML state machine diagrams.

We have to point out that currently we do not take message broadcasting into account. Given a transformation rule (L $\leftarrow$ K $\rightarrow$ R) and an AHL-net G, the occurrence of L in G is not unique, and therefore we may get multiple AHL-nets. Not all of them are valid (sub)systems with the concern of requirements. A consistent condition is given for each kind of productions to guarantee that the system model we obtain is valid. The formal de nition and consistent conditions for transformation rules can be found in [6].

The methodology to model systems in a modular and incremental way under the framework is summarized in the following. An example is given in next section to explain the approach further.

- Modeling function nets for various components. In this step, transformation rules can also be explored in to model function nets in a modular and incremental way.
- Constructing generic component nets as in Fig. 2 by exploring idea of nets as tokens.
- De ning appropriate transformation rules according to requirements such as component communications, component semantics etc.
- Applying re nement transformation rules to basic component nets in the way that consistent conditions are satis ed.
- Applying creation/destruction transformation rules to component nets in the way that consistent conditions are satis ed.
- Composing component nets into a valid system model by applying interaction transformation rules and creation (destruction) message passing productions such that consistent conditions are satis ed.

3. Example

In this section, a small example is showed to explain the framework and the approach. Although the example is trivial, most of distinguished features of our framework are covered. Our running example is an internet timer server. Three kinds of components are involved in this example:

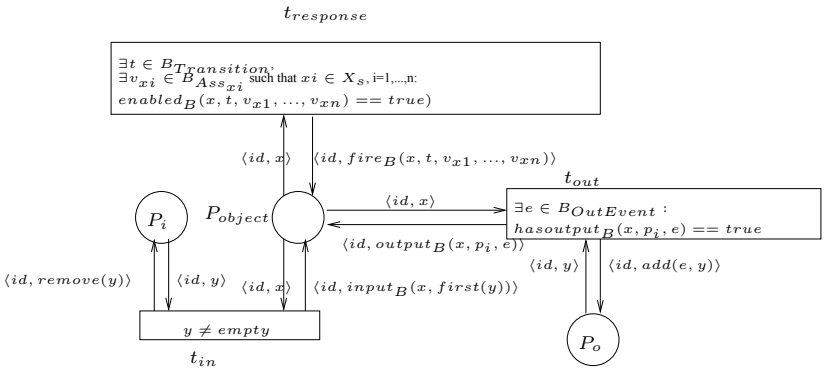
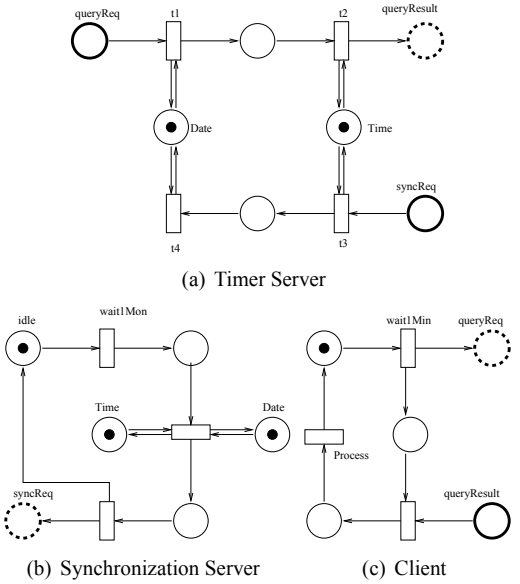


Figure 2. A Generic Component Net

Types	Description
Re nement Production	Re ning generic component nets
Creation Production	Adding dynamic component creation functionality to component architectures
Destruction Production	Adding dynamic component destruction functionality to component architectures
Interaction Production	Connecting component architectures through message exchange
Message Creation Passing Production	Passing creation messages from a component to a component net
Message Destruction Passing Production	Passing destruction messages from a component to a component net

Table 1. Summary of Production Types



(a) Timer Server

(b) Synchronization Server (c) Client

Figure 3. Function Nets

timer server, synchronization server, and clients. A timer server records the most accurate date and time information. A synchronization server, having correct time information, is used to synchronize time information on timer servers regularly. Clients only have rights to query time information by submitting requests to timer servers every minute. The timer server handles query requests by fetching date and time in sequence, while updating in reverse sequence for synchronization requests. The function nets are shown in Fig. 3. Input and output places are denoted by circles with thick lines and dashed thick lines respectively.

Traditional way to integrate them into a system is through place merge. However, place merge may introduce errors by violating semantics of some individual components. In this case, a system can be obtained by merging input and output places according to place names. It is possible that there is a token in the places *queryReq* and a token in the place *syncReq* representing a query request and a synchronization request respectively. Suppose the time information in the timer server is 2005-08-16 23:59:59 while the time information in the synchronization request is 2005-08-17 00:00:01. The possible transition ring sequence $t1$,$t3$,$t2$,$t4$ results in the reply of 2005-08-16 00:00:01 to the client, which is obviously wrong. The error is due to the violation of request processing atomism, which motivates us to propose the two-level methodology: the low level (function net) is used to model component behavior, while the high level (component architecture) is used to model com-

munication mechanism.

Fig. 4 shows some transformation rules de ned in this example. In this example, we do not consider dynamic component creation and destruction. Fig. 4(a) is a re nement transformation rule, which distinguishes synchronization request from other query requests. More speci cally , synchronization request has higher priority than query requests, i.e. timer servers process synchronization requests immediately whenever it is available and function nets are stable. Fig. 4(b)-4(d) show interaction productions that integrate different component nets into a system. More speci cally , Fig. 4(b) describes a unreliable channel, i.e. message may be lost. Fig. 4(c) shows a controllable channel, in which a message may be ltered if it does not satisfy some condition with regard to history data. This kind of channel can protect timer server from evil clients who may attack a timer server by sending huge amount of requests in a short period of time. Fig. 4(d) shows a simple pipeline channel. Fig. 5 shows the system obtained by applying transformation rules in Fig. 4.

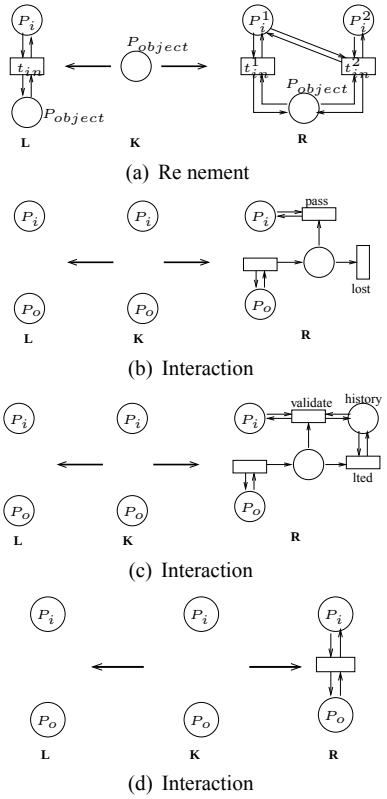


Figure 4. Transformation Rules

4. Discussion and Conclusions

Among the previous works, the works of Padberg [15–17] based on category theory and Sibertin-Blanc [20, 21] based on arc fusion are the closest to ours. Padberg et al. speci ed a component as a model speci cation of import interface IMP, export interface EXP, and body BOD connected by an embedding morphism $imp : IMP \rightarrow BOD$ and an substitution morphism $exp : EXP \rightarrow BOD$. IMP, EXP, and BOD are objects of the category of Place/Transition nets. Three module operations Disjoint Union, Union, and Composition are de ned to provide at and hierarchical composition semantics for Place/Transition nets. Unlike our work, they considered low level Petri nets with a marking as the basic objects, and composition of components is always well-de ned, which is guaranteed by a consistency condition in our work. They model components by importing and exporting functionalities, while we focus on concurrent distributed systems interacting with each other through message exchange.

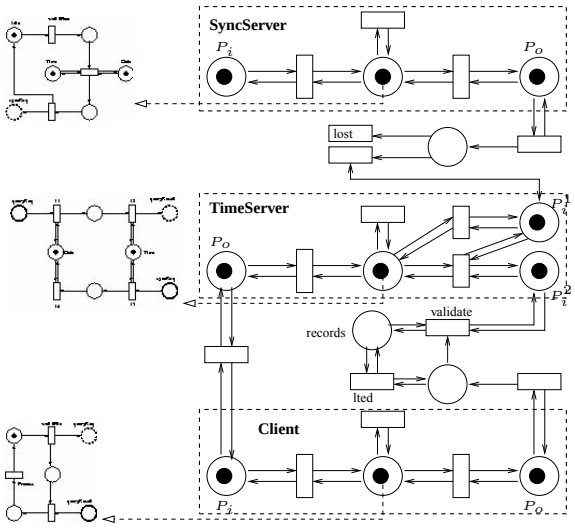


Figure 5. Timer Server System

Blanc [20, 21] proposed another Petri net based formalism for the modeling, analysis and simulation of systems: Cooperative net and communication net, both of which can model complicated distributed systems as a set of components that have their own internal structures and behaviors, and also communicate with each other through message passing. Each component is a cooperative/communication net. The composition of components is achieved through arc fusion, a looser coupling compared with place and transition fusion. Although the enabling of a transition can be judged locally, the ring of a transition is de ned globally. Even worse, there is a structural dependence among components due to the potential structural reference in transition actions. More speci cally , one component has to refer to other components' internal places for the purpose of

communication, which is in general not available during modeling process. Therefore, structural dependence makes the reuse of components and support for incremental design harder.

This paper presents a framework to model component-based system in an incremental way. The framework separates concerns of component models and their interaction models explicitly. A component is modeled by an algebraic high-level Petri net. By introducing the idea of "nets as tokens" to algebraic high-level Petri nets, we can model more complex components due to the exibility in handling the relationship between component behavior and communication mechanism. Component interactions are speci ed by productions based on a HLR-category. In addition, productions can also be explored to re ne component behavior and its relationship with communication mechanism, and model functionalities of dynamic component creation and destruction. In the framework, different techniques are synthesized seamlessly.

Acknowledgements This work is supported in part by NSF under grant HRD-0317692, and by NASA under grant NAG 2-1440. The author Zhijiang Dong appreciates the - nancial support from University Graduate School, Florida International University in the form of Dissertation Year Fellowship.

References

[1] F. Arbab. Abstract Behavior Types: A Foundation Model for Components and Their Composition. *Science of Computer Programming*, 55(1–3):3–52, 2005.

[2] P. Baldan, A. Corradini, et al. Compositional Semantics for Open Petri Nets based on Deterministic Processes. *Mathematical Structures in Computer Science*, 15(1), 2005.

[3] J. P. Barros and L. Gomes. A Unidirectional Transition Fusion for Coloured Petri Nets and its Implementation for the CPNTools. In *Proceedings of the 5th Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, pages 199–218, 2004.

[4] T. Basten. *In Terms of Nets : System Design with Petri Nets and Process Algebra*. PhD thesis, Department of Computing Science, Eindhoven University of Technology, Eindhoven, The Netherlands, 1998.

[5] S. Christensen and L. Petrucci. Towards a Modular Analysis of Coloured Petri Nets. In *Proceedings of 13th International Conference on Application and Theory of Petri Nets*, volume 616 of *Lecture Notes in Computer Science*, pages 113–133, 1992.

[6] Z. Dong. An Incremental Approach for Composing Petri Net Models. Technical report, CADSE, School of Information and Computing Sciences, Florida International University, USA, 2006.

[7] J. Gore. *Object Structures: Building Object-Oriented Software Components With Eiffel*. Addison-Wesley Pub, 1996.

[8] K. Hoffmann, H. Ehrig, and T. Mossakowski. High-Level Nets with Nets and Rules as Tokens. In *Applications and Theory of Petri Nets 2005, 26th International Conference*, volume 3536 of *Lecture Notes in Computer Science*, pages 268 – 288, 2005.

[9] E. Kindler. A Compositional Partial Order Semantics for Petri Net Components. In *ICATPN '97: Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, volume 1248 of *Lecture Notes in Computer Science*, pages 235–252, 1997.

[10] J. Lilius. On the Compositionality and Analysis of Algebraic High-Level Nets. *Digital Systems Lab. Series A: Research Report*, (16), 1991.

[11] D. C. Luckham and J. Vera. An Event Based Architecture De nition Language. *IEEE Transactions on Software Engineering*, 21(9), 1995.

[12] J. Meseguer and U. Montanari. Petri Nets are Monoids. *Information and Computation*, 88(2):105–155, Oct 1990.

[13] M. Nielsen, L. Priese, and V. Sassone. Characterizing Behavioural Congruences for Petri Nets. In *Proceedings of 6th International Conference on Concurrency Theory*, volume 962 of *Lecture Notes in Computer Science*, pages 175–189, 1995.

[14] O. Nierstrasz and F. Achermann. A Calculus for Modeling Software Components. In *Formal Methods for Components and Objects, First International Symposium (FMCO 2002)*, volume 2852 of *Lecture Notes in Computer Science*, 2003.

[15] J. Padberg. Place/Transition Net Modules: Transfer from Speci cation Modules. Technical Report Technical Report 2001-03, TU-Berlin, 2001.

[16] J. Padberg. Petri Net Modules. *Journal on Integrated Design and Process Technology*, 6(4):121–137, 2002.

[17] J. Padberg and H. Ehrig. Petri Net Modules in the Transformation-based Component Framework. *Journal of Logic and Algebraic Programming*, accepted, 2005.

[18] J. Padberg, H. Ehrig, and L. Ribiero. Algebraic High-Level Net Transformation Systems. *Mathematical Structures in Computer Science*, 5:217–256, 1995.

[19] L. Priese and H. Wimmel. A Uniform Approach to True-Concurrency and Interleaving Semantics for Petri Nets. *Theoretical Computer Science*, 206(1-2):219–256, 1998.

[20] C. Sibertin-Blanc. A Client-Server Protocol for the Composition of Petri Nets. In *Proceedings of 14th International Conference on Application and Theory of Petri Nets*, volume 691 of *Lecture Notes in Computer Science*, pages 377–396, 1993.

[21] C. Sibertin-Blanc. Cooperative Nets. In *Proceedings of the 15th International Conference on Application and Theory of Petri Nets*, volume 815 of *Lecture Notes in Computer Science*, pages 471–490, 1994.

[22] C. Szyperski and R. Vernik. Establishing System-wide Properties of COmponent-based Systems – A Case for Tiered Component Frameworks. In *OMG-DARPA-MCC Workshop on Compositional Software Architectures*, January 1998.

[23] R. Valk. Petri Nets as Token Objects - An Introduction to Elementary Object Nets. In J. Desel and M. Silva, editors, *19th International Conference on Application and Theory of Petri nets*, volume 1420 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 1998.

[24] R. Valk. Concurrency in Communicating Object Petri Nets. volume 2001 of *Lecture Notes in Computer Science*, pages 164–195. Springer-Verlag, 2001.

Evolution problem within Component-Based Software Architecture

D.Tamzalit, N. Sadou and M. Oussalah

Nantes University, Nantes AtlanticUniversities, CNRS
LINA, FRE 2729
2, rue de la Houssinière
BP 92208 44322, Nantes F-44000 France
Dalila.Tamzalit, Nassima.Sadou, Mourad.Oussalah@univ-nantes.fr

Abstract—**we approach in this article the evolution problem of component-based software architecture. We present first the important issues of this problem through the two categories of evolution supported by the principal ADLs: static and dynamic evolutions. We come out of these issues with a set of criteria of comparison and a set of evolution problems that given evolution approach can answer. According to these criteria and regarding these problems we propose SAEV, a model for enhancing software architecture evolution. SAEV proposes a set of concepts and operations allowing software architectures to evolve. It also considers that software architecture is represented on three abstraction levels, namely: the *meta level*, the *architectural level* and the *application level*. SAEV aims to manage the evolution at these different levels. For that purpose, it is based on evolution operations, described using evolution strategies and evolution rules. These rules satisfy invariants associated to the elements in order to carry out and to guarantee the coherence of the evolution. SAEV is intended to be a generic, uniform, and an open model, allowing the description of the evolution at a high level of abstraction and therefore facilitating its reuse .**

1. Introduction

Software architecture needs to be changed during its lifetime. Original requirements may change to consider, for sample, the customer needs or the system environment changes. Evolution is an important mechanism to defer these changes on the system and to guarantee its long lifespan. In spite of important progress on architecture description languages, we note that few approaches or mechanisms are proposed to face software architecture evolution problems. Before dealing with software architecture evolution, we briefly introduce what is software architecture.

Software architecture offers a high abstraction level for the description of systems, by defining their architectures generally in terms of components and connectors and without code details. It determines not only how the system should be constructed but also it may guide its evolution [11]. Currently, several Architecture Description Languages (ADLs) are proposed to aid the architecture-based specification and development, such as C2 [2], ACME [8] and Rapide [12]. Their common descriptive main concepts are: component, connector, configuration and interface. A component represents a computational element and data stores of a system while a connector is used to represent interactions among components as well as the rules that control these interactions. A configuration represents a graph of components and connectors. It describes how they are fastened to each other. Finally, an interface is the only visible part of components, connectors and configuration. It provides a set of services (provided or required) and interaction points.

We present in this article an analysis of evolution within component-based software architectures. This analysis relates to various approaches of evolution supported by principal ADLs. We come out of this analysis with a set of comparison criteria and a set of evolution problems that a particular evolution approach can answer. In line with these criteria and problems, we propose SAEV: Software Architecture Evolution Model. SAEV aims to describe and manage the evolution of a given software architecture. It considers architectural elements (like component, connector, configuration and interface) as first-class entities and leans on its own concepts and evolution mechanisms. The evolution of each architectural element is managed through the evolution strategies and evolution rules. An evolution rule describes an evolution operation on an architectural element.

The reminder of the paper is organized as follows: section 2 approaches software architecture evolution problems, according to two categories of evolution: static and dynamic evolutions. Section 3 describes the model SAEV through its main objectives and motivations, the abstraction levels that it considers and through its meta-model, concepts and mechanism. Section 4 presents a projection of SAEV on UML2.0, and the last section presents our perspectives before concluding.

2. Evolution issues within Software Architecture

As any software artifact, software architecture may evolve to reflect evolution needs. For example, we may need to add new components, to modify connection between these components… We distinguish two categories of evolution: static evolution and dynamic one.

2.1. Static Evolution

Static evolution is supported by mechanisms proposed by an ADL. These mechanisms are often influenced by object-oriented evolution mechanisms. We can cite for example

instantiation, *subtyping*, *inheritance*, c*ompositionality* or *refinement.* Only ADLs that consider architectural elements as first-class entities propose mechanisms to ensure their evolution. We illustrate in the following the *instantiation* and *subtyping* mechanism used by main ADLs (ACME [8], C2 [2], C2SADEL [14], Rapide [12], UniCon [21]…)

a. Instantiation*:* software architecture distinguishes between component and connector types, where component types are abstractions that encapsulate functionalities into reusable blocks, and connector types are abstractions that encapsulate components' communication. All ADLs consider component as a type that can be instantiated many times in a single system. Regarding connectors, only ADLs that model them as first-class entities support their instantiations. ADLs such as Darwin [13], Metah [16] and Rapide [12] do not support connector instantiation, since they do not represent them as first-class entities. Regarding configurations, the majority of ADLs define them only as a graph of component-instances and connector-instances.

b. Subtyping: subtyping is where an object of one type may safely be substituted where another type was expected. In software architectures components and connectors are architectural types and they are distinguished from the basic types (e.g., integers, characters, array, etc.). C2SADEL [14] is the ADL that most exploited the mechanism of subtyping. It proposes three different components subtyping relationships: *interface* subtyping, *behavior* subtyping and *implementation* subtyping. C2SADEL supports also heterogeneous subtyping by combination of the three types of subtyping, using clauses such as *and*, *or*, *not*. Whereas, ADL like ACME [7] supports both components and connectors subtyping using its *extends* clause. Unicon defines components by enumeration, so they do not provide any mechanism to evolve them.

2.2. Dynamic evolution

Dynamic evolution is a new concern in ADLs. It means the possibility of introducing modifications in a system during its execution... Dynamic architecture changes may be either planned at specification time or unplanned [15]. In the first case, changes likely to occur during the execution of the system must be known initially, so they must be specified during the description of the architecture. Rapide[13], for example supports conditional configuration, using its clause *where* which enables a form of architectural rewiring using the *link* and *unlink* operators [15].

Dynamic ACME associates with each element a multiplicity which can indicate the number of instances of this element or if it is optional.

The unplanned evolution places no restrictions at the architecture specification time on the kinds of allowed changes [14]. Thus, the ADL supporting this kind of evolution must offer architecture modification features, which allows the architect to specify changes during the system execution. P. Oreizy [18] enumerates the following operations that must be supported by ADLs at runtime: *insertion*, *removal* and *rewriting* of elements in architecture. ArchStudio[18] is an interactive tool which supports these operations on architectures specified in the C2 style.

2.3. Conclusion of the analysis

Regarding the presented evolution approaches, we can note that evolution approaches are concentrated on the evolution of components and more precisely on their structure. The specification of static evolution is integrated within the architecture specification, depending thus on the architecture's ADL. This category of evolution concerns only component or connector types, it doesn't propose mechanisms to evolve the component/connector instances. Regarding the dynamic evolution, notably the unplanned evolution, proposals remain at a research stage and what is proposed by Archstudio is used only with architectures described in the C2 style. We can conclude that evolution is not considered as an explicit problem within software architectures. Proposed supports for the two categories of evolution are not sufficient to answer all needs of software architecture evolution.

According to this analysis, we have identified the following criteria to characterize and compare architecture evolution mechanisms. These criteria can be used to describe in a uniform way evolution approaches (by informing each criterion) but also to describe any evolution problem, in order to find a solution regarding these criteria. We introduce them byway of interrogations:

- **CR1**: is the evolution independent or not from the specification of architectural element behaviour? Thus, it implies the independence of the evolution mechanism from the Architecture Description Language;
- **CR2**: what are the architectural elements (A.E) on which evolution supports are proposed?
- **CR3**: are there evolution operations proposed for each architectural element?
- **CR4**: is the management and the propagation of the evolution impacts automatically done or not?

We use the identified criteria to illustrate comparison among the ADLs C2, ACME and UniCon (table 1).

By exploiting this table, we note that in the case of the ADLs ACME, C2, the specification of the static evolution is incorporated within the architecture specification and it is described with the architecture description language. So, the specification of a given evolution can not be reused. We can note also that these ADLs do not offers the same mechanisms, and the same operations to ensure the evolution of each concept. C2 remains the ADL, which most exploit the dynamic evolution by proposing different evolution operations.

Crit	Ev. Categ.	ACME	C2	UniCon
CR1	Static	The evolution is specified with the ADL. Thus it is integrated in the architecture description		Don't propose mechanisms to evolve components and connectors (they are defined by enumeration...
	Dynamic	With dynamic ACME the architecture can evolve dynamically, but it still integrated in the architecture description	Proposes the ArchStudio tool which allows an interactive evolution, independently of the architecture specification	
CR2	Static	It allows the specification of the evolution of *components and connectors* type and of families which define the whole of the system's types.	It allows the specification of the evolution of the *components* and *connectors* type.	Defined the configuration evolution (called composite component)
	Dynamic	It allows the evolution specification of components , connectors, and configurations (called systems)	It allows the evolution specification of components, connectors, and configurations.	Don't propose mechanisms for dynamic evolution
CR3	Static	Don't provide explicit evolution operations but evolution mechanisms such as: structural subtyping, composition, genericity and refinement	Don't' provide explicit evolution operations. Offers heterogeneous subtyping and refinement mechanisms.	Don't' provide explicit evolution operations. Supports refinements and composition mechanism
	Dynamic	Don't provide explicit evolution operations	operations : addition, deletion, upgrading, of component, and reconnection of the connectors and components	/
CR4	Static	Evolution specification is integrated with the architecture description before compilation, so no automatic propagations		/
	Dynamic	The dynamic changes are planned in the specification, as well as the propagations of the impacts of these changes	The evolution is managed by the Archstudio tool as well as the necessary propagations.	/

/: not defined

Table 1. Evolution mechanisms comparison

Finally, evolution needs are not completely treated and managed. We are convinced that all evolutions must be identified and uniformly managed to answer each evolution needs without violating architecture coherence. Considering these purposes, we propose the model SAEV.

3. SAEV: a model for Software Architecture Evolution

SAEV considers the software architecture evolution as the different changes that can occur during the architecture lifecycle. These changes can be for example the addition of new concepts (on a meta-model), or new components (on an architecture), the modification of the connections between components (within an application)… All these changes need to be identified and managed. We first present the motivations and the mains objectives of SAEV. We then outline the different abstraction levels that we consider, before presenting SAEV:

3.1. Motivations and objectives of SAEV

SAEV aims to describe and model all evolutions that occur during the life cycle of a given architecture. It aims also to manage the execution and the impacts of each evolution. SAEV is interested more precisely by the structural evolution of these architectures. For these purposes and considering the previous identified criteria (section 2), SAEV aims to:

- Abstract the evolution, from the specific behaviors of architectural elements. That allows to:
  - Define description and management mechanisms of the evolution independently of the architectural elements behavior and their description languages.
  - Support the re-use of these evolution mechanisms in several cases of evolution.
- Identify the impacts of each evolution, and determine the mechanisms to propagate these impacts.
- Support static evolution (at the specification time) as well as dynamic one (at the application execution time).

Software architectures can be described at different abstraction levels. SAEV considers three abstraction levels: meta level, architectural level and application one. These levels are necessary in order to be able to reify all architectural elements and, consequently, to be able to manage their evolution.

3.2. Software Architecture abstraction levels

Some of the surveyed ADLs such as C2 [2], ACME [8] and Rapide [13] consider components and connectors as first-class entities and distinguish respectively the component-type and the connector-type from their component-instances and connector-instances. But, the configuration is often considered only at the application level as a graph of components-instances and connectors-instances and not as first-class entity. In our work, we consider all architectural elements as first-class entities, according to the three abstraction levels:

- **Meta level**: is the level where each concept of any ADL is defined, like Configuration, Component and Connector.
- **Architectural level**: at this level the architecture of a given system is described through instances of the architectural elements of the meta level. Figure 1 presents Client/Server architecture with a *Configuration* CSConf; three *components* Client, Server, Database and of two *connectors* N1 and N2.
- **Application level**: in this level one or more applications can be defined according to their architecture. For example, from the previous client/server architecture, the following application is build and is made up of: Cf, instance of the configuration *CSConf*; C1, C2: two instances of the component *client*; and DBoracle: instance of the component *Database*; S1: instance of the component *server*; N1-1, N1-2: two instances of connector N1, and N2.1: instance of connector N2. These levels are illustrated in the following figure:

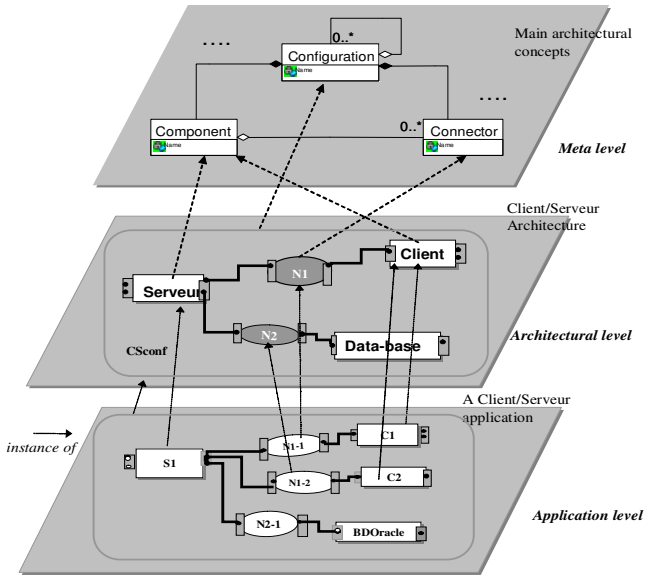


Fig. 1 Architectural elements abstraction level

SAEV can thus manage evolution of any level element

3.3. SAEV Specifications

To identify the concepts of the model SAEV, we have considered the following interrogations:

- What are the architectural elements involved?
- What are the different operations executed and on which architectural element?
- What are the impacts of this evolution and how to manage them?

Regarding these concerns, SAEV offers, by way of reification, concepts to describe and manage architectures' evolution. We present the SAEV meta-model, before explaining each concept.

1) SAEV Meta model

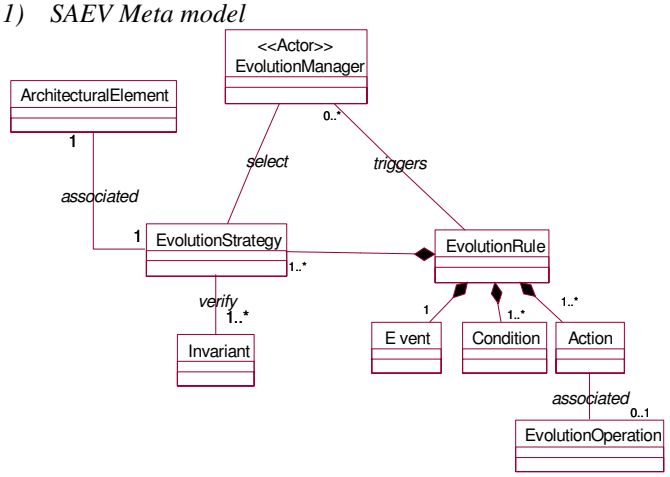


Fig. 2 SAEV Meta_model

SAEV associates to each architectural element an evolution strategy. A strategy gathers all evolution rules of its architectural element. Each evolution rule illustrates how an operation is applied on an architectural element and proposes the necessary propagations. We detail each concept hereafter:

2) SAEV concepts

a) Architectural element

This concept represents each evolving element of each abstraction level. So, at the *meta level*, an architectural element can be the concept "Component", at *the architecture level*, it can be the component "Client", at the *application level*, it can be the "Client X".

b) Invariant

An invariant represents a constraint of an architectural element, which must be respected, whatever is evolution operation applied on it. Any change in the architecture must maintain the correctness of this invariant. At each level we can define different invariants. For example, at the *meta level* we define the following structural invariants of *Configuration:*

I1- a configuration must be composed at least of one component;

I2- a provided port of configuration-interface must be connected at least to one provided port of its component;

The architectural elements of the *architectural level* must respect the invariants defined at the *meta level*. Those of the *application level* must respect the invariants of the *meta* and the *architectural* levels. In addition, each level can also define its own invariants. For example, if we consider the *client/ server* architecture (section 3.2), we can define invariants relative to client/server style: the type of the connector between the component *client* and the component *server* must be *RPC*.

c) Evolution operations

We consider each operation that can be applied to a given architectural element and that can cause its evolution, or the evolution of its architecture, as an evolution operation. Regarding architecture evolution needs, we identify following evolution operations: Addition, Deletion, Modification, and Substitution. Generally, evolution of software architecture is caused by the execution of one or more evolution operations on its architectural elements. We present some examples of evolution operations associated respectively to a configuration and its interface.

- Addition/deletion/ substitution/ of component or connector;
- Addition/ deletion/ substitution/ modification of a provided port or service;

d) Evolution Rule

An evolution rule describes the evolution of a given architectural element and the impacts of this evolution. Each evolution is invoked by an event, which represents the evolution need, and to execute this evolution rule a set of conditions must be satisfied.

We use the ECA formalism (Event /Condition /Action) to describe evolution rules. Each rule is described by:

- An event: is the evolution invocation.
- One or more conditions: that must be satisfied to execute the action part of the evolution rule.
- One or more actions, an action can be an:
  - Event, in this case, it will trigger another rule. Its role is the propagation of the impact.
  - Elementary action which is the invocation of an operation on an architectural element. We note them: *Arch-elt-name.Execute.operation-name(parameters).*

SAEV proposes a set of evolution rules but also offers possibilities to the designer to create its own evolution rules. For each level, it is possible to define different evolution rules. They will be used for the evolution of the architectural elements of its lower level.

Here is an example of evolution rule defined at the meta level.

R1: **deletion of component**
Event : delete-component(**Cf**: Config, **C**: comp); **Condition:** **C** ∈ comp(**Cf**), provided-interface(**C**) connected to provided-interface(**Cf**), ∃ NC⊂ connect(Cf) and ∀ **N**∈ NC **N** is connected to **C** and **N** is not charred **Action:** For **N** ∈ NC delete-connector (**Cf**,N) For b ∈ bindings(**Cf**,**C**) delete-binding(Cf,C,b) For I∈ interface-comp(**C**) delete-interface-comp(Cf,C,I) *C.Execute-delete-component(**Cf, C**)*

Table 2: Example of evolution rule

The rule **R1** describes the deletion of the component **C** belonging to the configuration **Cf,** given as a parameters.

The condition part of this rule expresses that the component **C** is positioned at the extremity of the configuration (provided-interface(**C**) connected to provided-interface(**Cf**)). It expresses also that all connectors attached to this component are not charred (∃ NC⊂ connect(Cf) and ∀ **N**∈ NC **N** is connected to **C** and **N** is not charred).

The action part of R1 is composed of three events and one elementary action.

- The first event triggers the rule of deletion of connector, to delete all connectors connected to the component **C.**
- The second event triggers the rule of deletion of bindings, to delete all bindings between the configuration **Cf** and the component **C**.
- The third event triggers the rule of deletion interface-component to delete the interface of the component **C**.
- The last is an elementary action which invokes the execution of the deletion of the component **C**.

e) Evolution strategy

An evolution strategy is defined for each architectural element. It gathers all evolution rules associated with each operation, which can be applied to this architectural element. These rules can be already defined by SAEV as well as they can be newly defined by the designer. We present an example of strategy **S1**: Evolution strategy of configuration.

Strategies	**A. Elements**	**Evolution operations**	**Evolution rules**
		Addition	R1, R10
		Deletion	R2, R3
		Substitution	R6, R9

Table 2. Configuration evolution strategy

The strategy S1 gathers the evolution rules associated with the configuration evolution operations. Details of each rule is note presented in this article. They are all defined like the presented rule R1.

f) Evolution manager

The evolution manager, as an actor, has to intercept events coming from the designer or the evolution rules. Then it execute corresponding evolution rules, according to the evolution strategy associated with an architectural element. We detail this process hereafter:

3) SAEV Evolution mechanism

Evolution mechanism describes the execution process to carry out a given evolution. We describe this process using UML sequence diagram [5] (fig 4).

The evolution is triggered for each incoming event. The evolution manager:

- **1:** intercepts this event;
- **2:** and **3:** selects the evolution strategy associated with the architectural element invoked by the event ;
- **4**: and **5:** selects in this strategy the evolution rule that correspond to the event and which have its satisfied conditions (for restriction space reason we don't present sequences which test the event and evaluate the conditions);
- **6: , 7:** and **8:** triggers the execution of the action part of the selected rule. Two cases can arise:
  - if the action corresponds to an event, the manager intercepts it also and follows the same precedent steps (**1:** to **8:**).
  - if it corresponds to an elementary action, it triggers then its execution.

- At the end of all evolutions, SAEV carry out the verification of the invariants. If inconsistencies are detected, it alerts the designer, who can correct them. In the contrary case, SAEV will cancel all executed evolutions.

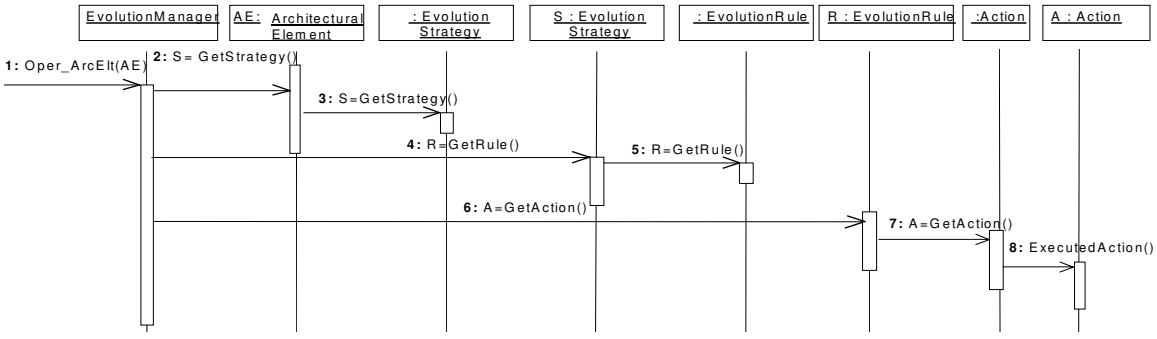


Fig. 3 SAEV evolution mechanism

Figure 3 illustrates the execution process of a given evolution without any conflict. Thus implies that at the step **4** of the process, SAEV selects only one evolution rule. But, others cases can occur, for example SAEV can selects for the same event more evolution rules. So, it must provide all the eventual evolutions according to the evolution rule

considered. The designer will validate one of these evolutions (these is not yet supported but represent an important perspective of our work).

4. A projection of SAEV on UML2.0

For reflexivity purpose, we describe SAEV as a component-based architecture. Thus, SAEV can be used to manage its own evolution. For example, by the addition of new evolution mechanisms, or by the modification of it's already proposed mechanisms. Each concept of SAEV is represented as a component and each association as a connector. SAEV is thus regarded as a configuration. This configuration, each of its components and connectors has its own interface.

To specify the architecture of SAEV we have chosen the ADL UML2.0 [17]. The main benefits of this choice are:

- UML is a standard for the analysis and design of software systems, UML and has many tools for its implementation, e.g. Rational Rose, Microsoft visual studio, Poseidon for UML.
- UML2.0 does not provide any support for software architecture evolution, so SAEV can be used as an evolution support for UML2.0 architectures.

We have used this SAEV's specification on UML2.0 to obtain a prototype of SAEV in the JAVA language. For that, we have defined mapping between UML 2.0 concepts and those of JAVA language (not presented for space restriction reason).

5. Conclusion and perspectives

We have presented SAEV, a model for software architecture evolution. It allows the description and management of the evolution on three abstraction levels (meta, architectural and application levels). To each architectural element, SAEV associates an evolution strategy, which gathers all its evolution rules. Evolution rules describe all evolution operations that can be applied to this element. Evolution rules must respect architectural elements invariants to maintain the architecture coherence. SAEV answers the most objectives fixed in section 3: it considers the evolution as an explicit problem in software architecture. It treats this problem independently of the Architecture Description Language by proposing uniform mechanisms for the evolution of each element and at each abstraction level.

As perspectives to this work, notably to face the dynamic evolution, we introduce a new evolution operation, which is the *versioning*. Like with the others operations, versioning can concern any architectural element. We will define then its strategies and evolution rules. Until here we have outlined only how an impact of a given evolution is propagated at the same level. We aim to study the impacts of a given evolution through different levels, and to manage unsolved situations.

REFERENCES

[1] R. Allen : A formal Approach to software Architecture Description, PHD thesis, Carnegie Mellon University Center, Dr. Minneapolis, MN, April 1996.

[2] R. Allen, R. Douence, D.Garlan: Specifying and analyzing Dynamic Software Architectures, In proceeding of the 1998 Conference on Fundamental Approaches to Software Engineering(FSE'98), Lisbon, Portugal, vol 1382 of lecture Note in computer sciences, pp 2137, spring verlag 1998.

[3] L F. Andrade, J.L. Fiadeiro: Architecture based evolution of software systems, LNCS 2804 : 148-181, 2003.

[4] T. Batista, A. Joolia, G. Coulson: Managing Dynamic Reconfiguration in Component based systems, in European workshop on software architecture, Pisa, Italy, June 2005.

[5] I.Crnkovic, M.Larsson : Challenges of component-based development, in the journal of Systems and Software 201-212, 2002.

[6] F. Duclos, J.Estublier and R. Sanlaville : Opened Architecture to software adaptation, software engineering, review N° 58, September 2001.

[7] W.B. Frakes, A case study of a reusable component collection in the information retrieval domain, The Journal of Systems and Software, pp 265-270, 2004.

[8] D. Garlan, R. Monroe, D. Wile: ACME: Architectural Description Of Components-based systems, Leavens Gary and Sitaraman Murali, Foundations of component-Based Systems, Cambridge University Press, 2002,pp. 47-68.

[9] D. Garlan, D. Perry: introduction to the special issue on software architecture, IEEE Transactions on Software Engineering, 21(4), April 1995.

[10] R. Land: An architectural approach to software evolution and integration, Licentiate thesis, ISBN 91-88834-09-3, Barcelona, Spain, September 1995.

[11] M. Jazayeri: On Architectural Stability and Evolution. Lecture Notes in Computer Science, Springer Verlag, Berlin (2002)

[12] D. Luckham, M. Augustin, J. Kenny, J. Vera, D. Bryan, W. Mann: Specification and analysis of System Architectures using Rapide", IEEE Transaction on Software Engineering, vol.21, N° 4, April 1995, pp 336-355.

[13] J. Magee, N Dulay, S. Eisenbach, J. Kramer : Specifying Distributed Software Architectures, In Proceedings of the fifth European Software Engineering conferenc, Sitges Span , pp 137-154, 1995.

[14] N. Medvidovic, D.S.Rosenblum, and R.N. Taylor: A Language and Environment for Architecture-based Software Development and evolution. In proceeding of the 21st international conference en Software engineering, pp.44-53, may 1999.

[15] N. Medvidovic, D.S.Rosenblum: A Domains of Concern in Software Architectures and Architecture Description Languages. In Proceedings of the 1997 USENIX Conference on Domain-Specific Languages, pp 199-212 October 15-17, Santa Barbara, California.

[16] N. Medvidovic, R. N. Taylor : A Classification and Comparison Framework for Software Architecture Description Languages, IEEE Transactions on Software Engineering, Vol. 26, 2000.

[17] OMG, UML 2.0 infrastructure specification, Technical Report ptc/03-09-15,Object Management Group (2003).

[18] P. Oreizy: Issues in the runtime modification of software architectures. Technical Report, UCI-ICS 96-35, University of California, Irvine, August 1996.

[19] M. Oussalah, Changes and Versioning in complex objects, International Workshop on Principles of Software Evolution, IWPSE 2001, Sep. 10 & 11, Vienna University of Technology, Austria.

[20] R. Roshandel, A.V.D. Hoek, M. Miki-Rakic, N. Medvidovic : Mae-A System Model and Environment for Managing Architectural Evolution : ACM Transactions on Software Engineering and Methodology, vol. 13 N° 2 pp 240-276, April 2004.

[21] M. Shaw, R. Deline, D. V. Klein, T.L. Ross, D. M. Young and G. Zelesnik: Abstractions for Software architecture. Transactions on Software Engineering, page 314-335, April 1995.

Application of Execution Pattern Mining and Concept Lattice Analysis on Software Structure Evaluation

Kamran Sartipi and Hossein Safyallah
Dept. Computing and Software, McMaster University
Hamilton, ON, L8S 4K1, Canada
{*sartipi, safyalh*}*@mcmaster.ca*

Abstract

Software maintenance activities for producing a feature-rich system tend to impair the software's structure into an unshaped and cost-prone legacy system. Thus, it is desirable to keep track and measure the impacts of the newly added features on the structure of the software system. The proposed technique in this paper is based on extracting frequent patterns in the execution traces of a software system using a pattern discovery technique. The patterns represent functionalities that correspond to the feature specific scenarios. In a further step, the generated execution patterns are distributed on a concept lattice to separate feature specific patterns from commonly used patterns. The proposed technique allows for assigning software features onto the software system modules and provides a means for assessing the degree of functionality scattering among the system modules. Consequently, we measure the impact of individual features on the structure of the system. A case study on the Unix Xfig drawing tool is used to present the accuracy of the approach.

KEYWORDS: Software Maintenance; Dynamic Analysis; Execution Pattern Mining; Concept Lattice; Feature.

1. Introduction

Software systems are continuously evolving throughout their life time from early development to their maintenance and retirement. During the maintenance phase the software system is still changing through activities such as bug- xing, migration to new platforms, and adding new features which were not planned from the beginning. Therefore, even a nicely designed and accurately implemented software system will probably incur several changes to its functionality and consequently to its structural design. This common scenario is the main cause of structural damage, high maintenance cost, and eventually retirement of a legacy system. To help this situation, the task of the software maintainers is to measure the impair on the structure of the software system and assess the current state of the resulting legacy system. In this paper, we propose a novel approach to assess the structural merit of the software system based on the degree of functional scattering of software features among the structural modules. In the proposed approach, the functionality of the system is represented as a set of features that are implemented within the software modules and are manifested as constituents of different scenarios to be run on the software system.

The proposed approach takes advantage of dynamic analysis, data mining technique sequential pattern discovery, and concept lattice analysis to provide comprehensive information about the software system from different aspects. A key characteristic of this approach is to identify a mapping between the implementation of the software features and the structure of the system to assess the impact of a feature on the structure of the system. We execute a set of carefully de ned scenarios that are speci c to certain software features in order to reveal the realization of the scenario functionality within the software system modules. In this paper, we propose a novel approach to dynamic analysis and structural evaluation of a software system that contributes to the software maintenance eld by the followings: i) identi cation of the set of core functions that implement both speci c features and commonly used features of a software system; ii) providing a measure of scattering of the feature functionality to the structural modules as well as a measure of cohesion for a structural module; and iii) visualizing the functional distribution of speci c features on a lattice using concept lattice analysis. The approach uses techniques such as sequential pattern discovery from data mining domain, loop-based redundant trace elimination from string processing eld, as well as the visualization power of the concept lattice analysis, to automatically extract the functionality of the speci c features as well as the common features of a software system. The proposed structural assessment directly represents the cohesion of module(s) im-

plementing a speci c feature; this measure of cohesion is much closer to the original de n ition of cohesion ("relative functional strength of a module" [11]) than using static structural techniques such as inter-/intra-edge connectivity of the components.

The remaining of this paper is organized as follows: Section 2 describes the related work. In Section 3 the proposed framework for dynamic analysis is presented. Section 4 outlines the data cleansing process. Section 5 presents an overview of the techniques used in our approach. Section 6 describes the execution pattern analysis and structural evaluation in more detail. In Section 7 X g system is analyzed as the case study. Finally, the paper concludes in Section 8.

2. Related Work

In this section, we brie y present the closest related approaches and compare them with our work.

The applications of concept lattice analysis in software engineering have been studied for both static and dynamic analyzes. In static analysis, C. Lindig et al. [10] and Siff et al. [12] have applied concept lattice analysis in order to modularize legacy software systems; where the relationships between procedures and global variables in legacy code are investigated to extract and evaluate the candidate modules. In contrast to the above concept lattice analyzes, our approach exploits dynamic behavior of a software system to evaluate its structural properties. In dynamic analysis approaches using concept lattice, Eisenbarth et al. [6, 7] introduced a formal lattice to locate computational units that implement a certain feature of the software system. Similarly, we use the relation between scenarios and computational units (i.e., functions, methods, procedures) that are invoked during the scenario execution. However, our technique reduces the number of invoked computational units in each scenario execution to those reside in the execution patterns in order to relax the complexity of the resulting lattice.

In a different context, El-Ramly et al. [8] apply sequential pattern mining on user-interaction traces of the system, in order to reveal interaction patterns between graphical user interface components.

The following approaches use program slicing techniques to support software maintenance activities. Gallagher et al. [9] and Beszedes [3] apply (static and dynamic) program slicing in order to isolate the effects of a change in the software source code. The method allows maintainers to identify the statements and variables that may require modi cations as a result of this change.

Overall, in the proposed technique we amalgam the advantages of data mining techniques with mathematical concept lattice to explore non-trivial feature-based execution patterns; consequently we identify individual modules that implement software features as a means to measure the module's structural merit.

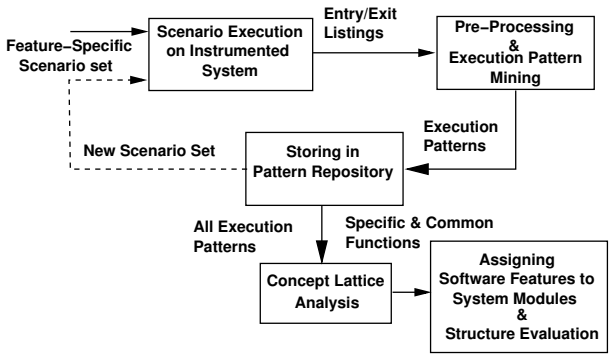


Figure 1. Proposed framework for assigning software feature onto the system modules.

3. Proposed Framework

Figure 1 illustrates different steps of the proposed framework for assigning software's feature-functionality onto the system modules. The framework takes advantage of the relation discovery power of *data mining* and *concept lattice analysis* and provides a means for measuring the impact of individual features on the structure of the system. The operations in the framework of Figure 1 are as follows.

STEP 1: extracting execution traces. Important features of a software system are identi ed by investigating the system's user manual, on-line help, similar systems in the corresponding application domain, and also user's familiarity with the system. A set of relevant task scenarios are selected that share a single software feature. We call this set of scenarios as *feature-specific scenario set*. For example, in the case of a drawing tool software system, a group of scenarios that share the "move" operation to move a gure on the computer screen would constitute such a feature-speci c scenario set. In the next step, the software under study is instrumented [1] to generate text messages at the entrance and exit of each function execution. By running the group of feature-speci c scenarios on the instrumented software system a set of unprocessed *entry/exit listings* of function invocations are generated. In order to make the large size of the generated traces manageable, in a *preprocessing* step we transform the extracted entry/exit listing into a sequence of function invocations and also remove all redundant function calls caused by the cycles of the program loops. The trimmed execution traces are then fed into the execution pattern mining engine in the next stage. The preprocessing operation will be discussed

[1] Instrumentation refers to inserting particular pieces of code into the software system (source code or binary image) to observe its runtime behavior.

in more details in Section 4.

STEP 2: execution pattern discovery. In this step, we reveal the common sequences of function invocations that exist within the different executions of a program that correspond to a set of task scenarios. We apply a *sequential pattern mining* algorithm on the execution traces to discover such hidden *execution patterns* and store them in a *pattern repository* for further analysis. This operation will be discussed in more details in Section 5.1. Each execution pattern is a candidate group of functions that implement a common feature within a scenario set. We employ a strategy to spotlight on execution patterns corresponding to speci c features within a group of scenario sets. This is performed by identifying those execution patterns that are speci c to a single software feature within one scenario set (namely *feature-specific patterns*). Similarly, we identify the execution patterns that are common among all sets of scenarios (namely *common patterns*). Each execution pattern has signi can ce in localizing an important feature of the subject software system. However, even for a speci c feature, a large group of execution patterns are generated that must be organized (and some must be ltered out) to identify core functions of a feature. Concept lattice is an ideal tool for such a task, hence we use the visualization power of concept lattice to generate clusters of functions within feature-speci c patterns and common patterns. Finally, by associating the functions of the generated clusters to the system's structural modules, i.e., les of the system, and applying two metrics for measuring *module cohesion* and *feature functional scattering*, we measure the impact of individual features on the structure of the software system. This operation is discussed in Section 6.

4 Preprocessing

Dynamic analysis of a medium size software system using execution traces can produce very large traces ranging to thousands or tens of thousands of function calls. This would be a main source of dif culty in a dynamic analysis. Therefore, before using the extracted entry/exit listing in further stages, redundancies in a trace that are produced by program loops and recursive function calls should be eliminated. For the analysis in this paper, we ignore recursive function traces and focus on pruning the loop-based redundancies.

In doing this, we transform the *entry/exit listing* (discussed in section 3) that is generated by instrumenting the software system (using *Aprobe* tool [13]) into a dynamic call tree, where nodes represent functions and edges represent function calls. Since each loop resides in the body of a function, the loops will form identical subtrees as the children of the parent function.

In this context, the loop redundancy removal problem is reduced to identi catio n of identical subtrees that are repeated under a particular node. In order to nd the repetitions that exist among subtrees of a given node, we rst label these subtrees with unique IDs, where identical subtrees possess identical IDs. We then generate a string from IDs of these sibling subtrees. By applying a repetitive string nder algorithm (*Crochemore* [5]) we represent the original string (with repetitions) in the form of a new string with no repetitions. In this new string, each string repetition is shown as one instance of the repeated items that is labeled with the number of the repetitions. For example, in Figure 2(a) the string *F1,F2,F1,F2, ..., F1, F2* is transformed into a string with no repetition $(F1,F2)^n$ in Figure 2(b).

As a result, we keep only a subtree (among similar subtrees) that correspond to a single instance of each loop, which greatly reduces the complexity of the dynamic call tree. Finally, by traversing the loop-free dynamic call tree in a depth- rst order and keeping the visited nodes in a sequence, a loop-free execution trace is generated.

```
Procedure Foo
begin
    Call F1;
    while condition do
        Call F1;
        Call F2;
    end
end
```

$$\frac{\ldots, Foo, F1, F1, F2, F1, F2, \ldots, F1, F2, \ldots}{\text{(a)}}$$

$$\frac{\ldots, Foo, F1, (F1, F2)^n, \ldots}{\text{(b)}}$$

Figure 2. (a) A trace generated from Procedure Foo. (b) Loop free representation of (a).

In Procedure Foo a piece of code that produces a long trace with repetitions of "*F1, F2*" is shown. Figures 2(a) and 2(b) represent the parts of execution trace that is produced by Procedure Foo, and the result of applying Crochemore algorithm, respectively.

5. Techniques for exploring patterns

In the following subsections, we brie y present the application of sequential pattern mining and mathematical concept lattice analysis. The former is used to extract highly repeated execution patterns and the later is applied on the

extracted execution patterns in order to cluster the functions that exist within common / feature-specific patterns.

5.1. Execution Pattern Mining

A major characteristic of the run time analysis of a software system is generating execution traces with large sizes that make the task of analysis a daunting one. The effective function-trace of an intended scenario is cluttered by a large number of function calls from various places such as initialization / termination operations, utilities, and loop-based repetitions. In the rest of this subsection, we describe the application of a data mining technique to discover sequences of functions in a software system that correspond to certain system features. In the data mining literature, *sequential pattern mining* is used to extract frequently occurring patterns among the sequences of customer transactions [2]. In this context, the sequence of all transactions corresponding to a certain customer (already ordered by increasing transaction-time) is known as a *customer-sequence*. A customer-sequence *supports* a sequence s if s is a subsequence of this customer-sequence. A frequently occurring sequence of transactions (namely a pattern) is a sequence that is supported by a user-specified minimum number of customer-sequences (namely $MinSupport$ of this pattern).

In this paper, we use a modified version of the sequential pattern mining algorithm by Agrawal [2], where an *execution pattern* is defined as a contiguous part of an execution trace (as a customer-sequence defined above) that is supported by $MinSupport$ number of execution traces. A typical sequential pattern mining algorithm allows extracting noncontiguous sequences of function calls. In most cases, this characteristic drastically increases the time/space complexity of the pattern mining algorithm and particularly complicates the dynamic analysis of a software system. In the presented approach, each extracted sequential pattern consists of solely a contiguous part of different execution traces. This strategy produces meaningful execution patterns that correspond to core functions implementing specific functionalities of the system. Whereas, extracting execution patterns that contain noncontiguous function invocations would generate an overwhelming number of meaningless patterns that consist of unrelated parts of the execution traces.

5.2. Concept Lattice Analysis

The *mathematical concept analysis* was first introduced by Birkhoff in 1940 [4]. In this formalism, a binary relation between a set of "objects" and a set of "attribute-values" is represented as a lattice. A *concept* is a maximal collection of objects sharing maximal common attribute-values. A *concept lattice* can be composed to provide significant insight into the structure of a relation between objects and attribute-values such that each node of the lattice represents a concept.

A concept lattice can be used to collect the set of shared attributes contained in a set of objects such that the shared attributes appear in the nodes that are located in the upper region of the lattice. Consequently, the nodes in the lower region of the lattice collect the attributes that are specific to the individual objects in that region. We exploit this property to group functions of the extracted execution patterns. The strategy to identify groups of shared attributes will be described in the next section.

6. Assigning Feature to Structure

In this section, we present the assignment of *software feature families* onto the software system modules, as a means of assessing the merit of the software structure. As it was discussed in the proposed framework in Section 3, we define and execute a number of feature-specific scenario sets where each scenario set targets a different software feature. Furthermore, we generate execution patterns by applying the sequential pattern mining algorithm on the loop-free execution traces. The generated execution patterns and their functions are then mapped onto a concept lattice in order to identify clusters of functions, where each cluster corresponds to a family of related software features. In the remaining of this section, we describe pattern analysis aspects and software structure assessment of the proposed approach.

Scenario Model

In the context of this paper, we define a scenario as a sequence of relevant features of a software system, where:

- *feature* ϕ is a unit of software requirements that describes a single functionality in the software system under study. Φ is the set of all available features.
- *feature family* Φ_ϕ is a set of semantically relevant features to specific feature ϕ in the subject software system that are defined towards similar functionalities.
- *scenario* s is a sequence of features $\phi \in \Phi$; thus $s = [\phi_1, \phi_2, \ldots, \phi_n]$. Also $\mathcal{S}$ is the set of all applicable scenarios on the system.
- *feature-specific scenario set* $\mathcal{S}_\phi$ is a set of scenario s that uses specific feature ϕ; thus $\mathcal{S}_\phi = \{s \mid s \in \mathcal{S} \ \wedge \ \phi \in \textit{setOf}^2(s)\}$.

[2] setOf(s) denotes to the set representation of sequence s

An execution pattern is treated as a sequence of functions that implement common feature(s) within a scenario set. In the following, the different kinds of execution patterns that may exist in the execution of a group of feature-speci c scenario sets along with the corresponding extraction mechanisms are presented.

Feature-specific execution pattern
A feature-speci c execution pattern corresponds to the core functions that implement a targeted feature ϕ of a scenario set $\mathcal{S}_\phi$ (e.g., drawing a speci c gure such as rectangle). Such a pattern exists in the majority of a feature-speci c scenario-set $\mathcal{S}_\phi$. In order to extract a feature-speci c pattern, we should increase the level of $MinSupport$ of the generated execution patterns to a number that covers the majority of the scenarios in $\mathcal{S}_\phi$.

Omnipresent execution pattern
An omnipresent execution pattern is common to almost every task scenario of the software system (e.g., software initialization / termination operations, or mouse tracking). Such a pattern exists in every trace of every scenario-set $\mathcal{S}_\phi$. In order to extract such a pattern, we should use a lter ing mechanism (concept lattice in section 5.2) to lter out the feature-speci c patterns from this group of patterns.

Concept Lattice Analysis
We employ a strategy to spotlight on execution patterns corresponding to speci c features within a group of scenario sets. In this context, we use concept lattice analysis to cluster the group of functions in patterns that exclusively correspond to a shared feature of a scenario set; also to cluster the group of functions in patterns that are common to every scenario set. A key property of the functions in both kinds of clusters is that these functions belong to shared *contiguous sequences* of function calls not to the shared scattered function calls within scenario set executions. In our setting for concept lattice analysis, an object is a targeted feature $\phi \in \Phi$ of a feature-speci c scenario set $\mathcal{S}_\phi$, and an attribute is a function f that participates in the execution patterns within $\mathcal{S}_\phi$.

In this context, the omnipresent function clusters appear in the upper region of the lattice, and feature-speci c function clusters are collected by nodes in the lower region of the lattice. In the following, we de ne the group of concepts that are relevant to feature-speci c clusters.

- *feature-specific concept* c_ϕ is a concept whose support consists of a single feature ϕ. We de ne F'_ϕ to be the set of functions corresponding to c_ϕ on the lattice.

- we de n e F_{Φ_ϕ} to be the set of functions that implement the feature family Φ_ϕ. Thus:

$$F_{\Phi_\phi} = \bigcup_{\varphi \in \Phi_\phi} F'_\varphi$$

where F_{Φ_ϕ} represents a software system *logical module* that implement the core functionality of a feature family Φ_ϕ.

6.1. Structural Cohesion and Functional Scattering

In this section, we assess the degree of distribution of collected functions of logical module F_{Φ_ϕ} over the structure of the system as a means for evaluating the feature functional scattering among software modules. Moreover, we assess the degree of concentration of logical module functions F_{Φ_ϕ} within a speci c software module (e.g., le) to evaluate the software module's cohesion with respect to feature family Φ_ϕ. These two measures provide a mechanism to evaluate the impact of individual features on the structure of the software system.

In the following, $SC_{\Phi_\phi}(m)$ denotes structural cohesion of module m with respect to logical module F_{Φ_ϕ} and $FS(\Phi_\phi)$ denotes functional scattering of feature family Φ_ϕ:

- Let $M_{\Phi_\phi} = \{m_1, m_2, \ldots, m_k\}$ be the set of modules where all the functions in F_{Φ_ϕ} are de ned in elements of M_{Φ_ϕ}.

- Let F_m denote the set of functions that are de ned in module m.

- Structural Cohesion $SC_{\Phi_\phi}(m)$ of module m with respect to logical module F_{Φ_ϕ} is de n ed as:

$$SC_{\Phi_\phi}(m) = \frac{|F_m \cap F_{\Phi_\phi}|}{|F_m|}$$

- Functional Scattering $FS(\Phi_\phi)$ of feature family Φ_ϕ is de ned based on the distribution of functions in F_{Φ_ϕ} over modules in M as:

$$FS(\Phi_\phi) = 1 - \frac{\sum_{m \in M_{\Phi_\phi}} SC_{\Phi_\phi}(m)}{|M_{\Phi_\phi}|}$$

A software system with high structural cohesion $SC_{\Phi_\phi}(m)$ for its individual modules and low functional scattering $FS(\Phi_\phi)$ among its structure represents a modular system that requires less maintenance efforts. However, a high degree of functional scattering corresponding to a feature family Φ_ϕ directly signi es a high structural impact that is caused by that feature family. Hence the system requires more maintenance efforts to tackle with the consequences of propagated change to other software modules.

7. Case Study

In this section, we discuss the result of applying the proposed dynamic analysis technique on X g 3.2.3d [1]. X g is an open source, medium-size (80 KLOC), menu driven, C language drawing tool under X Window system. X g has the ability to interactively draw and manipulate graphical objects (circle, ellipse, line, spline) through operations such as copy, move, edit, scale, and rotate. In order to extract the core functions that implement a speci c feature (e.g., Circle-Diameter in Table 2) we de ne a group of feature-speci c scenarios to target this feature and execute on the instrumented X g system to obtain the corresponding execution traces. After pruning the loop-based function calls (section 4) we apply the execution pattern mining process to obtain the existing execution patterns. We repeat the above process for each member of a feature-family in order to collect the corresponding execution patterns. Table 2 presents the statistical information for two feature-families of X g .

In a further step, we supply the resulting execution patterns to a concept lattice generation tool. Viewing the distribution of the concepts and their functions throughout the concept lattice allows to get insight into the structure of the feature-speci c concepts and their functions. Consequently, it allows us to collect the group of functions that correspond to different feature-families. Finally, based on inspecting the source les of X g, we measure the structural cohesion of corresponding source les, as well as the feature functionality scattering of the feature families under study. The results of this evaluation are presented in Table 3.

In the followings, we discuss the important properties of the proposed pattern based dynamic analysis technique using the X g case study.

Mapping logical modules onto structural modules. Table 1 demonstrates the results of experimentation with X g tool to reveal the core functions for two X g feature families. We focus on drawing a gure in the ellipse family (including circle and ellipse) such that each g ure can be drawn in two different ways, i.e., by-radius and by-diameter. Furthermore, we expand our experiments on editing operation of the X g tool (i.e., copy graphical objects). The extracted logical modules are shown in Table 1 and according to the X g naming convention it is clear that the logical modules truly re ect the core functions of the feature families.

Focusing on the important sub-traces. Table 2 represents the attributes of a group of feature-speci c scenario sets that we use in the analysis process. This table illustrates a major characteristic of the proposed dynamic analysis with regard to reducing the scope of the analysis from huge sizes of the execution traces (Average Trace Size) to the manageable sizes of the execution patterns (Average Pattern Size).

Feature Family	**Extracted Core Functions representing logical module F_{Φ_ϕ}**
Ellipse	init_circlebyradius_drawing, elastic_cbr, resizing_cbr, create_circlebyrad, circlebyradius_drawing_selected, init_circlebydiameter_drawing, elastic_cbd, resizing_cbd, create_circlebydia, circlebydiameter_drawing_selected, init_ellipsebydiameter_drawing, elastic_ebd, resizing_ebd, create_ellipsebydia, ellipsebydiameter_drawing_selected, init_ellipsebyradius_drawing, elastic_ebr, resizing_ebr create_ellipsebyrad, ellipsebyradius_drawing_selected, add_ellipse, pw_curve, create_ellipse, center_marker, draw_ellipse, redisplay_ellipse ellipse_bound list_add_ellipse set_latestellipse toggle_ellipsemarker list_delete_ellipse
Copy	copy_selected init_copy init_arb_copy set_lastlinkinfo init_arb_move init_move move_selected set_lastposition set_newposition moving_line init_linedragging adjust_pos place_line translate_line adjust_links place_line_x

Table 1. Results of feature to code assignment for 2 features of Xfig drawing tool.

Structural evaluation. For each feature family Φ_ϕ in Table 2 we inspect the X g source les that de ne the functions that implement the corresponding logical module F_{Φ_ϕ} of that feature family. The results of measuring the structural cohesion $SC_{\Phi_\phi}(m)$ of these les are presented in Table 3. These results indicate that le d_ellipse has high cohesion with respect to logical module of feature family *Ellipse* and les e_copy, and e_move are also highly cohesive with respect to feature family *Copy*. However, study of the feature functional scattering measures allows us to better interpret the characteristics of these logical modules. For example, in the case of *Ellipse* a portion of the logical module F_{Φ_ϕ} is located in a large structural module u_elastic which results in a high functional scattering measure. Whereas, in the case of *Copy* feature family, the logical module almost covers two structural modules e_copy and e_move which indicates low scattering.

We also adopt a minimum threshold value of 10% in order to consider a structural module in the calculation of the above measurements. The results in Table 2 are promising in the sense that they re ect meaningful measures with respect to the sizes of logical and structural modules shown. Regarding the results of our structural evaluations, we can predict high maintenance activities for any change to the *Ellipse* feature family. Similarly, changes to the *Copy* feature family would not propagate throughout the system which indicates less maintenance activities.

Feature Family	Specific Feature of Xfig	Number of Different Scenarios	Average Trace Size	Average Pruned Trace Size	Number of Extracted Patterns	Average Pattern Size
Draw Ellipse	Circle-Diameter	10	7234	2600	46	33
	Circle-Radius	10	8143	2463	48	32
	Ellipse-Diameter	10	6405	2536	41	37
	Ellipse-Radius	10	7351	2549	39	35
Copy	Move Objects	4	11887	3166	31	53
	Copy Objects	4	11460	3269	37	50

Table 2. Results of execution trace extraction and execution pattern mining for 2 Xfig feature families.

Feature Family Φ_ϕ	Contributed File (m)	$\lvert F_m \rvert$	$\lvert F_m \cap F_{\Phi_\phi} \rvert$	Structural Cohesion $SC_{\Phi_\phi}(m)$	Functional Scattering $FS(\Phi_\phi)$
Ellipse	d_ellipse.c	16	12	75%	57%
	u_elastic.c	67	8	12%	
Copy	e_copy.c	5	3	60%	32%
	e_move.c	4	3	75%	

Table 3. Structural cohesion and feature functional scattering measures for 2 Xfig feature families.

8. Conclusions

In this paper, we proposed a novel approach to dynamic analysis and structural assessment of a software system that takes advantage of repeated patterns of execution traces that exist within the executions of a set of carefully designed task scenarios. The proposed approach bene ts from the discovery nature of data mining techniques and concept lattice analysis to extract both feature speci c and common groups of functions that implement important features of a software system. The resulting execution patterns provide discovery of valuable information out of noisy execution traces. The proposed approach is centered around a set of task scenarios that share a speci c system feature and introduces a means for measuring the impact of individual features on the structure of the software system. The proposed technique has been applied on a medium size interactive drawing tool with very promising results in extracting both feature speci c and common functions. Moreover, the level of "structural cohesion" and "feature functional scattering" are measured that provides a way for assessing the structure of the experimented tool.

References

[1] X g version 3.2.3. http://www.x g .org/.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In Proceedings of *ICDE '95* , pages 3–14, 1995.

[3] A. Beszedes. Union slices for program maintenance. In Proceedings of *ICSM '02*, pages 12–21, 2002.

[4] G. Birkhoff. *Lattice Theory*. American Mathematical Society, 1st edition, 1940.

[5] M. Crochemore. An optimal algorithm for computing the repetitions in a word. *Information Process Letters*, 12(5):244–250, 1981.

[6] T. Eisenbarth, R. Koschke, and D. Simon. Derivation of feature component maps by means of concept analysis. In Proceedings of *CSMR '01*, pages 176–179, 2001.

[7] T. Eisenbarth, R. Koschke, and D. Simon. Locating features in source code. *IEEE Transactions on Software Engineering*, 29:210 – 224, March 2003.

[8] M. El-Ramly, E. Stroulia, and P. Sorenson. Recovering software requirements from system-user interaction traces. In Proceedings of *SEKE '02:*, pages 447–454, 2002.

[9] K. B. Gallagher and J. R. Lyle. Using program slicing in software maintenance. *IEEE Trans. Software Eng.*, 17(8):751–761, 1991.

[10] C. Lindig and G. Snelting. Assessing modular structure of legacy code based on mathematical concept analysis. In Proceedings of *ICSE '97*, pages 349–359, 1997.

[11] R. S. Pressman. *Software Engineering, A Practitioner Approach*. McGraw-Hill, third edition, 1992.

[12] M. Siff and T. W. Reps. Identifying modules via concept analysis. In Proceedings of *ICSM '97*, pages 170–179, 1997.

[13] OC. Systems. Aprobe version 4.2 for unix, 2003.

A Practical Quality Model for Evaluating Business Components[1]

Ji Hyeok Kim, Sung Yul Rhew, and Soo Dong Kim

Department of Computer Science
Soongsil University
1-1 Sangdo-Dong, Dongjak-Ku, Seoul, Korea 156-743
jhkim@otlab.ssu.ac.kr, {syrhew,sdkim}@ssu.ac.kr

Abstract

A business component is an implementation of common functionality among family members in a specific domain. Therefore, business components should be developed with common features among them. Also, variability among members should be well modeled into components for customizability. Evaluating the quality of such business components is an important prerequisite to successful component-based development (CBD) projects. In this paper, we propose a quality model that can be used practically to evaluate business components. We first identify characteristics of business components, and then derive a quality model that consists of characteristics, subcharacteristics, and associated metrics. By using the quality model, we believe that candidate business components can be evaluated quantitatively before they are purchased or acquired.

Key words: business component, quality model

1. Motivation

Component-Based Development (CBD) is gaining popularity in both academic and industry for its promise on large-scaled software reuse. Component is one of the recent and effective reuse approaches. Component is a concept of business component which is a software implementation of an "autonomous" business concept and process [1].

Therefore, various family members use the business components for developing the system. However, to use the business components, we evaluate the quality of business components. Since the business components provide the variability for various family members, evaluating the quality of business components becomes an important prerequisite to a successful component-based system development.

In this paper, we identify the unique features of business components. Then, we derive a quality model for business components from the unique features and extend C-QM [2] as our preliminary research.

The paper is organized as follows. Section 2 examines related works and discusses their limitations. Section 3 describes the unique features of business components. In section 4, we derive the quality attributes as characteristics and subcharacteristics, and section 5 defines metrics of the given quality attributes.

2. Related Work

2.1. Software Quality Model

A software quality model is a specification of software quality attributes and their relationship. ISO 9126 [3] is a representative quality model for generic software. In that, the quality model has two layers; characteristics layer and sub-characteristic layer. A characteristics is further refined into multiple sub-characteristics, and each sub-characteristic has a set of associated metrics, where a metric has a formula used to compute the metric value. However, this quality model was proposed for assessing the quality of finished software products.

2.2. Component Quality Model

Martin-Albo's work [4] proposed the component quality model. This work is based on ISO 9126. However, this work defined the only characteristics and subcharacteristics. Proposed metrics by this work might not measure the value. This work only defined the definition of metrics. Also, this work was not reflecting of component specific features since the work intends to apply all characteristics of ISO 9126 to component quality model.

M. Goulão's work [5] also proposed the component quality model. However, this work only proposed the quality attribute without definition of metrics.

[1] This work was supported by Korea Research Foundation Grant. (KRF-2004-005-D00172)

C-QM [2] was our preliminary research. This work was based on the features of COTS components and we defined its quality model which consists of characteristics, subcharacteristics, and metrics for evaluating COTS components. We derived 4 characteristics for C-QM such as functionality, reusability, maintainability, and conformance. Also, we derived 12 subcharacteristics and metrics for measuring the characteristics. However, scope of this work is limited as COTS components. Therefore, quality model of extended business component are needed.

3. Unique Features of Business Components

To derive a quality model, we should comprehend the unique features of business component. Hence in this section, we identify the inherent features of business component based on representative works on them [6][7][8][9][10].

Components provide high modularity. A component typically consists of multiple objects and classes. Hence, a component is a larger-grained unit than the object of OOP. A component is a coherent package in software development. Since a component can be independently developed and delivered, and has explicit and well-specified interfaces for the provide interface and require interface. Also, a component can be composed with other components, perhaps customizing some of theirs properties.

Components provide common features. A component should capture a common functionality among family members to increase inter-organizational reuse and commonality among organizations in a business sector. This feature makes the component-based reuse distinguishable from object-oriented reuse.

Components capture variability. A component should capture variability in domain to be used by the various family members. After the commonality analysis is performed, the variability analysis is performed. That is, the variability exists in the commonality. Also, the variability influences customization mechanisms. The variability is captured by applying to a various family members. The type of variation points may exist on logic, attribute or workflow.

Components are replaceable through contracts. A component should satisfy the contract between producers and consumers. This contract represents the interface such as provide interface and required interface. If the contracts between interfaces are satisfied then components can be composed with other component and can be replaceable other component that satisfy the standard interface. The composition mechanism is typically defined by its component reference model, and that mechanism is implemented by component platforms. Components integrated in an application should be replaceable with newer version of components efficiently without modifying other parts of the application.

Components are customizable. Commonality and variability (C&V) analysis is a key activity in extracting components from a model. A component should implement this variability and provide some mechanisms to customize for solving the variability by using selection, plug-in, external profile technique.

Components conform to reference model. A component implementation environment, i.e. component platform, implements a specific component reference model. Components developed on an EJB platform are not interoperable with components developed on .NET platform unless a specialized translator or gateway is used. Hence, component consumers should consider the component reference model of the target components, and how well they conform to the reference model.

4. Quality Model for Business Components

Based on the unique features of business components, we now define the quality model which consists of characteristics, their sub-characteristics, and metrics. Quality models should be defined in a way that the various aspects-of-interest for the target systems can be accurately measured. That is, quality models should be derived from the unique features of the target systems.

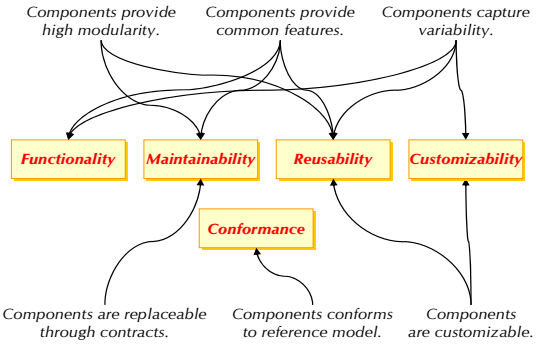


Fig. 1. Mapping Features to Characteristics

In our research, we also derive the quality model of the business components from the unique features we

identified in section 3. We derived five main characteristics of the quality model, as in

Fig. 1. For example, the quality characteristic *Reusability* is derived from the four features of business components; *high modularity, common features, variability,* and *customization mechanism*.

Functionality: This characteristic is to measure how a component provides functions and quality attribute (non-functional requirement) which meet the stated and implied requirements. Moreover the business component is a domain specific component. Therefore, this characteristic is the most important characteristic than others. In case of components, the functional and non-functional requirements are defined as the set of functions that are common to family members, called family requirement specification (FRS).

Reusability: Business components are target for specific domain, therefore business component should provide a highly reusability in same domain. The require interface of a component which realizes the variability from C&V analysis will yield a higher reusability since a broader range of family members may be able to reuse the component through the require interface and customization mechanism. The high modularity and larger-grained nature of component will also yield a higher reusability than object-oriented reuse.

Maintainability: The facts that components provide the high modularity, common feature, and replacement contribute the need for the maintainability characteristics. Software maintenance involves correction, enhancement, adding new functionality and preventive maintenance. Maintainability characteristic will measure how effectively the maintenance activities can be carried out on business components.

Customizability: Customization is to modify the components corresponding with consumer's objective. Capturing the variability causes customization. A component provides not only the common features for family members but also the variability for each family member. Therefore, to use the component in each family member, customization should be needed. Selection, plug-in, external technique is used to support the modification as customization.

Fig. 2 shows the difference between maintainability and customizability. Maintainability is to maintain without functional change of a component. However, customizability is to modify a new another component using customization techniques.

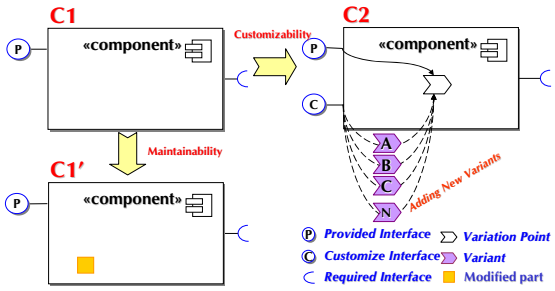


Fig. 2. Difference between Maintainability and Customizability

Conformance: Components are produced on a component platform which in turn implements a component reference model. Hence, a component is dependent on a component reference model and partially on the component platform. The Conformance characteristic will measure how well the components conform to the selected reference model. Components that do not conform to a reference model will not easily be interoperable with others and so they are not highly reusable.

A subcharacteristic is a lower-level quality attribute of a characteristic. To derive subcharacteristics of business components, we first adopt and customize subcharacteristics of ISO/IEC 9126 and then define new ones to fully measure the features of a business component. Subcharacteristics for functionality, maintainability, and conformance are partially derived from ISO/IEC 9126 and other subcharacteristics are newly defined. Subcharacteristics for reusability and customizability are newly defined. Fig. 3 shows the hierarchy of business component which are composed of characteristics and subcharacteristic for each characteristic.

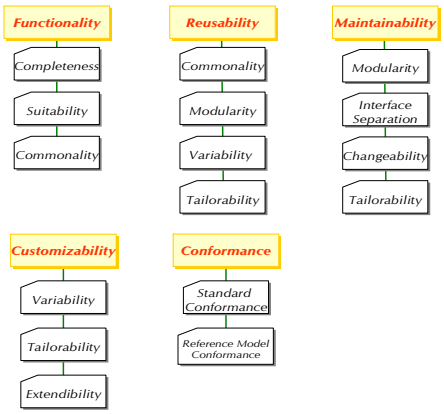


Fig. 3. Subcharacteristics of Business Component

5. Metrics of Business Component

In this section, we provide several representative metrics for subcharacteristics derived in section 4. We don't mention all metrics for subcharacteristics because of on account of space consideration. Therefore, we provide metrics for evaluating representative subcharacteristics per each characteristic. In this section, we define metrics for each subcharacteristic in terms of metric description, formula, value range and relevant interpretations. Fig. 4 shows the corresponding metrics for quality attributes.

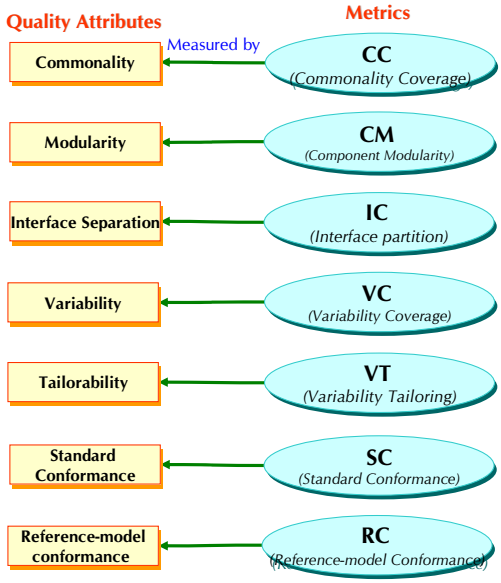


Fig. 4. Mapping between Subcharacteristics and Metrics

5.1. Metric for Commonality

When measuring the commonality of components, both functional aspect and non-functional aspect must be considered. We defined ***Functional Coverage (FC)*** metric to measure the functional commonality and ***Non-functional Coverage (NC)*** metric to measure the non-functional commonality.

FC measures the degree of each functional feature in the component. The commonality of each functional feature can be measured by calculating the degree of family members sharing the functional feature. Therefore, *FC* is defined as;

$$FC = \left(\sum_{i=1}^{n} \frac{\text{number of family members using } i^{th} \text{ functional feature}}{\text{total number of family members in FRS}} \right) \Big/ \text{n}$$

where n is the total number of functional features in the component.

Hence, the range of ***FC*** is 0..1. A lower value of ***FC*** indicates a lower functional commonality, and so the value 1 is the best since the functionality of such a component can be applied to all family members in the domain. The higher the metric value, the wider range of applicability the component has.

NC measures the degree of each non-functional feature in the component. This is similar to ***FC*** metric and can be computed as;

$$NC = \left(\sum_{i=1}^{m} \frac{\text{number of family members using } i^{th} \text{ non-functional feature}}{\text{total number of family members in domain}} \right) \Big/ \text{m}$$

where m is the number of non-functional features in the component.

Hence, the range of ***NC*** is 0..1. A higher value of ***NC*** indicates a higher non-functional commonality, and so value 1 of ***NC*** implies that all non-functional features.

Finally, the value of ***Commonality Coverage (CC)*** is derived from these two metrics;

$$CC = W_{FC} \cdot FC + W_{NC} \cdot NC$$

where W_{FC} and W_{NC} are the weights for each metric which the sum is 1. The value of each weight is set by considering the ratio according to priority in domain.

Hence, the range of ***CC*** is 0..1. The value 1 of ***CC*** indicates that all functional and non-functional features that are satisfied by the requirement are common among family members.

5.2. Metric for Modularity

The Modularity is measured by ***Component Modularity (CM)***. The ***CM*** measures the degree of independence of the component having related elements, which is the degree of functional cohesion.

CM = *(the set of functionality provided solely by the component itself without invoking functions of other components) /*
(the total set of the functionality provided by the component regardless of the need to invoke other components)

where a component includes a logical set of functionality in terms of use cases and the functionality is physically structured as a set of classes. Therefore, cohesion is considered to function to function, function to entity, and entity to entity.

The range of ***CM*** is 0..1, and A higher value of ***CM*** indicates that a component composes of independently related elements are provided by set of functionality. The value 1 is the best since such a component is self-

contained. However, components with the value 1 will be atypical since most components will have some degree of interaction with other components in forms of invocation and delegation.

5.3. Metric for Interface Separation

The *Interface Separation* is measured by ***Interface Partition (IP)***. The ***IP*** measures the degree of representing inner logics or workflows of a component through interface. This can be computed as;

IP = *(number of modified logic or workflow without effecting interface) / (number of modifiable logic or workflow function in inner component)*

Where modification for maintenance takes place in inner code of component.

Hence, the range of ***IP*** is 0..1, and a higher value for ***IP*** indicates that modifications of component inside may not effect the interface.

5.4. Metric for Variability

The variability is measured by ***Variability Coverage (VC).*** The ***VC*** measures how much of the variability identified in FRSs. This can be computed as;

VC = *(number of variation points included in the FRSs) / (number of explicitly or implicitly embedded variation points)*

where some identified variation points may considerate the perspective of the economic value.

Hence, the range of ***VC*** is 0..1 and a higher value for ***VC*** indicates that a component is comprehensive enough to include a larger number of variation points.

5.5. Metric for Tailorability

The tailorability can be measured by assessing how many variation points and elements of interface can be effectively tailored. Valid here means that variation point and interface can be bound and assemble without unexpected side effects or faults due to the effective design.

The tailorability is measured by ***Variability Tailoring (VT).*** The ***VT*** measures how many variation points can be efficiently and correctly resolved.

VT = *(number of effectively resolvable variation points) / (total number of variation points)*

where variation points are effectively resolvable when mechanism to solve each variation point is appropriately and efficiently defined considering the constraint of the variation point.

However, it is difficult to quantitatively measure effectively resolvable variation points, so we recommend using checklists for deciding if each variation point is effective to resolve or not. Checklists should be defined considering types and scopes of each variation point such as binary, optional, alternative, and open variability. For instance, quality assessor should check if the plug-in mechanism defined for the open variability is effective to pass objects.

Hence, the range of ***VT*** is 0..1 and a higher value for ***VT*** indicates that a component is effectively resolved to variation points.

5.6. Metric for Standard Conformance

This standard conformance is measured by ***Standard Conformance (SC)*** metric. The ***SC*** measures the degree of conformance to relevant standards. This can be computed as;

SC = *(number of standard elements strictly followed in the implementation of a component) / (total number of standard elements specified in the relevant set of standards)*

where a standard element can be governmental regulation, de factor standard, standard policies in the domain, and others.

Hence, the range of ***SC*** is 0..1, and a higher value for ***SC*** indicates that a component conforms to many relevant standard elements.

5.7. Metric for Reference Model Conformance

This subcharacteristic is measured by ***Reference-model Conformance (RC)***. The ***RC*** measures the degree of conformance to the adopted component reference model. This can be computed as;

RC = *(number of syntactic and semantic elements strictly obeyed in the implementation of a component) / (number of syntactic and semantic elements specified in the component reference model)*

where an element can be names of interface and classes, rules, constraints, policies or requirements.

The range of ***RC*** is 0..1, and a higher value for **RC** indicates that a component conforms to many syntactic and semantic elements of the component reference model.

6. Application of Business Components in Practice

Using the proposed quality model, the overall quality of the business component can be computed as follows.

Let C_i be the i^{th} characteristic defined in quality model. Therefore the range of i will be 1 <= i <= 5. Let S_{ij} be the j^{th} subcharacteristic of C_i. For example, S_{32} is *interface separation* subcharacteristic for *maintainability* characteristic according to the Fig. 3.

Depending on the purpose of business components, different weights are assigned to the subcharacteristics to compute the value of each characteristic. Let W_{ij} be the weight on S_{ij}, and the sum of all the weights of subcharacteristic for each characteristic should be 1. If the number of subcharacteristic for characteristic C_i is n, then the quality value for characteristic C_i, QC_i, is defined as:

$$QC_i = \sum_{j=1}^{n} W_{ij} \cdot S_{ij}$$

And the weights related to the characteristics can be different according the applied context. Let W_i be the weight related to C_i, so the sum of the weights for 5 characteristics may be 1. Then, the final quality of a business component, Q, can be computed as follows, and the range of Q can be $0 <= Q$, where the higher value implies better quality.

$$Q = \sum_{i=1}^{5} W_i \cdot QC_i$$

In order to apply the proposed quality model for business components effectively, a quality evaluation process, instruction and a set of templates should also be defined.

7. Concluding Remark

Business components should be developed with common features among them. Also, variability among members should be well modeled into components for customizability. Evaluating the quality of such business components is an important prerequisite to successful component-based development (CBD) projects.

In this paper, we proposed a quality model that can be used practically to evaluate business components. We first identified characteristics of business components, and then derived a quality model that consists of 5 characteristics, 16 subcharacteristics, and associated metrics. By using the quality model, we believe that candidate business components can be evaluated quantitatively before they are purchased or acquired.

References

[1] Clemens Szyperski, Component Software - Beyond Object Oriented Programming, Addison Wesley, 1997

[2] S. Kim and J. Park, "C-QM: A Practical Quality Model for Evaluating COTS Components", *Proceedings of the 21st IASTED International Conference*, pp.991-996, February, 2003.

[3] Software Engineering—*Product Quality—Part 1: Quality Model*. ISO/IEC 9126-1, June, 2001.

[4] J. Martin-Albo, M. F. Bertoa, C. Calero, A. Vallecillo, A. Cechich, M. Piattini., "CQM: A Software Component Metric Classification Model", Proceeding of the 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2003), June, 2003.

[5] M. Goulão, F.B. Abreu, "Towards a Component Quality Model", Work in Progress Session of the 28th IEEE Euromicro Conference, 2002.

[6] Crnkovic, I. and Larsson, M., *Building Reliable Component-Based Software Systems*, Artech House, Inc., 2002.

[7] Heineman, G. T. and Councill, W. T., *Component–Based Software Engineering*, Addison-Wesley, 2001.

[8] D'Souza, D. F. and Wills, A. C., *Objects, Components, and Frameworks with UML*, Addison Wesley Longman, Inc., 1999.

[9] Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B., Wust, J., and Zettel, J., *Component-based Product Line Engineering with UML*, Pearson Education Ltd, 2002.

[10] S. Kim, J. Her, and S. Chang, "A Theoretical Foundation of Variability in Component-Based Development," *Information and Software Technology*, Vol.47, pp.663-673, 2005.

THE EVOLUTIONARY ROLE OF VARIABLE ASSIGNMENT AND ITS IMPACT ON PROGRAM VERIFICATION

Daniel E. Cooke, J. Nelson Rushton, and Robert Watson
Computer Science Department
Texas Tech University

1. Introduction

This paper addresses the evolution of variable assignment in Computer Languages, particularly as it pertains to program verification. Although we claim that these remarks are relevant to any form of verification, we will focus on proofs of properties of programs, including correctness. Sections 2 and 3 discuss verification from a formal standpoint, comparing the relative difficulties of proving correctness for programs with and without assignment. Section 5 describes a case study in which the same large "real world" program was written in both ways, and compares the results with respect to informal program verification – that is, the process of becoming reasonably convinced that the program works correctly.

2. Assignment and proofs for procedural languages.

The first goal we seek is to demonstrate how assignment affects program proofs. Consider a simple procedural program to compute *n!*. A formal definition of *n!* is:

0! = 1
*n! = n * (n-1)! for n>0*

The following outline of a proof uses Hoare's Triples, which can be found at [7]. We focus here on the two major semantics required for the proof. The rest of the steps are simple mathematical inferences or involve straightforward compositions. Recall a *triple* is of the general form:

{P} S {Q}

where *P* and *Q* are logical equations representing the pre- and post-conditions of a statement or set of statements written in a procedural language. The axiom of assignment states:

{P} x := E {Q}

That is *P* is obtained from *Q* by substituting all occurrences of *x* in *Q* by *E*. Formally, assignment is:

(assign) $\dfrac{}{\{P[a/x]\}\ x := a\ \{P\}}$

where the **P[a/x]** denotes the term obtained from P by replacing all occurrences of *x* by *a*. Many people (including us) find this inference rule counterintuitive. Yet, it provides the simplest and most commonly used axiomatic semantic for assignment. This is the first clue that assignment is not mathematically straightforward, and does much to confound any effort to determine properties of a program. In the proof to follow we will also employ the axiom defining the *while* loop:

(while) $\dfrac{P \wedge b\ \{S\}\ P}{\{P\}\ \text{while b do S}\ \{P \wedge \text{neg(b)}\}}$

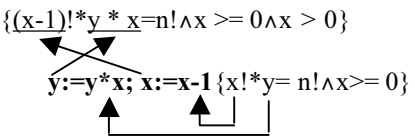
This rule underscores the importance of the loop invariant shown above as *P*. Let's focus now on the *n!* example. Consider the triple containing the program:

{x = n ∧ n > 0 }
y:=1; while (x > 0) do (y:=y*x; x:=x-1)
{y=n! }

Since *x=n* and given the first assignment *y:=1* the axiom of assignment and a few mathematical inferences permit:

{ x = n ∧ x > 0 } **y:=1** {x! * y = n! ∧ x >= 0 }

The formula *x! * y = n! ∧ x >= 0* is the loop invariant. We can now tackle the loop's body. First note that with simple mathematical inferences, the invariant implies *x* (x-1)! * y = n! ∧ x >= 0 ∧ x > 0.* We can now rearrange the first conjunct in the loop invariant due to commutative properties and employ the axiom of assignment twice to obtain:



This step contains the intuitive crux of the argument – that the correctness of the program for x implies correctness for x+1. However, this point is rather badly obscured in the proof, by the necessary treatment of assignments to x and y.

Notice, that because *x* is decremented, we now take the more general *x>=0* and drop *x>0*. Thus, *x>0* takes the role of *b* in the while loop and we obtain:

{x! * y = n! ∧ x >= 0 ∧ x > 0}
y:= 1; while (x > 0) do (y:=y*x; x:=x-1)
{x! * y = n! ∧ x >= 0 ∧ neg(x > 0)}

Which, implies *x=0,* which gives *0! * y = n!* or *y=n!*
Q.E.D.

The detailed proof can be found at http://www.cs.cornell.edu/courses/cs411/2004fa/lectures/lec11.txt and is summarized below:

{x = n ∧ n > 0}
(x = n ∧ n > 0) *implies* (x! * 1 = n! ∧ x >= 0)
{x! * 1 = n! ∧ x >= 0}
y := 1
{x! * y = n! ∧ x >= 0}
while (x > 0) do
{x! * y = n! ∧ x >= 0 ∧ x > 0}
(x! * y = n! ∧ x >= 0 ∧ x > 0) *implies* {(x-1)! * y * x = n! ∧ (x-1) >= 0}
{(x-1)! * y * x = n! ∧ (x-1) >= 0}
y := y * x;
{(x-1)! * y = n! ∧ (x-1) >= 0}
x := x - 1
{x! * y = n! ∧ x >= 0}
{x! * y = n! ∧ x >= 0 ∧ neg(x > 0)}
(x! * y = n! ∧ x >= 0 ∧ neg(x > 0)) *implies* (y = n!)
{y = n!}

3. Functional programs and simpler proofs.

With functional and logic programming languages that do not support assignment, proving the correctness of a program is typically much easier. Assignment does not confound and obscure the points one is trying to make in the proof. Consider the language SequenceL [2-6], which does not have *reads, writes,* or *assignments.* A recursive factorial definition appears as:

```
fact: s → s
fact(n)::= fact(n-1)*n
       when n>0 else 1
```

The SequenceL function definition is identical to the formal definition assuming inputs, *n≥0:*

0! = 1
*n! = n * (n-1)! for n>0*

which calls into question whether a proof is needed at all. But what the heck? Here we go with an inductive proof of correctness.

The proof requires two axioms from the denotational semantics of SequenceL:

(A1) If *f* is a symbol for a user defined n-ary function in a SequenceL program *P*, and $t_1,...,t_n$ are terms, then the value of the term $f(t_1,...,t_n)$ with respect to program *P* is the value of the term obtained by substituting $t_1,...,t_n$ for their respective formal arguments in the function body of *F*.

(A2) If *x*, y, and z are SequenceL terms, then the value of the term "*x* when *y* else *z*" is the value of *x* if the value of *y* is *true,* and the value of *z* if the value of *y* is *false.*

We now proceed with the proof. For a SequenceL term T we will write $v(T)$ to denote the semantic value of *T* obtained from axioms A1 and A2. By definition of factorial, it suffices to show that $v(\text{fact}(0)) = 1$, and that for every $n>0$ $v(\text{fact}(n)) = n*v(\text{fact}(n-1))$.

- From (A1) it follows $v(\text{fact}(0)) = v(\text{fact}(0\text{-}1) *0$ when 0>0 else 1).
- By A2, this is $v(1)$, which equals 1.
- Using (A1) and (A2) again similarly, for all n>0 we have $v(\text{fact}(n)) = v(\text{fact}(n\text{-}1) *n$ when n>0 else 1) $= v((\text{fact}(n\text{-}1) *n) = v((\text{fact}(n\text{-}1))*v(n) = v((\text{fact}(n\text{-}1))*n$.

This completes the proof.

We offer a few observations about this proof. First, we have left out some trivial steps involving the semantics of the built-in arithmetic operations of SequenceL (for example, we did not explicitly use the fact that the value of the SequenceL expression <*u*v*> is the product of the value of *u* with the value of *v*). However, our induction proof was, on the whole, at least as detailed and pedantic as the Hoare's triple proof given previously, and yet still an order of magnitude shorter and simpler.

It might be objected that we "cheated" by choosing our functional program to match the definition of factorial we were working with. Our response is twofold. First mathematical specifications are typically functional, so it is essentially always possible to do this when programming functional. It *should* be done at every opportunity, but procedural syntax with assignment does not allow one the luxury of cheating in this manner. Second, given a different mathematical specification of factorial, it would be a simple matter to prove by induction that it is equivalent to the one we used, which when combined with our proof would serve as verification of our code against the alternative specification. Thus, the verification of the functional code remains an order of magnitude less complex than the verification of the procedural code given *any* reasonable specification of factorial.

Whereas the above relates the SequenceL function with the mathematical definition of *n!* through the use of

an inductive proof, the Hoare's Triples proof did not clearly relate the program to the mathematical definition of *n!* and also obscured the inductive proof.

Since the proofs in language without assignment are so easy and, as a consequence, more convincing, one can imagine the possibility of proving other properties of a program. For example, one might desire a proof that showed that when *n* is nonnegative, the *v(*fact(n)) always gives a positive result.

For the inductive basis of the proof, we have:

- *v(fact(0)) = 1,* which is greater than zero.
- Next, we have the inductive step, where we must show that for all $n \geq 0$ *v(fact(n)) > 0 implies v(fact(n+1)) > 0*. The function definition gives us, for n>0 , *fact(n+1) = fact(n) * (n+1).* Since *fact(n) > 0* by inductive hypotheses, and *n* is positive, and since addition and multiplication are closed under the positive integers,
- We can conclude:*f(n+1) > 0*

This completes the proof.

4. Assignment and Input are equivalent when it comes to State.

As in most scientific endeavors intuition and precision play an important role in computer science. Precision in our understanding of computational phenomena arises in our study of equations that characterize the phenomena. The study of denotational semantic equations for assignment and input reveals that both have the same destructive qualities on the state of a variable. Destructive here is meant to indicate that one can "destroy" the previous value of a variable with a read or assignment – both of which replace the contents of a variable.

Given the following denotational definitions, one can clearly see that both instructions make program verification difficult to accomplish as seen in the Hoare's proof of *n!:*

Syntax:
S ::= Id:= E | read(Id) | …
E::= (E) | Id | Numb | E1 + E2 | E1 * E2 | E1 – E2 | E1/E2
:
Syntactic Domains:
S: Stmt E: Exp …
Semantic Domains:
σ : St = Id → I γ : CF = St × Fi × FI
ν : I = {...,-2,-1,0,1,2,....}° ϕ : Fi = I*
Semantic Function Domain:
Σ : Stmt → CF → CF ε : Exp → St → I
Semantic Equations:
:
h. Σ « Id **:= E**»γ **= <** σ**[**ε «E »γ↓1 / Id**],** γ↓2 **,** γ↓3 **>**
i. Σ « read(Id) »γ **= <** σ**[hd(**γ↓2**)** / Id**], tl(** γ↓2 **),** γ↓3 **>**
:

Respectively, in the equations *h* and *i,* the state of the machine is modified to replace an *Id* by the meaning of an expression *E,* in the case of *assignment*, and by the head of an input file, in the case of the *read.* Many languages that otherwise have no assignment, make some compromise for input and output, which is likely to affect program verification.

To avoid problems with verification that this compromise entails, SequenceL does not currently allow for assignment or input-output. However, we are experimenting with two approaches for I-O. The first is already implemented. We have a very small language called CSML that allows for conditional I-O based on Abstract State Machines. [1] This language performs all the I-O functions for SequenceL through calls and returns to the SequenceL interpreter. Proofs of correctness in the computational parts of the program remain pristine as seen in the proofs of *n!* in section 3. Proofs of correctness are easier in the CSML, because it is a small language, with a simple semantic, which views all variables as vectors indexed by some time stamp. Therefore, assignments like:

s := s +x;
:
v := s +y;

are not data-dependent on one another. In other words in a traditional language, assuming some initial value of *x* and *s* together with no intervening assignments to variables *x, s,* and *y* the definition of *v* would be *(s+x) + y.* In CSML the *s* on the right-hand sides of the assignment refer to the current time step's value of *s* and the *s* (and *v*) on the left-hand side is the next time step's value. I.E., assuming *t* is the current time step, the arrangement above is viewed as:

$s_{t+1} := s_t + x_t$;
:
$v_{t+1} := s_t + y_t$

Data flow dependencies that confound verification are thus avoided making verification easier. Furthermore, program understanding is simplified coming closer to declarative languages where assignments are not temporal, but are viewed as definitions – they are axiomatic as far as the program is concerned.

Another approach we are developing involves viewing I-O as a function. In this approach, obtaining inputs will not set global variables but will simply return the current head of an input list. In other words, rather than modifying a state like equation *j's* σ**[hd(**γ↓2**)** / Id**],** the function is referenced and its body returns *hd(input_list).*

5. Experimental Work.

In this section we describe our experience with SAFM, the Shuttle Abort Flight Management software developed by the Guidance Navigation and Control (GNC) group at Johnson Space Center. [4] This system was developed using a procedural style specification document. We became involved with this system using it as an example to illustrate the differences between procedural program specifications and functional specifications and to demonstrate the advantages of the functional specification.

5.1 Description of SAFM

During launch and reentry, the SAFM system monitors the vehicle state and maintains a prioritized list of feasible landing sites that are available to the crew in the event that the flight must be aborted. The preferred landing sites are displayed to the crew. If it is necessary to abort, the crew can select one of the listed sites and the guidance system will direct the shuttle to the new destination.

SAFM must prioritize a large number of potential landing sites and many sites will be eliminated because the vehicle cannot reach the site in its current state. Of the remaining feasible sites, the best (3 to 10) must be selected for presentation to the crew. The criteria for prioritizing sites include the margin of feasibility, geographic preference (e.g. North American sites have a high preference), and facilities available at the landing sites. Here we are concerned with the design and verification of the system, so the criteria for ranking sites are not under consideration, only that the system correctly computes the specified ranking.

5.2 Specification Example

The SAFM Requirements Specification (SRS) consists of variable definitions, assignments to variables, and conditional expressions that control the assignments. These variables indicate global vehicle state and are used in the specification of all input/output between SAFM and other modules (flight software). They are also used to specify parameters passed within the SAFM description. Some of the variables have initial values; others do not. The use of these variables permits other modules outside of the scope of the SAFM specification to access variables whether or not a value has been defined for the variable. The specification is around 300 pages in length.

5.3 Previous Experience With SAFM

NASA found the procedural SAFM specification difficult to verify. Our claim is that the difficulty can be traced to the use of assignment. Each assignment in the specification results in a state transition. There are approximately 200 variables and 500 assignments that may be executed in each SAFM cycle. The fact that a small subset of these assignments is executed in each cycle does not greatly simplify the analysis of the system because the expression that controls the assignment must be evaluated for each assignment to determine the effect of the assignment on the system state.

Each execution (run) of the SAFM system can consist of 1000 or more SAFM cycles. With a potential for 500 state transitions in each cycle, there can be as many of 500,000 state transitions in a run. This begins to illuminate the size of the analysis problem.

Another view of the complexity of analyzing the system can be found by examining the process of determining the value of a variable at some point in the system's execution. This process consists of selecting the variable whose value is of interest (the target variable), locating all variables on which the target variable depends, and computing the value of the target variable from the dependent variables. Obviously, this process proceeds recursively. We can graphically depict the process as a tree structure where the root of the tree is the target variable, the children of the target variable are the variables on which it immediately depends, and additional variables descend from each child node as needed. Leaf nodes are variables whose values do not need to be computed such as input variables and constants.

Figure 1 shows one such tree expanded for the SAFM variable 1_Out_TAL_Auto_Availability Sel. From this figure, the size of the problem should be evident even though the size of the tree makes it difficult to read. The tree in this figure has been simplified in several ways to reduce its size. First, each variable is only expanded once. Other nodes in the tree that refer to a variable that is expanded elsewhere in the tree are represented by elliptical leaf nodes, indicating that additional expansion is not required. Input and constant variables are also shown as elliptical leaf nodes. Second, many variables have common subsets of dependent variables. These common subsets are made children of a single circular node that serves as a child of each parent of the common subset. Strictly speaking, the graph is no longer a tree, but we overlook this detail as a matter of convenience in its presentation. The third simplification is that variables are not expanded beyond the current SAFM cycle. That is, if a node in the tree represents the value of a variable taken from the prior SAFM cycle, that node is represented by a double rectangle.

These simplifications enable the tree of Figure 1 to be displayed. However, it should not be forgotten that this tree is not fully expanded. If we use an estimated average branching factor of 4 and depth of 3, the tree will require the evaluation of 81 variable values within the cycle.

5.4 Issues Found

In the process of developing a functional specification for SAFM, we uncovered one significant issue and numerous smaller issues, many of which may

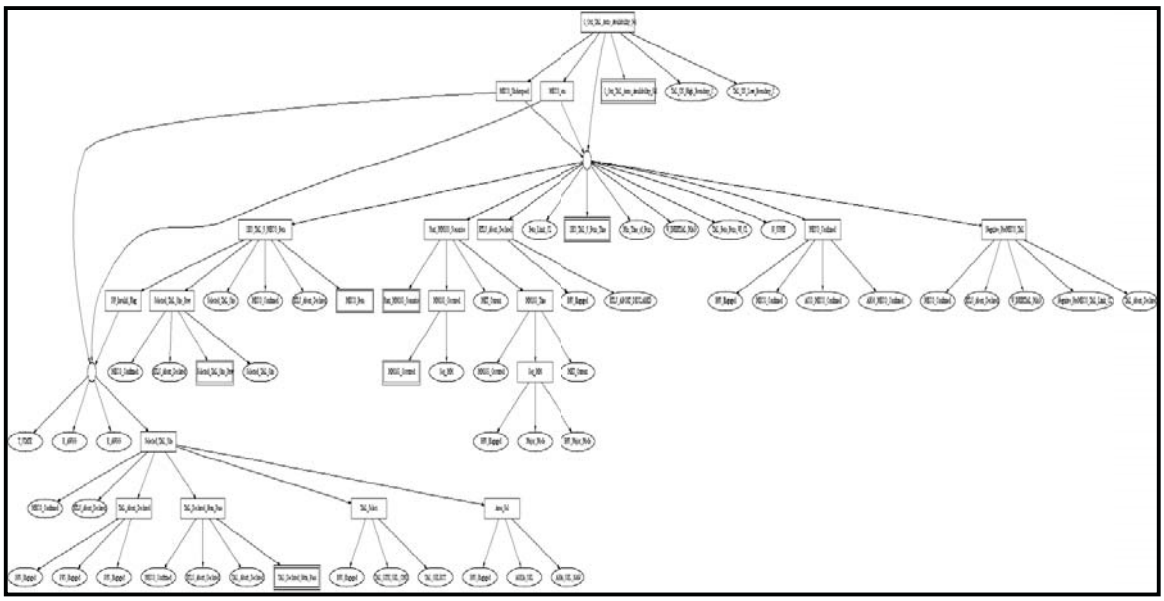


Figure 1: SAFM Variable Dependency Tree

make future changes to the specification difficult. Most of these issues involve un-initialized variables.

We present these issues here because they are significant to our purpose. They are significant because their existence was detected as a direct result of developing the SAFM specification in the functional paradigm. The absence of variables and assignment in a functional specification make incomplete specification of a value easy to identify when reading the specification document.

The first and most interesting issue involved a variable named Negative_Return. No initial value is specified for this variable and it is clear that the initial assignment to the variable cannot occur prior to the evaluation of an expression in which the variable appears. The specification where this first appeared is shown in Figure 2. It is easy to see in the figure that the assignment to Negative_Return (the only assignment to this variable in the specification) cannot occur unless Negative_Return is initially false. Since no initial value is specified for Negative_Return, the value of Negative_Return remains undefined throughout the specification.

The next class of issues, which also involve initialization, probably never manifests faults in the specification. However, they were numerousand it is likely that they could be the source of manifest faults during specification maintenance if they were allowed to persist. These issues involve variables that have no initial values, but have values assigned prior to their first use in the evaluation of an expression.

5.5 Origin of the Issues

We claim that assignment is at the root of the problem. As discussed previously, each assignment within a procedural construct represents a state change. Analysis of the specification or program is complicated by the frequency with which state changes occur. Consider a procedural function call that, as a side effect, makes an assignment to a global variable. During analysis, at the point immediately after the appearance of the function call, all facts previously known about the state of the program become suspect because the program has made a state change. This is the justification for eliminating global variables, but variables do not have to be global, they could be object members. And if objects are not used in the design, the problem continues to persist within functions when the values of local variables change. The change can be due to an explicit assignment or an implicit assignment to a loop control variable. Many rigid assumptions about variables cannot be made because the value of the variable changes during the analysis. Without assignment, variables are simply constants, which are indistinguishable from nullary functions.

Commonly an analysis of a procedural design produces an abstract representation of the design that faithfully represents the behavior of the design, but also yields to the mathematical tools available for performing the analysis. Unfortunately, this abstract representation is often very different from the original design so that creating the abstraction is not easy. It seems intuitively obvious that we would like to start with a design that parallels the abstraction. This would make analysis easier provided that we can effectively produce an implementation from this design. Elimination of assignments from the design is a step in that direction.

```
3.7.1.1.3.10 If MECO Confirmed = 0 (false), Sequencer shall set Negative Return via the following logic:
    IF BFS Engaged = 0 (false) AND 1 Out MET < 1.0 AND N SSME = 2 AND FIRST EO VI < RTLS VI Boundary CL THEN
        1 Out MET = MET Current
    END
    IF Negative Return = 0 (false) THEN
        IF RTLS Abort Declared = 0 (not declared) THEN
            IF (V INERTIAL MAG > RTLS VI Boundary CL AND (1 Out MET + Crew Delay CL) < MET Current) OR TAL Abort
                Declared = 1 (declared) OR S ATO = 1 (declared) THEN
                Negative Return = 1 (true)
            END
        ELSE
            IF BFS Engaged = 0 (false) AND CONT 2EO START = 1 (true)
                THEN Negative Return = 1 (true) END
        END
    END
```

Figure 2: Negative Return

6. Conclusion.

This paper is meant to provide some insights into program verification we have gained as a result of our experiences in language design. These insights have led to decisions in the design of SequenceL. The resulting features of SequenceL together with its ability to correctly derive iterative program structures has led to our current partnership with aerospace engineers at Johnson Space Center, who are currently employing SequenceL as an executable specification language to develop the requirements of the Abort Executive for the Crew Exploration Vehicle.

Acknowledgement

The research leading to the results in this paper was funded in part by NASA-NNG05GP48G.

References

[1] Andreas Blass, Yuri Gurevich: The Linear Time Hierarchy Theorems for Abstract State Machines and RAMs. J. UCS 3(4): 247-278 (1997).

[2] Cooke, D.E., "An Introduction to SEQUENCEL: A Language to Experiment with Nonscalar Constructs," *Software Practice and Experience,* Vol. 26(11), (November, 1996) 1205-1246.

[3] Cooke, D.E. and J.N. Rushton, "SequenceL—An Overview of a Simple Language," *Proc. Int'l Conf. Programming Languages and Compilers* (PLC 05), 2005**,** pp. 64-70.

[4] Cooke, D. et al., "Application of Model-Based Technology Systems for Autonomous Systems," *Proc. Infotech@Aerospace*, AIAA-2005-7063, Infotech@Aerospace, 2005.

[5] Cooke, D., M. Barry, M. Lowry and C. Green, "NASA's Exploration Agenda and Capability Engineering," IEEE Computer, January 2006, pp. 59-69.

[6] Cooke, D.E. and J.N. Rushton, "Normalize, Transpose, and Distribute: A Basis for the Decomposition and Parallel Evaluation of Nonscalars," in revision *ACM Trans. Programming Languages and Systems*, 2006.

[7] Hoare, C.A.R. "An Axiomatic Basis for Computer Programming," *Communications of the ACM,* 12:10, 576-580.

Pattern-Based System Evolution: A Case-Study

Neelam Soundarajan
Computer Science & Eng.
Ohio State University
neelam@cse.ohio-state.edu

Jason O. Hallstrom
Computer Science
Clemson University
jasonoh@cs.clemson.edu

Abstract

Design patterns help system designers apply proven solutions to commonly occurring problems in particular contexts. While their impact on software practice has been signi cant, the application of patterns has been limited to the early stages of the software lifecycle. In this paper, we show how the bene ts of a pattern-centric approach can be leveraged across the lifecycle, speci cally as the system evolves to meet changing requirements. The approach is based on a notation we have previously proposed for providing precise speci cations of patterns and of how patterns are specialized for use in particular systems. In this paper, we summarize the main elements of the speci cation approach, show how such speci cations allow us to capture essential pattern-centric information about a system, and demonstrate, via a simple but illustrative case-study, how this information allows us to preserve the *design integrity* of the system as it evolves.

1 Introduction

Design patterns [3, 1, 11] have had a signi cant impact on software practice, especially in the engineering of large systems. But the application of patterns has been limited to the early stages, mainly the initial design phase, of the software lifecycle. The thesis underlying our work is that the bene ts of patterns can be maximized by ensuring a pattern-centric approach across the *entire software lifecycle*. Indeed, unless we do so, the *design integrity* of a system is likely to be compromised as the system evolves over time. In this paper, we present such an approach and demonstrate its effectiveness via an illustrative, if simple, case-study.

Design patterns are commonly described in an informal (although stylized) manner in various pattern catalogs [3, 1, 11, 10]. While such descriptions are of great value during initial design, they are inadequate for use *across* different stages of the lifecycle. The problem is that while the design team may have intended a given pattern to be applied in a particular manner in a system, the implementation team may have a slightly different understanding of how the pattern was to be applied; and the maintenance team's understanding may be yet again different. And as the system evolves over time to meet changing requirements, and as new team members replace older members, the problem can become acute, resulting, as we noted above, in a breakdown in the design integrity of the system. The problem is especially serious for large systems that, ironically, bene t the most from the use of design patterns. This is because these systems are likely to have larger design, implementation, and maintenance teams so that differences in understanding of how a given pattern was intended to be applied are likely to be present not just across teams, but within each team. As we will see, our approach addresses this problem by being based on precise speci cations of patterns and of how patterns are specialized for use in particular systems.

Consider an example that will serve as our case-study. An application is being designed that includes an Elevator class whose instances represent an elevator in a high-rise building. The state of an elevator object[1] includes _np, the number of people currently in it, and _nf, the number of the oor it is currently at. The application also includes a Display class whose instances display the current oor number of the elevator. Any number of displays (typically one per oor and one inside the elevator) may be attached to, i.e., associated with, an elevator. There may even be multiple types of displays, for example, an EnhancedDisplay that not only displays the oor number of the elevator but also raises an alarm if the _nf of an elevator takes on certain values during certain hours of the day, etc.

Most designers would recognize this as a potential application of the *Observer* pattern [3] with the elevator playing the Subject role, and the displays playing the Observer role. Thus a reasonable design of Elevator would include a notifyDisplays() method –which would play the role of the notify() method of the *Observer* pattern– that would be invoked whenever the state of the elevator undergoes a signi cant change, i.e., it goes to a new oor , so the value of _nf changes. notifyDisplays() would invoke an appropriate method of each associated display. And these methods, in turn, would appropriately update the displays, possibly performing operations that are *customized* to the individual displays. Thus in the case of EnhancedDisplay, this method would raise an alarm if appropriate, whereas the update() of the regular Display class would simply get information about the new value of _nf from the elevator and display it.

Suppose now that during system evolution, perhaps following incidents of dangerously overcrowded elevators, a

[1] We use names starting with uppercase letters such as Elevator for classes; and corresponding lowercase names such as elevator for objects that are instances of these classes. We use a similar notation for *roles* in patterns and objects that play these roles in individual pattern instances.

SafetyDisplay class is created. Instances of this class will display both the oor number of, and number of people in, the elevator. So its update() method, when invoked by Elevator.notifyDisplays(), will obtain, using appropriate *getter* methods of the Elevator class, information about both the current oor number as well as the current number of people in the elevator – unlike the update()s of the original Display classes which obtained only the oor number.

When this class is implemented and tested, everything seems to work well. But reports start coming in that safetyDisplays are *occasionally* displaying incorrect information about the number of people in the elevator. After careful analysis, the maintenance team traces the problem to the fact that notifyDisplays() is invoked only when the value of _nf, the oor number, changes. This was ne so long as displays only cared about _nf. But safetyDisplay objects are also concerned with the value of _np. And this part of the elevator state can change even if _nf has not changed; for example, following an openDoor() operation which does not change the oor number, people may enter or leave the elevator. This was overlooked during the initial testing of SafetyDisplay because, by coincidence, during testing, whenever the value of _np had changed, so had the value of _nf; hence notifyDisplays() had been invoked at each of these points, and the information displayed by the safetyDisplay devices had been accurate. The problem in this system is related to the pattern being used. More precisely, the bug is related to the fact that the requirement of the *Observer* pattern that at certain speci c points the notify() method of the subject should be invoked, has not been met.

The approach we describe in this paper is intended to help avoid just this type of problem; or, if not avoid, track it down and resolve it more easily and at an earlier stage. Our approach is based on precise speci cations, in the form of pattern *contracts*, of the design patterns underlying a system; and of the precise manner in which the patterns are specialized for use in the form of *subcontracts*. In writing these speci cations, we use a notation we have previously proposed [12, 5] for this purpose. Our speci cations will allow the software team, during detailed development and implementation, to check that the requirements that the correct usage of the pattern calls for, as speci ed in the contracts/subcontracts, are satis ed. As the system evolves to meet changing requirements, such as the introduction of the SafetyDisplay class, the maintenance team will be able to check that the modi cations in the system are such that the system *continues* to meet the requirements contained in the relevant contracts and subcontracts; or, in some cases, if the evolution is drastic enough that the subcontract(s) are no longer appropriate, to revise them to re ect the changes in how the pattern is being applied. In either case, the underlying design of the system will be precisely documented and will serve to guide its evolution through its entire lifecycle thereby helping preserve its design integrity.

Techniques for formalizing design patterns have been proposed by a number of authors and we will brie y consider some of these proposals later in the paper. Compared to these other approaches, our pattern contracts are particularly well-suited for use in a pattern-centric software lifecycle since they focus on *behavioral* –rather than, structural– issues involved in using the patterns; and the subcontracts allow software designers to specify, in a very natural fashion, exactly how a pattern is specialized in a particular system. Moreover, the contracts and subcontracts can also be checked by appropriate runtime *monitoring tools* to identify violations of the contracts. All of this should make our approach very appealing to practioners. In a companion paper [4], we consider performance issues related to runtime monitoring of such speci cations. In this paper, we will not consider monitoring in any detail.

The rest of the paper is organized as follows. In Section 2, we consider further details of the design of the *Elevator-Display* case-study and its evolution. In Section 3, we summarize our approach to pattern contracts and subcontracts, show how they are used in the case-study, how the problems in the system introduced during evolution manifest as violations of the contract/subcontract requirements, and how a runtime monitoring system can help the system implementers/maintainers pinpoint the problem and correct it. In Section 4, we consider related work. In Section 5, we conclude with a brief summary and pointers to future work.

2 Pattern-Based Design and Evolution

Outlines of the Elevator class and the Display class, written in *Java* syntax, appear in Fig. 1. An elevator object in-

```
1  public class Elevator {
2    private Integer _nf, _np;
3    private Set _displays;
4       // displays attached to this elevator
5    ...constructors, field accessor methods...
6    ...addDisplay(d), removeDisplay(d)...
7    public void move() {
8      _nf=...; openDoor(); notifyDisplays(); }
9    public void openDoor() { _np=...;  }
10   private void notifyDisplays() {
11     ...call update() on displays in _displays...}
12 public class Display {
13   private Elevator _myelevator;
14   private  ...info about the floor number of my elevator...
15   ...constructors...
16   public void update() {
17     ...get info about _nf of _myelevator & update this...
18   ...for EnhancedDisplay, raise alarm as appropriate...}
```

Figure 1: Elevator-Display Code (partial)

cludes a pair of variables _nf and _np representing the num-

ber of the floor it is currently at, and the number of people currently in it. The elevator object also includes a variable, _displays, that holds a set of references to the display objects attached to this elevator. A display object holds a reference to the elevator it is attached to. This will be used by the Display.update() method, when it is invoked, to in turn invoke the appropriate *getter* methods on the appropriate elevator object to obtain current information about this elevator. Details on how the display stores this information is omitted as that will vary from one type of Display to another.

Let us now turn to the requirements of using the *Observer* pattern. This pattern's intent is to keep a group of observer objects *consistent* with the state of a subject. In order to achieve this goal, the class playing the Subject role must, according to the standard description [3] of the pattern, provide a notify() method; the class playing the Observer role must provide an update() method; and "whenever a change occurs [in the subject] that could make any of its observer's states inconsistent with its own," the notify() method must be invoked on the subject. notify() must, in turn, invoke update() on each attached observer. The update() method will "reconcile its [the observer's] state with that of the subject." In the *Elevator-Display* system, the role of notify() is played by Elevator.notifyDisplays() (lines 10, 11 in Fig. 1). Display.update() (line numbers 16, 17, 18) plays the role of the update() method of Observer[2].

Consider now the requirement in the pattern description that notify() be called whenever a change occurs that could make the state of an observer *inconsistent* with the state of the subject. When exactly can this happen? As we saw earlier, the initial designers/implementers assumed this could happen only when the value of elevator._nf changed. Thus, only in Elevator.move() –the only method that changes _nf– is there a call to notifyDisplays(). Elevator.openDoor(), which changes (only) the value of _np, does not include such a call. As long as all designers and implementers share a common understanding of what *inconsistency* means, there will be no (pattern-related) problems.

Let us now turn to the system's evolution. As we saw, changing system requirements called for the system maintainers to introduce SafetyDisplay, the new type of display. The outline for this class appears in Fig. 2. As we saw, when instances of this class are created and attached as observers to elevator objects[3], the system does not always function as

```
1  public class SafetyDisplay {
2    private Elevator _myelevator;
3    private ...info about my elevator including
4                number of people in it...
5    ...constructors...
6    public void update() {
7    ...get info about _nf, _np of _myelevator & update this...}
```

Figure 2: SafetyDisplay Class (partial)

expected. But note that the source of the problem is *not* in this new class but rather in the violation of the shared understanding of what consistency/inconsistency between the states of the observers and subject means, and the resulting failure to meet the *Observer* pattern's requirements concerning *interactions* between these objects. The primary source of the problem is that there is no *precise* specification of what these requirements are, nor a clear documentation of the "shared understanding". Our approach described in the next section is designed to provide these and, thereby, help ensure that the system remains faithful to its original design even as it evolves.

3 Contracts, Subcontracts and Evolution

The idea of precisely specifying a given class using *class invariants* and *pre-* and *post-conditions* for characterizing the behaviors of individual methods of the class is, of course, well known and is the basis of the *design-by-contract (DbC)* [8] approach. But there are two key differences between *DbC* and the problem we are interested in. First, most design patterns are concerned with the *interactions* among two or more objects, whereas, in specifying a class, we are only concerned with that single class and how its methods must behave. Indeed, as we just saw, the problem introduced in the *Elevator-Display* system during its evolution arose because of our failure to meet the *Observer* pattern's requirements with respect to these interactions. Second, there is a real risk in formalizing patterns in the same way as class behaviors in that *flexibility* –the hallmark of almost every pattern, and is what makes patterns applicable to a wide variety of systems– may be compromised. Thus, if in formalizing the *Observer* pattern, we were to adopt one notion of what it means for an observer state to be *consistent* with that of a subject, the pattern may not be applicable to systems that need to use a different notion of consistency. We will see that our approach addresses both of these issues.

3.1 Pattern Contracts

In [12, 5], we describe the full details of pattern contracts. Here, we provide a brief summary and consider some of the key aspects of the contract for the *Observer* pattern. A *pattern contract* includes a *role contract* corresponding to each role of the pattern. In the case of the *Observer* pattern, we will have contracts for the Subject and Observer

[2]The Subject role is also required to provide attach() and detach() methods that are invoked to *attach* a new observer to the given subject and to *detach* a currently attached observer. Elevator.*addDisplay()* and *removeDisplay()* (line 6) play the roles of these methods but, due to space limitations, we will not discuss them further.

[3]We should note that, for simplicity, we have omitted some essential details in the code. In particular, classes such as EnhancedDisplay and SafetyDisplay should be defined as derived classes of a base Display class; or possibly, they should be defined to implement a common Display interface. Only then can instances of the Elevator class treat all attached display objects in a uniform manner as shown in Fig. 1.

roles. A role contract will specify the *role state*, consisting of variables of appropriate types; and specify, using pre- and post-conditions, the behaviors of methods of the role. Thus the Subject contract will include a specification of the notify() method of this role, and the Observer role contract will specify the update() method of this role. One important point is that the post-conditions for these methods may include information about the *trace*, i.e., the sequence, of calls that the method makes during its execution. Thus, the post-condition of the notify() method will state that, during its execution, the method will call the update() method on each object to which there is a reference in the _observers variable (which is part of the state of the Subject role).

The pattern contract also includes a portion corresponding to the pattern as a whole. An important component of this portion is the *pattern invariant* which is a relation involving the states of all the objects playing roles in the pattern. The essential point of abiding by the requirements specified, for example, in the specifications of the methods in the role contracts is that, if all the requirements are indeed met, the invariant is guaranteed to be satisfied.

At any point during the execution of a given system built using a particular pattern, we may have *multiple* groups of objects, with each group of objects interacting among themselves according to the requirements of the pattern. For example, in our case-study, there may be several elevator objects, each playing the Subject role, each with its own set of display objects playing the Observer role. Each such group of objects can be considered an *instance* of the *Observer* pattern. The pattern contract includes an *instantiation clause* that specifies how a *new instance* of the pattern is created. In addition, each role contract specifies how an object *enrolls* to play the particular role. The enrollment clause for the Observer role will state that when an object that can play this role invokes the attach() method on a subject, it enrolls in the corresponding pattern instance. The *disenrollment clause* similarly states the action that must be performed for an object to disenroll from a pattern instance. Due to space limitations, we will not discuss these clauses in detail, referring the interested reader to [5].

The requirements specified in the pattern contract apply to each pattern instance. A runtime *monitoring tool* that monitors for violations of the contract will keep track of the objects enrolled in the different instances of each pattern used in the design of the system; details regarding runtime monitoring are available in a companion paper [4].

How do we preserve *flexibility*? In specifying the invariant as well as in specifying the methods of the individual roles, we use *auxiliary concepts*. An auxiliary concept is a relation involving two or more states of objects enrolled in a pattern instance. The contract for the *Observer* pattern (Fig. 3) uses two such concepts, *Modified*() and *Consistent*(). *Modified*() is a relation involving two states, say, s1 and s2 of the subject; if *Modified*(s1, s2) is true, that indicates that if the subject state were to change from s1 to s2, the modification is significant enough that the notify() method must be invoked (which, according to the specification of that method,) will in turn invoke update() on each attached observer. Similarly, *Consistent*(s, o) is a relation involving a subject state s and an observer state o. The observer role contract will require that when update() finishes execution, the state of the particular observer and that of the subject will be *consistent* with each other. The pattern invariant for the *Observer* pattern will specify that the subject state will be *consistent* with that of each of its observers.

The key point is that while the auxiliary concepts are *used* in specifying the pattern contract, they are not *defined* in the contract. Instead, the concepts will be defined as part of the *subcontract* corresponding to each system built using the pattern with the definitions being customized in a manner suitable to the particular application.

However, the pattern contract may impose *constraints* on the concepts that must be satisfied by the definitions in the subcontract. Suppose in a system built using *Observer*, s1, s2 are two possible states of the subject and o1 a state of the observer. Suppose the definitions of *Modified*() and *Consistent*() in the subcontract are such that *Consistent*(s1, o1) is *true*, *Modified*(s1, s2) is *false*, and *Consistent*(s2, o1) is *false*. This causes a problem since in this case, according to the pattern contract, if the subject state were to change from s1 to s2, notify() may not be called, therefore the observer and subject states may now be *inconsistent* with each other. The constraint in Fig. 3 (lines 7, 8) will prevent this[4].

The role contracts provide additional flexibility. Thus, although the Subject role contract states that _observers is a *set* variable containing references to the observers (attached to this subject), in an actual system, the class playing the Subject role may save these references in a different manner, say, as a linked-list. The subcontract will specify how the *class state* can be mapped to the *role state*.

The partial contract for the *Observer* pattern appears in Fig. 3. Lines 2, 3 specify the states of the Subject and Observer roles. Lines 4, 5 specify that the contract uses two auxiliary concepts as discussed above. Lines 7–8 specify the constraint on the concepts. It ensures that changes in the subject state that are not considered by the Subject to be significant enough to warrant updating the observers leave the subject state *consistent* with that of its observers. Line 10 is the pattern invariant: the state of the subject will be *Consistent* with the state of each observer object referenced by subject._observers.

Lines 11–17 specify part of the Observer role contract.

[4] Note that the constraint itself does not refer to the behavior of methods such as notify(). It is simply a required relation that must be satisfied by the concepts *Modified*() and *Consistent*(). Its definition, however, is motivated by the (expected) behaviors of methods such as notify().

```
1  pattern Observer contract
2    role Subject state Set _observers;
3    role Observer state Subject _subject;
4    concept Consistent(Subject, Observer);
5    concept Modified(Subject, Subject);
6    constraint:
7      [Consistent(s1, o1) ∧ ¬Modified(s1, s2)] ⇒
8                  Consistent(s2, o1)
9    invariant:
10     ∀ob ∈ Subject._observers : Consistent(Subject, ob)
11 role Observer contract
12   void update():
13     preserves: _subject;
14     ensures: Consistent(_subject, this)
15   others:
16     preserves: _subject;
17     ensures: Consistent(_subject@pre, this)
18 role Subject contract
19   void notify():
20     preserves: _observers;
21     ensures:
22       [¬Modified(this@pre, this)∧
23       ∀ob ∈ _observers :
24           trace.project(ob, "update").length() = 1]
25   others:
26     preserves: _observers;
27     ensures:
28       [¬Modified(this@pre, this)]∨
29       [ (trace.length() = 1)∧
30         (trace.project(this, "notify").length() = 1)]
```

Figure 3: Observer Pattern Contract (partial)

The speci cation for update() (lines 12–14) states that it does not modify _subject, and when it nishes, the state of the observer is *Consistent*() with that of its subject.

From the point of view of the pattern, the only method that the class playing the Observer role must provide is update(). But, in general, the class will have additional methods. If one of these *other* methods were to, for example, modify the value in _subject or change the state of observer so that it is no longer *consistent* with that of its subject, clearly the intent of the pattern would be destroyed. The "others" speci cation in lines 15–17 prevents such behavior; it applies to all methods in the class other than the ones explicitly named in the role contract (in this case, update()).

Lines 18–30 specify the Subject role. The ensures clause of notify() requires the *trace* of calls made during the execution of notify() be such that update() is called on each observer object to which there is a reference in _observers. Line 22 requires that when notify() nishes, the state of the subject not be *Modified*() from what it was at the start. The ensures clause of others requires that either the subject state not be *Modified*() or, if it is, that there be a call to notify() (which in turn will call update() on each observer).

3.2 Elevator-Display System Subcontract

The *subcontract* for any system built using a pattern P will consist of a *role-map* for each class C of the system that plays a role R of P. This will specify how the state of C may be mapped to the state of R and which methods of C correspond to the various (named) methods of R. The subcontract also provides de nitions for each auxiliary concept.

```
1  subpattern ElevatorDisplay : Observer
2    rolemap Elevator as Subject:
3      state:
4        _observers = {return this._displays;}
5      methods: notify() : notifyDisplays();
6    rolemap Display as Observer:
7      state:
8        _subject = {return this._myelevator;}
9      methods: update() : update();
10   auxiliary concept Modified(e1, e2) =
11     { return (e1._nf != e2._nf); }
12   auxiliary concept Consistent(e1, d1) =
13     { return ( true/false if d1 contains
14         right/wrong info about e1._nf ); }
```

Figure 4: Elevator-Display Subcontract)

The subcontract for the *Elevator-Display* system using the *Observer* contract appears in Fig. 4. Lines 2–9 specify how the Elevator class plays the Subject role and Display plays the Observer role, providing the mappings from the states/methods of the classes to the states/named methods of the roles. Lines 10–14 provide the de nitions for the auxiliary concepts. It is worth noting that the subcontract consists mostly of pieces of code; thus the de nition of each of the concepts speci es the code needed for evaluating these concepts. This makes the job of the monitoring tool straightforward because it can simply execute these de nitions whenever the values need to be computed.

3.3 System Evolution

An alert designer might be troubled by the de nition of *Modified*() (lines 10, 11, Fig. 4). Given how this concept is used in the Subject role contract (lines 22–24, 28–30, Fig. 3), this de nition clearly implies that only changes in the _nf value of the elevator are considered signi cant enough to warrant updating the display objects. Since the Elevator class (Fig. 1) includes another variable _np and the value of _np *can*, because of the openDoor() method, change even if _nf does not, such a designer is likely to ask whether this de nition of *Modified*() is indeed appropriate.

Note that a similar issue does not arise with respect to the de nition of *Consistent*(). This is because the Display (and the EnhancedDisplay) class only stores information about the oor number of an elevator. Thus there is no question of checking whether the display contains correct information about the value of _np of the elevator.

When the question about the appropriateness of the definition of *Modified*() is considered, the design team can either leave the definition alone or revise it to take account of changes in the value of _np. Suppose the team chooses the former course of action[5]. In the scenario sketched in Section 2, the maintenance team evolved the system by introducing the SafetyDisplay class (Fig. 2). This will require the subcontract to be revisited. First, a new definition will have to be provided for *Consistent*() for the case when its argument (the display object) is an instance of this new class. Clearly, here the team will require the information in the safetyDisplay object about the number of people in the elevator to match el._np. At this point, two problems will become evident. Given this new definition of *Consistent*(), the constraint on the auxiliary concepts (lines 6–8, Fig. 3) is not satisfied since the definition of *Modified*() does not take into account the value of _np. Second, if a revised definition of *Modified*() is considered, the method Elevator.openDoor() will not meet the others requirements of the Subject role (lines 25–30, Fig. 3). These problems might be caught either by (formally or semi-formally) reasoning about the system or by runtime monitoring [4]. Once discovered, different solutions may be adopted, which will involve changes in the definitions of one or both auxiliary concepts, as well as corresponding implementation changes. For example, the team might refactor the Elevator class so that a new method is charged with determining how the elevator state has changed and invokes notifyDisplays() if necessary, rather than individual methods such as move() doing this.

4 Related Work

Several authors have proposed ways to provide formal characterizations of patterns and we consider some of them. Eden *et al.* [2] propose a higher-order logic formalism and a graphical notation in which patterns are represented as formulae. The formalism can specify rich structural properties, but provides only limited support for behavioral properties which is our focus. Mikkonen [9]'s focus is on behavioral properties. Here, data classes model role objects, and guarded actions in an action system model role methods. Relations, somewhat analogous to our auxiliary concepts, can be defined among the roles, and the guarded actions can be based on these. One limitation is that by separating the data objects from the actions that operate on them, the modeling language is structurally inconsistent with the OO paradigm. Further, the resulting specifications cannot be specialized to the needs of particular systems.

Our specifications have some similarity to Helm *et al.*'s [6] *contracts*. There are also some key differences. For example, although they describe a construct analogous to our auxiliary concepts, there is no way to impose constraints on the supplied definitions. Further, although the formalism allows us to specify that certain method calls and state conditions must occur, there is no way to impose conditions on what happens in these calls. Perhaps most important, we are not aware of any other work that addresses the question of preserving the design integrity of the system as it evolves. Kobayashi and Saeki [7] consider evolution in the context of patterns but they are concerned with evolving a specialized pattern from a more general pattern, not evolution of the software over its lifecycle.

[5] Due to lack of space, we will not consider the case where the team chooses the latter course of action.

5 Conclusion

The goal of our work was to see how to preserve the *design integrity* of a system during its maintainence and evolution phases. In other words, to ensure that the system continues to be faithful to the patterns used in its initial design even as it evolves to meet changing requirements. We showed that this can be achieved by working with suitable formal specifications of patterns and their applications, and demonstrated our approach on a simple case-study.

For future work, we intend to consider larger scale case-studies. After all, patterns are most useful in the design and implementation of large systems. Especially interesting would be case-studies of distributed systems. But before we can consider these, we will have to develop contracts for patterns intended for use in such systems [11].

References

[1] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-oriented softw. arch.: A system of patterns*. Wiley, 1996.

[2] A. Eden, A. Yehudai, and J. Gil. Precise specification and application of patterns. In *Automated Softw. Eng.*, pages 143–152, 1997.

[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable OO Software*. Addison-Wesley, 1995.

[4] J.O. Hallstrom, A.R. Dalton, and N. Soundarajan. Parallel monitoring of design pattern contracts. In *Int. Conf. on Softw. Eng. and Knowledge Eng. (SEKE)*. (to appear), 2006.

[5] J.O. Hallstrom, N. Soundarajan, and B. Tyler. Amplifying the benefits of design patterns. In J. Aagedal and L. Baresi, editors, *Proc. of Fund. Approaches to Softw. Eng. (FASE)*, pages 114–129. Springer, March 2006.

[6] R. Helm, I. Holland, and D. Gangopadhyay. Contracts: Specifying behavioral compositions in object-oriented systems. In *OOPSLA-ECOOP*, pages 169–180, 1990.

[7] T Kobayashi and M Saeki. Softw. development based on pattern evolution. In *6th Asia-Pacific Softw. Eng. Conf.*, pages 18–25, 1999.

[8] B. Meyer. *OO Software Construction*. Prentice Hall, 1997.

[9] T. Mikkonen. Formalizing design patterns. In *Proceedings of 20th ICSE*, pages 115–124. IEEE Computer Society Press, 1998.

[10] L. Rising. *Pattern almanac 2000*. Addison, 2000.

[11] D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-oriented softw. architecture: Patterns for concurrent and networked objects*. Wiley, 1996.

[12] N. Soundarajan and J.O. Hallstrom. Responsibilities and rewards: Specifying design patterns. In A. Finkelstein, J. Estublier, and D. Rosenblum, editors, *Proc. of 26th Int. Conf. on Software Engineering (ICSE)*, pages 666–675. IEEE Computer Society, 2004.

An Ontology-Based Metamodel for Software Patterns

Scott Henninger, Padmapriya Ashokkumar
Univ. of Nebraska-Lincoln
Computer Science and Eng.
Lincoln, NE 68588-0112
{scotth, ashokkum}@cse.unl.edu

Abstract

Patterns have been successfully used in software design to reuse proven solutions. But the complex interconnections and the number of pattern collections is becoming a barrier for identifying relevant patterns and pattern combinations for a given design context. More formal representations of patterns are needed that allow machine processing and the creation of systematic pattern languages that guide composition of patterns into coherent design solutions. In this paper, we present a technique based on Description Logic and Semantic Web technologies to address these problems. A metamodel is presented for developing pattern languages using this technology. Usability patterns are used to demonstrate how this metamodel can be instantiated to form a pattern language for that domain. Our technique provides a computational basis for building intelligent tools that utilize patterns to support software development activities.

1 Introduction and Motivation

Software patterns represent knowledge about successful solutions to recurring problems within contextual constraints [2, 9]. Patterns are increasingly being used to not only capture and disseminate best practices, but also to turn named patterns into a shared vocabulary for expressing and communicating technical knowledge [1, 16]. Pattern usage is currently practiced informally, often through folklore and textbooks, and at best being embedded in hypertext systems supporting semantic-free "related-to" relationships [10, 11]. But the continued proliferation and interconnected nature of patterns presents a number of problems that have not been adequately addressed. First is the sheer number of patterns available. One source states there are 250 patterns for Human-Computer Interaction alone [16], and still misses a number of usability pattern collections. Second, the interconnected nature of patterns makes it very difficult to understand the potential interactions, necessary couplings, contradictions, and inconsistencies amongst the different patterns. Third, there are a number of pattern collections currently in print or on Web-based resources that differ in both format and emphasis. They have overlapping content in the form of multiple variants with subtle differences [5]. Fourth, each pattern is itself a piece of potentially complex knowledge that requires a degree of mastery [16] to apply to specific problem contexts.

These dimensions of scale, complexity, heterogeneity, and required expertise conspire to create barriers to the effective use of pattern technologies. Addressing these and other issues require increasing levels of formality for representing patterns. In particular, the development of *pattern languages* (as opposed to loosely coupled pattern collections [11]), which are systematic means of organizing patterns to provide holistic design solutions [1, 19] require levels of formal representation that have yet to be applied to pattern technologies [4].

A number of previous efforts have looked into formalizing design patterns to create formal pattern languages [6], formalize UML specifications [8], specify pattern compositions [14], organize class frameworks [12], and facilitate rigorous pattern reasoning [18]. Most of these formal pattern definitions focus on a single pattern collection, the seminal GoF patterns [9], and do not address either intra-collection (distributed) issues or issues arising with diverse pattern collections.

In this paper, we describe a *metamodel* for describing patterns. Similar to the Meta-Object Facility (MOF) [17], the purpose of this metamodel is to provide the building blocks for developing pattern languages in specific domains. We demonstrate this process by using the metamodel to create a pattern language for usability design. In the following, ontologies and the Web Ontology Language (OWL) [13] are briefly introduced. This is followed by a presentation of our pattern metamodel and an example usability pattern language based on this metamodel. The paper concludes with a discussion of contributions and future directions.

2 Ontology-based Pattern Languages

The use of ontologies to represent pattern languages is a marriage of two complementary philosophies. An objective of pattern languages is to provide the means for professionals to use a common vocabulary about design and other issues. The Semantic Web [3, 15] and its technologies, such as OWL, implement techniques for formally defining ontologies through shared vocabularies, axiomatic definitions, and formal logic to support machine-

based automated reasoning. The following describes how these technologies are used to create a metamodel for intelligent pattern languages.

2.1 OWL Description Logics

The OWL standard [13] defines a frame-based knowledge representation language with axiomatic constructs for logic-based expressivity [13]. OWL includes vocabulary for describing properties and classes that support the construction of class taxonomies and relationships between class properties and class instances. OWL Description Logic (OWL-DL) is founded on decidable fragments of first order logic and axiomatic definitions that can be used by Reasoners to infer new facts and to check the consistency of resulting ontologies. OWL properties are predicates that operate on subjects (domains) and map to an object (range). Range values can be restricted through various axiomatic class construction operators. For example, if a Login pattern is defined as the set of pattern instances that are a subclass of the UIAuthenticate pattern and that all values of the hasWidgets property are instances of TextBox and Button classes, then a local restriction would be created (in DL form):

Login ⊑ UIAuthenticate ⊓
∀hasWidgets(TextBox ⊔ Button)

3 Usability Design Pattern Metamodel

The purpose of a metamodel is to define the basic building blocks and rules for constructing well-formed models within some domain of interest [17]. A pattern language metamodel will therefore provide the basic building blocks for creating types of patterns, or pattern languages. Figure 1 depicts a pattern metamodel designed to support the creation of usability patterns, named *UIPatternLanguages*. These patterns describe best practices for interface usability. The metamodel builds on the Pattern Language Markup Language PLML [7], extending the XML DTD definitions to OWL and including properties to support creating semantically meaningful pattern descriptions and relationships between patterns that facilitate computational reasoning..

In addition to the *PLMLPatternFormat*, the UIPatternLanguages metamodel also builds on the *PatternCore* metamodel (see ① in Figure 1). Window ② in Figure 1 shows the resulting properties for the UIPatternLanguages metamodel. Each of the properties listed is defined by domain and range restrictions. For example, the *requiresPattern* property selected in Figure 1 is defined as a property of UIPatternLanguage, creating a relationship between pattern instances of that type (the Domain, see ③) and any other pattern concept defined through the root pattern concept, PatternCore (the Range,

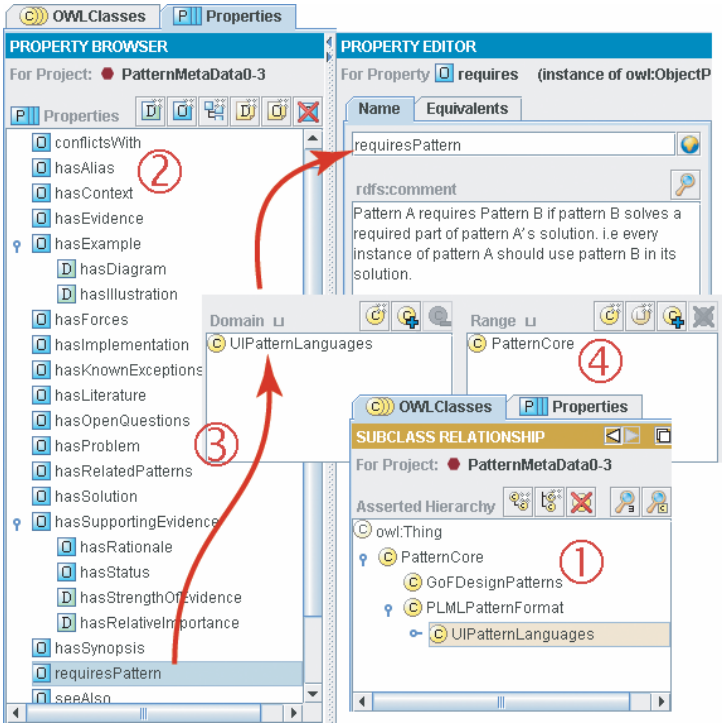


Figure 1: Property and Class Views of the Usability Pattern

see ④). This defines a semantic relationship where the use of one UI pattern will require the use of another pattern). The range of values for this property can be further specified through local restrictions for a more precise definition of the type of pattern needed.

3.1 The PatternCore Metamodel

The PatternCore metamodel defines the original pattern properties [2] that are included for all pattern definitions. Some of the properties include:

- *hasProblem*: describes the need of an actor (person/system) for which the pattern was designed. For example, the Problem of an Ecommerce Website Pattern is to provide online users a virtual shopping experience. The actor here is the Ecommerce Website owner (company/individual).
- *hasSolution:* provides a set of instructions to solve the problem described in the Problem attribute.
- *hasContext:* a set of applicable design conditions in which the pattern can be most usefully ("naturally") applied [7].
- *hasRationale:* provides principled reasons from behavioral psychology, cognitive psychology or another science for why the solution is effective [7].
- *hasForces:* discusses constraints and tradeoffs in choosing the solution suggested in the pattern.

While these generic properties are defined for every pattern, range restrictions are used to define local semantics for a pattern metamodel. For example, while

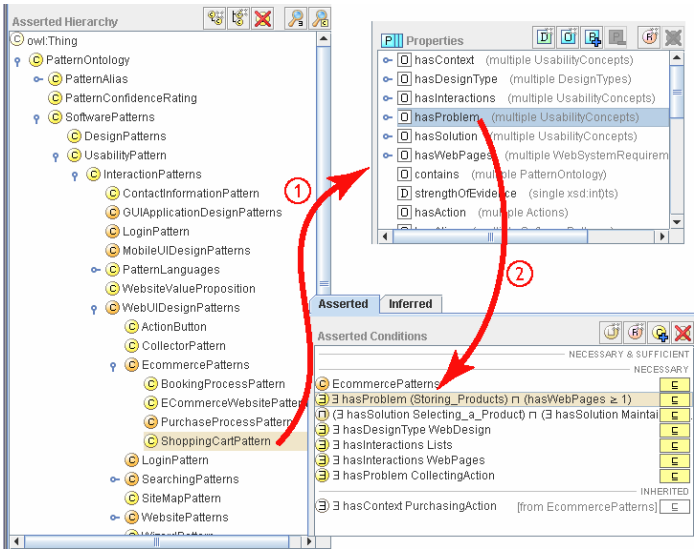


Figure 2: OWL description of a usability design pattern.

GOFDesignPatterns defines hasProblem to be restricted to design pattern concepts, UIPatternLanguages will restrict the same hasProblem property to a range of usability concepts. Hence the hasProblem property, as well as the other metamodel properties, can be given precise semantic definitions for the type of metamodel being defined.

3.2 Extending the PatternCore for Semantic Relationships

Instead of being restricted to using generic "related to" relationships between patterns, defining the metamodel with OWL, which treats properties as classes, allows the definition of many subtypes of pattern relationships. At the metamodel level, we define several types of pattern relationships to facilitate semantic relationships between patterns supporting pattern languages, including:

- *uses*: Pattern A uses pattern B if the usage is optional In other words not every instance of pattern A uses pattern B, some variants do and some do not [21].
- *requires*: Pattern A requires Pattern B if pattern B is a required part of A's solution. i.e. every instance of pattern A should include pattern B in the solution [21].
- *alternative*: Two patterns are alternative if they have the same problem and context but different solution.
- *conflictsWith*: Pattern A conflicts with pattern B if they should not be used together in a design.

4 Using the Metamodel

The domain and range definitions of the *metamodel* properties are purposefully defined to be as general as possible, yet restrict models to the domain of interest. Models created as instances of the metamodel then adapt metamodel elements to define specific pattern concepts. For example, suppose a pattern language was being built for Web-based browser interfaces. An instance of the UIPatternLanguages metamodel would be created, inheriting the generic usability properties shown in window ② of Figure 1. Then properties, such as the requiresPattern property (see ③ and ④ of Figure 1), are further restricted to denote characteristics required by Web-based interfaces. For the requiredPattern property, the range could be restricted to values in the WebUIDesignPatterns, where WebUIDesignPatterns is a subset of (subsumed by) PatternCore, which is defined by the metamodel.

Even after creating a specialized metamodel, additional restrictions can be used to refine the semantic definition of a given pattern relationship. Suppose one wants to state that when a large hierarchical Web site is developed, any interface that uses the "SelectAction" pattern must use either the "Bread Crumbs" pattern or a combination of the "Double Tab" pattern and the "Main Navigation" pattern [20]. Then the SelectAction pattern would be defined in part by the restriction:

$$\text{SelectAction} \sqsubseteq \exists \text{requiresPattern}(\text{BreadCrumbs} \sqcup (\text{DoubleTab} \sqcap \text{MainNavigation}))$$

Therefore, a valid member of a SelectAction pattern would have at least one requiresPattern property with a value from BreadCrumbs or the DoubleTab and MainNavigation patterns. This demonstrates how OWL description logic can be used to create a *pattern language* where the selection of one pattern leads to other patterns until a composition of related patterns is created that represents a set of known best practices for a specific problem context.

4.1 A Web-Based Usability Pattern Ontology

Figure 2 shows part of an instance of the usability pattern metamodel specifically designed for Web-based e-commerce. This shows a "Shopping Cart" pattern as a solution to the problem of storing products from multiple web pages that a user has selected items from. This is represented by the statement in the line pointed to by arrow ② with the property restriction ∃hasProblem (Storing_Products ⊓ hasWebPages ≥ 1). This describes the problem though a combination of existential quantification on the hasProblem property and a cardinality restriction on the number of Web pages in the problem statement (hasWebPages).

The ShoppingCartPattern utilizes many of the properties from the metamodel (a partial list is shown by the arrow marked ①), including the hasSolution property. Following arrow ② reveals that in addition to the metamodel definition for hasProblem, the pattern uses a number of other properties to define the pattern and requirements on potential solutions.

5 Reasoning with OWL Models

Describing pattern attributes using this formal medium facilitates automated inferences of relationships between

patterns. For example, standard OWL reasoners can infer that any pattern that has the same problem as the Shopping Cart pattern is an alternative pattern when the contexts are same and the solutions are different. Further, if these restrictions are stated as equivalence axioms, a Reasoner can infer classification. Although not shown here, if the restriction "∃hasDesignType WebDesign" were stated as a Necessary and Sufficient condition for membership in the ShoppingCartPattern concept, then all new patterns specifying one or more relationships of type WebDesign would be inferred to be a Shopping Cart Pattern. More complex restrictions can be used to precisely define class membership as deeply as deemed necessary by ontology or pattern designers.

6 Conclusions and Future Work

In this paper, we have shown how formally defined ontologies using description logic defined in OWL can be used to support a structured representation and shared vocabulary of interconnected pattern languages. The metamodel creates an infrastructure upon which distributed pattern languages and pattern-based tools capable of automated axiomatic reasoning can be built.

Continued research is needed to further understand the complexities of creating repositories of usability patterns and applying them proactively in the software development process. We have taken small steps in this direction, and hope that future validation and use of our approach provides more information about the contextual factors that impact usability knowledge.

7 References

[1] C. Alexander, "The Origins of Pattern Theory: the Future of the Theory, and the Generation of a Living World," OOPSLA 1996 Keynote Address, http://www.patternlanguage.com/archive/ieee/ieeetext.htm, 1996.

[2] C. Alexander, S. Ishikawa, M. Silverstein, A Pattern Language: Towns, Buildings, Construction. Oxford University Press, New York, 1977.

[3] T. Berners-Lee, "Semantic Web Roadmap," W3C Semantic Web Vision Statement, http://www.w3.org/DesignIssues/Semantic.html, 1998.

[4] J. Dietrich, C. Elgar, "A Formal Description of Design Patterns Using OWL," 2005 Australian Software Engineering Conference (ASWEC'05), pp. 243-250, 2005.

[5] J. Dietrich, C. Elgar, "Towards a Web of Patterns," Proc. Semantic Web Enabled Software Engineering (SWESE), 117-132, Galway, Ireland, 2005.

[6] A. H. Eden, "Formal Specification of Object Oriented Design," Int'l Conf. on Multidisciplinary Design in Engineering CSME-MDE 2001, Montreal, Canada, on-line at: http://www.eden-study.org/articles/2001/csme.pdf, 2001.

[7] S. Fincher, "Perspectives on HCI patterns: concepts and tools (introducing PLML)," Interfaces, 56, pp. 26-28, 2003, on-line at: http://www.bcs-hci.org.uk/interfaces.html.

[8] R. B. France, D.-K. Kim, S. Ghosh, E. Song, "A UML-based pattern specification technique," IEEE Trans. Software Engineering, 30(3), pp. 193- 206, 2004.

[9] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, 1995.

[10] S. Henninger, "An Organizational Learning Method for Applying Usability Guidelines and Patterns," in Engineering for Human-Computer Interaction (revised papers, EHCI 2001), vol. LNCS 2254, Springer, 2001, pp. 141-155.

[11] S. Henninger, P. Ashokkumar, "An Ontology-Based Infrastructure for Usability Design Patterns," Proc. Semantic Web Enabled Software Engineering (SWESE), Galway, Ireland, pp. 41-55, 2005.

[12] K. Lano, J. C. Bicarregui, S. Goldsack, "Formalising Design Patterns," 1st BCS-FACS Northern Formal Methods Workshop, on-line at: http://ewic.bcs.org/conferences/1996/formalmethods/papers/paper11.htm, 1996.

[13] D. L. McGuinness, F. van Harmelen, "OWL Web Ontology Language Overview," W3 Consortium, http://www.w3.org/TR/owl-features/, Last Updated February 10, 2004.

[14] T. Mikkonen, "Formalizing Design Patterns," Int'l Conf. Software Engineering, pp. 115–124, 1998.

[15] E. Miller, R. Swick, D. Brickley, B. McBride, J. Hendler, G. Schreiber, D. Wood, D. Connolly, "W3C Semantic Web," World-Wide Web Consortium, http://www.w3.org/2001/sw/, March 15, 2006.

[16] S. Montero, P. Díaz, I. Aedo, "A Semantic Representation for Domain-Specific Patterns," in Int'l Symp. on Metainformatics, U. K. Wiil, Ed., Springer-Verlag, LNCS 3511, 2005, pp. 129-140.

[17] OMG, "MetaObjectFacility (MOF) Specification, v1.4," Object Management Group, http://www.omg.org/technology/documents/modeling_spec_catalog.htm#MOF, Last Updated April, 2002.

[18] T. Taibi, D. C. Ling Ngo, "Formal Specification of Design Patterns – A Balanced Approach," Journal of Object Technology, 2(4), pp. 127-140, 2003, on-line at: http://www.jot.fm/issues/issue_2003_07/article4.

[19] J. Tidwell, "The Gang of Four are Guilty," http://www.mit.edu/~jtidwell/gof_are_guilty.html, 1999.

[20] M. van Welie, "Patterns in Interaction Design," http://www.welie.com/, Updated 05-25-2005, 2005.

[21] W. Zimmer, "Relationships Between Design Patterns," in Pattern Languages of Program Design, J. O. Coplien, D. C. Schmidt, Eds., 1995, pp. 345-364.

A formalism of ontology to support a software maintenance knowledge-based system

Alain April[1] , Jean-Marc Desharnais[1], and Reiner Dumke[2]

[1] *École de Technologie Supérieure, 1100 Notre-Dame West, Montreal, Canada*
email: alain.april@etsmtl.ca , Jean-Marc.Desharnais@etsmtl.ca
[2] *Department of Informatik, Otto-von-Guericke Universtity of Magdeburg, Germany*
email: dumke@ivs.cs.uni-magdeburg.de

Keywords: expert system, decision support, software maintenance maturity model, formal definition of process model.

Abstract: Maintaining and supporting the software of an organization is not an easy task, and software maintainers do not currently have access to tools to evaluate strategies for improving the specific activities of software maintenance. This article presents a knowledge-based system which helps in locating best practices in a software maintenance capability maturity model (***S^{3m}***). The contributions of this paper are: 1) To formalise the software maintenance ontology, 2) to instrument the maturity model with a support tool to aid software maintenance practitioners in locating specific best practices; and 3) to describe the knowledge-based approach and system overview used by the research team.

1. INTRODUCTION

Knowledge transfer of a large number of best practices, described in a maturity model, has proved difficult (Abran et al., 2004). A potential solution to this problem is to develop a knowledge-based system that the underlying process model used by the maturity model. According to van Heijst, there are at least five different types of knowledge to be taken into account when constructing such a system: tasks, problem-solving methods, inferences, the ontology and the domain knowledge[1]. For van Heijst, domain knowledge refers to a collection of statements about the domain (Van Heijst et al., 1997). At a high level, the ontology refers to a part of the software maintenance ontology proposed by (Kitchenham et al., 1999) presented in section 2. A formalisation of the software maintenance ontology follows in section 3. The inferences, problem-solving methods and tasks are described in section 4 showing the task analysis for the proposed knowledge-based system. The tool environment (showing a user interface layout of the knowledge-based system) followed by a conclusion, as well as future work, are presented in sections 5 and 6.

2. ONTOLOGY OF SOFTWARE MAINTENANCE

Software maintainers experience a number of problems. Dekleva ranked them in his 1992 survey. Key to helping software maintainers would be to provide them with ways of resolving their problems by leading them to documented best practices (April et al., 2005). There is a growing number of sources where software maintainers can look for best practices, a major challenge being to encourage these sources to use the same terminology, process models and international standards. The practices used by maintainers need to show them how to meet their daily service goals. While these practices are most often described within their corresponding operational and support processes, and consist of numerous procedures, a very large number of problem-solving practices could be presented in a KBS which would answer their many questions about those problems.

[1] van Heijst uses the different types of knowledge in a more generic way than we do in this document, and these have been adapted for us by Desharnais, J.-M.,Application de la mesure fonctionnelle COSMIC-FFP: une approche cognitive, UQAM,2004 Montréal

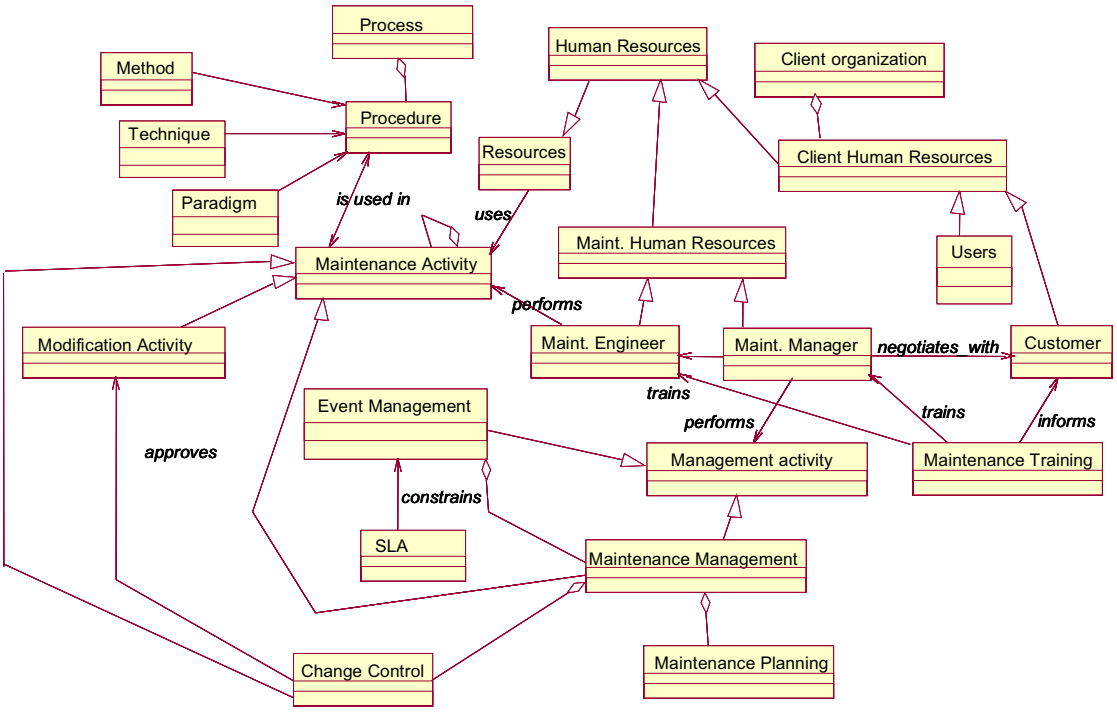


Figure 1: Part of the software maintenance ontology of (Kitchenham et al., 1999)

As part of the ***SM***^{xpert} KBS we elected to implement only a subset of the ontology developed by Kitchenham et al. (Kitchenham et al., 1999) and Ruiz et al. (Ruiz et al., 2004) for the initial trial of this research project. Figure 1 describes the different maintenance concepts considered surrounding a software maintenance activity. Software maintenance is highly event-driven, which means that some maintenance activities are unscheduled and can interrupt ongoing work. This subset of the ontology represents many, but not all, the concepts involved in responding to the questions related to the first problem identified by Dekleva: 'Managing fast-changing priorities.' Maintainers agree that this is the most important problem they face. How can they handle the fast-changing priorities of the customer? Solutions to this problem are likely to be found by using many paths through the maintenance concepts of the ontology. Navigation through these concepts should lead to associated concepts which are conceptually linked and likely to contribute to a solution, like the need for better event management, change control, maintenance planning, Service Level Agreements, maintenance manager negotiation, training, procedures, and so forth. Many more concepts must be involved to contribute to all aspects of the solution, but our purpose is to show the utility of a KBS in the software maintenance domain, and it therefore starts with a constrained number of concepts. Maturity models typically include the detailed best practices that could be of help in solving this type of problem. The main issue is that the best practice locations and their interrelationships are hidden in the layered architecture of the maturity model, specifically in its process domains, KPAs and roadmaps. It is therefore necessary to find a way to link this layered architecture with the maintenance concepts of the ontology and proceed to analyze the tasks required to build a KBS to support the maintainers in their quest for solutions. The next section presents a formalisation of software maintenance ontology using a representation based on set theory. The formal rules are used, as part of the knowledge based representation, to better define the maintenance domain concepts of the ontology for the task analysis activity that will be presented in section 4.

3. ONTOLOGY FORMALISATION

'Processes can be defined at different levels of abstraction (e.g., generic definitions vs. tailored definitions, descriptive vs. prescriptive vs. proscriptive) (Pfleeger, 2001)'. Various elements of a process can be defined, for example, activities, products (artefacts), and resources. SWEBOK (Abran et al., 2004) reports also that 'detailed frameworks that structure the types of information required to define processes are described in (Madhavji, 1994)'.

There is a number of notations being used to define processes. A key difference among them is in the types of information they each define, capture and use. Common approaches that software engineer use in their daily work are presented in the SWEBOK as: data flow diagrams, Statecharts, ETVX and Actor-Dependency modeling. Looking at recent software engineering process descriptions by Prof. Dr. Dumke (Dumke et al., 2005) reveals the many artefacts that compose a typical software product.

Taking the same descriptive approach as Prof. Dr. Dumke the software product maintained can le libeled $\boldsymbol{SP_M}$. This product is now maintained by a given software maintenance process labeled $\boldsymbol{SM}$. This software maintenance process is using a number of maintenance activities labeled $\boldsymbol{A_{SM}}$. These activities are carried out using the maintenance resources labeled $\boldsymbol{R_{SM}}$.

This expression can be written as:

$\boldsymbol{SM} = (A_{SM}, R_{SM}) = (\{maintenanceActivities, maintenanceResources\} \cup \boldsymbol{SP_M})$

The software product maintained is initially defined as a (software) system containing a number of artefacts that are sets of programs and sets of documentations:

$\boldsymbol{SP_M} = (M_{SP}, R_{SP}) = (artefacts, R_{SP}) = (\{programs, documentations\}, \boldsymbol{R_{SP}})$

The two sets are divided in the following elements or components (without achieving completeness):

programs ⊆ *{sourceCode, objectCode, macro, script, plugIn}*

documentations = {userManual, referenceManual, developmentDocumentation}

and $\boldsymbol{R_{SP}}$ describes the set of the relations over the $\boldsymbol{SP_M}$ elements. The given subsets could be described as follows:

developmentDocumentation = {documentationElements} ={productRequirements, productSpecification, productDesign, implementationDescription}

documentationElements ⊆ *{model, chart, architecture, diagram, sizeEstimation, reviewReport, auditReport, testCase, testScript, pseudoCode, extensionDescription, qualityReport }*

productRequirements = systemRequirement ⊆ *{functionalRequirements, qualityRequirements, platformRequirements}*

functionalRequirements ⊆ *{businessRule, function Map, informationSequence, control}*

qualityRequirements ⊆ *{maintainability} = {analysability, changeability, stability, testability}*

platformRequirements ⊆ *{systemSoftwareRequirements, hardwareServerRequirements, telecommunicationsBandwidthRequirements, peripheralDeviceRequirements}*

The software maintenance activities of the ISO/IEC 14764 can also be defined formally. "The maintenance activities are classified, according to ISO/IEC 14764, into investigations, modification and retirement activities. The modification activities consist of basically two types: Correction and Improvement. The first one belongs to corrective maintenance, where mistakes are eliminated. The second one is in charge of preventing problems (preventive maintenance), implementing changes in the requirements (perfective maintenance) or changing aspects of implementation (adaptive maintenance)." (Ruiz et al. 2004, p.335). Software maintenance processes can be defined by describing its activities. In this case the non-managerial maintenance activities of the software maintenance ontology by Ruiz et al. can be defined as:

$\boldsymbol{SM} = (A_{SM}, R_{SM}) = (\{\boldsymbol{maintenanceActivities}, maintenanceResources\} \cup \boldsymbol{SP_M})$ where:

maintenanceActivities *= {investigation, modificationActivities, retirement}*

modificationActivities *= {corrective, improvement}*

improvement = {adaptive, perfective, preventive}

Accordingly, some of the examples of the relations in $\boldsymbol{R_{SM}}$ could be derived in the following manner. An investigation can be expressed as the requirements identified on a specific investigation request with the use of the software maintenance investigation workflow resulting in findings to answer the questions asked on the request. In this maintenance service the software product is not modified:

$$r^{(investigation)}_{SM} \in R_{SM} : \boldsymbol{SP_M} \times investigationRequirements \times maintenanceInvestigationWorkflow \rightarrow investigationFindings$$

The definition of a corrective service to the maintained software can be expressed as the requirements identifying what has to be corrected (corrective requirements) with the use of the correction workflow resulting in a corrected software product still under maintenance:

$$r^{(corrective)}_{SM} \in R_{SM} : \boldsymbol{SP_M} \times correctiveRequirements \times maintenanceCorrectionWorkflow \rightarrow \boldsymbol{SP_M}^{(corrected)}$$

The definition of an adaptive maintenance to the maintained software can be expressed as the requirements that specify what has to be added (new requirements) with the use of the evolution workflow resulting in an adapted software product still under maintenance.

$$r^{(adaptive)}_{SM} \in R_{SM} : \boldsymbol{SP_M} \times newrequirements \times maintenanceEvolutionWorkflow \rightarrow \boldsymbol{SP_M}^{(adapted)}$$

The definition of a perfective maintenance to the software can be expressed as the requirements identifying the change to an existing requirements (perfective requirement) with the use of the evolution workflow. This results in an optimized software product under maintenance:

$$r^{(perfective)}_{SM} \in R_{SM} : \boldsymbol{SP_M} \times perfectiveRequirements \times maintenanceEvolutionWorkflow \rightarrow \boldsymbol{SP_M}^{(optimized)}$$

The definition of a preventive maintenance to the software can be expressed as the requirements that do not affect a user requirement but only the implementation of the software with the use of the evolution workflow resulting in a modified software product under maintenance:

$$r^{(preventive)}_{SM} \in R_{SM} : \boldsymbol{SP_M} \times preventiveRequirements \times maintenanceEvolutionWorkflow \rightarrow \boldsymbol{SP_M}^{(modified)}$$

The definition of a retirement of a software can be expressed as the requirements to retire a software with the use of the maintenance retirement workflow resulting in a retired software product:

$$r^{(migration)}_{SA} \in R_{SA} : \boldsymbol{SP_M} \times retirementSpecifications \times maintenanceRetirementWorkflow \rightarrow \boldsymbol{SP_M}^{(retired)}$$

The maintainers' resources $\boldsymbol{R_{SM}}$ are defined using the same approach. To address the concerns specific to the maintainer, a distinct maintenance body of knowledge and ontology needs to be defined formally. This formalisation is also useful for the development of the knowledge-based rules that support the process model.

The next section describes the navigation concepts that have been implemented in $\boldsymbol{SM^{xpert}}$. The user of the KBS navigates using a sequence of tasks that will lead him through a further sequence of tasks.

4. TASK ANALYSIS

According to (Van Heijst et al., 1997), the first activity in the construction of a KBS is the definition of task analysis. Task analysis begins, at a high level, with a definition of an index of terms. This index includes words commonly used in software engineering (see Figure 2). From this index, a subset of more restrictive words is identified. This subset is a list of keywords recognized specifically in software maintenance. Each keyword is then connected to one or more maintenance concepts. A maintenance concept, in software maintenance, is a concept found in the Software Maintenance Body of Knowledge and ontology. Using the software maintenance ontology, every software maintenance problem identified by Dekleva has been linked to themes (questions) which help the user of the KBS to navigate to the part of the maturity model that will propose recommendations in the form of best practices. Expanding the 5 high-level tasks in Figure

2, we propose 15 detailed tasks which will help identify a best practice related to the SM^{mm}. The link between the maintenance concepts and the maturity model is made in the themes concept. Themes are questions which have been developed to hop from node to node in the ontology. Themes concept can send the user to another theme, to another maintenance concept (*up arrow*), or, finally, to a recommendation of the maturity model (*down arrow*). At step 11, a number of themes, in the form of questions, are presented to the user to guide him through the network of maintenance concepts. For every best practice, there are a number of themes (or choices) from which the user can select (also called facts) which will lead to a specific recommendation. There are also a number of sub-tasks related to the maintenance processes and the maintenance best practices. This step-by-step process corresponds to the establishment of a diagnosis on the basis of the identification of symptoms. It indicates probabilities of occurrence of a specific software maintenance problem. No symptom is sufficient by itself to confirm the existence of a specific problem. This is why we should use the word "diagnosis".

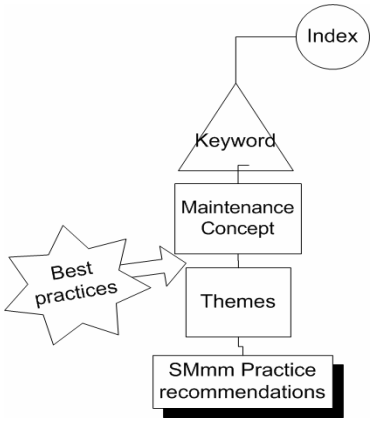


Figure 2: High-level view of ***SM**^xpert*

The task model is used to help "diagnose" the current maintenance practice and map it to the maintenance model. The KBS can help the user answer the following question: How do we accept or reject a new maintenance request?

5. TOOL ENVIRONMENT

The ***SM**^xpert* KBS was built using Java script and XML, and supports the ***SM**^mm*. The architecture, design and implementation details of this KBS are similar to those of the COSMIC KBS (Desharnais, 2003) which was developed as a diagnostic tool to help IT personnel in the estimation of functional size. The design of the KBS is based on using both the case-based and ruled-based approaches (Desharnais et al., 2002). The ***SM**^xpert* KBS was developed by two Master's degree students from the University of Namur, Belgium, during a research exchange program with our university (Desharnais et al., 2004). There is still a great deal of work required to populate the knowledge base for all the ***SM**^mm* practices to allow users to obtain answers to all the software maintenance problems identified by Dekleva. Figure 3 shows an example of the knowledge-based system user interface layout. In this case, the user requests a recommendation in a case where the service request is very costly. A number of questions (themes) are asked by the system. According to the answers, there will be a specific recommendation which could either suggest further research or provide an opinion. There are also interfaces for both the administrator and the expert. The administrator interface manages access to ***SM**^Xpert*, while the expert interface gives the expert the option of adding new keywords, concepts, cases, themes and recommendations.

6. CONCLUSION AND FUTURE WORK

Identifying the best practices in a maturity model is a difficult task, considering their number and the multiple appropriate answers associated with each of them. Our hypothesis is that a KBS could help in finding an appropriate recommendation. The next step in this research project is to populate the KBS, validate the results with experts in the domain and determine whether or not the KBS is a useful support tool for training on the content of the maturity model. It will be also necessary to improve the interface, mainly for the sake of the expert.

Figure 3: ***SM**^xpert* user interface layout

REFERENCES

Abran, A., Moore, J. W., Bourque, P., Dupuis, R. and Tripp, L., *Guide for the Software Engineering Body of Knowledge (SWEBOK),* Ironman version, IEEE Computer Society Press: Los Alamitos, CA, 2004; 6-1-6-15, Montréal, http://www.swebok.org [7 May 2006].

April, A., Huffman Hayes, J., Abran, A., and Dumke, R. Software Maintenance Maturity Model (***SM**^mm*): The software maintenance process model*, Journal of Software Maintenance and Evolution: Research and Practice* 2005;**17**(3):197–30.

Dekleva, S. M. *Delphi Study of Software Maintenance Problems,* International Conference on Software Maintenance (CSM 1992),1992, IEEE Computer Society Press: Los Alamitos CA.

Desharnais, J.-M., *Application de la mesure fonctionnelle COSMIC-FFP: une approche cognitive*, UQAM, Montréal, 2003.

Desharnais, J.-M., *Application de la mesure fonctionnelle COSMIC-FFP: une approche cognitive*, UQAM, Montréal, 2004.

Desharnais, J.-M., Abran, A., Mayers, A., Buglione, L. and Bevo, V. *Knowledge Modeling for the Design of a KBS in the Functional Size Measurement Domain,* KES 2002, IOS Press, Crema, Italy.

Desharnais, J. M., Abran, A., Mayers, A., Vilz, J. and Gruselin, F. (2004), *Verification and validation of a knowledge base system*, KI, Special Issue on Software Engineering for Knowledge-based Systems, Germany*,* **3**.

Dumke, R.; Schmietendorf, A.; Zuse, H.: Formal Descriptions of Software Measurement and Evaluations, Preprint: Fakultät für Informatik, University of Magdeburg, 2005.

Kitchenham, B. and et al. (1999), *Towards an Ontology of Software Maintenance*, J. Softw. Maint:Res. Parct., **11**(6):365–389.

Madhavji, N.; Hoeltje, D.; Hong, W.; Bruchhaus, T.: Elicit: A method for eliciting process models, Proceedings of the third international conference on the software process, 1994.

Pfleeger, S.L.: Software Engineering—Theory and Practice*.* Prentice Hall, 2nd ed., 2001; 659 p.

Ruiz, F.; Vizcaíno, A.; Piattini, M.; García,F.: An Ontology for the Management of Software Maintenance Projects*, International Journal of Software Engineering and Knowledge Engineering*, 1**4**(3), 2004; 323**–**349.

Van Heijst, G., Schreiber, A. T. and Wielinga, A., Using *Explicit Ontologies in KBS Development*, 2003 University of Amsterdam, Department of Social Science Informatics, Amsterdam, 1997.

Ontology-driven Model for Knowledge-based Software Engineering

Thaddeus S[*], Kasmir Raja S.V[+]
[*]Sacred Heart College, Tirupattur, India, thad@boscoits.com
[+]SRM Institute of Science and Technology, Chennai, India, svkr@srmuniv.ac.in

Abstract

Software engineering performance can improve if knowledge generated during software development is captured and structured formally for reuse. The developers need a knowledge framework to produce and consume knowledge within software engineering environment. Ontologies have matured to provide knowledge representation and retrieval with semantic completeness and sound reasoning. The explicit knowledge of software process and application domains in the context of software projects can be semantically expressed using ontological concepts and relations. Tacit knowledge from experience can be accumulated as instances of the respective concepts. This paper proposes a model for ontology-driven software engineering environment where explicit and tacit knowledge are accumulated using ontologies represented by OWL (Web Ontology Language). The preliminary results of a pilot implementation of this model have confirmed that structuring of application domain and process knowledge facilitates a better environment for domain modeling, process improvement and instant access to knowledge resources.

Keywords: *Knowledge-based Software Engineering, Empirical Software Engineering, Ontology, Software Process, Domain Modeling*

1. Introduction

The real asset of a software organization is its intellectual capital, developed and enriched by the software engineers. It is inherited as tacit knowledge by the organization. This knowledge is dynamic and evolving due to the changing nature of the software systems and the advancement of technologies. The challenge is to make tacit knowledge of software engineering explicit, and reuse it for future projects and learning as an organizational asset. The management of software engineering knowledge and leveraging benefits from it has become an active field of research in software and knowledge engineering research communities [1]. The tacit knowledge developed during a software project regarding the application domain or software process can be converted to explicit knowledge facilitating domain modeling or process improvement for future projects. A major bottleneck in knowledge management is the formalism to acquire, store and retrieve knowledge sources providing a shared understanding among all human and software agents. Ontologies in computer science have provided an effective solution and can be used to build up a knowledge base providing semantic completeness and sound reasoning. This paper presents a model for knowledge-based software engineering where a knowledge repository is developed for the application and software process domains using distinct ontologies. Section 2 of this paper presents the techniques from software and knowledge engineering that are adopted in this model. Section 3 describes stages of the model while Section 4 gives a pilot implementation to establish proof-of-concept as part of an on-going research.

2. Motive and Approach

It is evident that software and software development are inherently complex. However, an accomplished software project brings in a pool of knowledge in application domain and software engineering. This knowledge gained during a project life-cycle is seldom transferred to latter projects and/or development teams. While most software engineering literature speaks about management of software process knowledge alone [2,3,4], a few address both application domain and process knowledge [5]. The Experience Factory model [6] treats the knowledge organization separate from the software development organization. Based on the Experience Factory model, knowledge management methodologies and tools have evolved which focus mainly on process knowledge gathered from project experience [7,3]. Creating a software engineering

environment where product and process knowledge are handled using ontology is the base for the proposed model. Hence, we have named the model as Ontology-driven Software Engineering Environment (OSEE). The prime objective is to integrate knowledge acquisition activities with software development, so that the developers are the real consumers and producers of software engineering knowledge. The knowledge base must provide information for domain modeling and software process improvement in future projects.

A. Domain Modeling

A formal method for requirements engineering (known as Reference Model for requirements and specifications [8]) depicts a domain and its software specification in five artifacts {W, R, S, P, M}. Where {W} represents the domain knowledge of the environment with known facts and rules, {R} indicates the needs of a system from an end-user point of view, {S} stands for the specification for a programmer to build up a software system, {P} for the product to be developed and {M} for the machine environment (the target platform) in which the system is deployed. Based on this model, Diaz-Herrera proposed a framework for domain modeling where various combinations of W, R and S provide different views of the domain model [9]. If we restrict the domain to a particular scope {d} and gather user requirements, then {Wd+R} is the abstract domain model. The various entities in a domain and their relations {Wd} in a specific context are mapped as concepts in ontology. The user requirements specifications {R} elicited in different contexts are stored as instances of ontological concepts. Thus, the explicit knowledge of the domain is captured in concepts and their relations. Where as, the tacit knowledge from end-users and system analysts is stored as instances. To represent user requirements {R} of the domain, use-case approach of Unified Modeling Language (UML) can be used with a predefined format holding attributes such as description, preconditions, basic flow of events, alternate flow, exceptional flow and post conditions [10].

B. Software Process Engineering

Software process engineering refers to modeling, evaluation and improvement of software process to increase the productivity and quality of software development in an organization. Different types of process modeling based on the level of abstraction and goals and numerous notations are used for process engineering [11]. This can also be done using ontology where process elements such as activities, artifacts (both input and output) and roles are mapped as concepts. Tacit knowledge gained by adopting the process in a number of projects is stored as instances of process elements. The process engineer can evaluate and enforce process improvement by analyzing these instances. The instances are represented using case-based reasoning, an artificial intelligence technique to store and retrieve solution for a problem based on past experience [12]. Any past experience is stored in the form of solved problems organized as 'cases'. Each case has two parts: the description and solution. Process experience is stored as problem/solution cases, so that retrieval and updating of cases are possible.

C. Ontology Development

Ontology provides a framework to represent any domain knowledge as 'explicit, formal specification of shared conceptualization' [13]. Ontology enables shared knowledge and reuse where information resources can be communicated between human or software agents. Semantic relationships in ontologies are machine readable, which enable automatic representation of facts and rules, and querying in a subject domain.

We have selected OWL (Web Ontology Language) as the ontology representation language for our model as it has matured to become an accepted standard by World Wide Web Consortium. OWL is a consolidation of its preceding languages namely SHOE, OIL, DAML+OIL, RDF and RDF(S) [14]. Although OWL has evolved in the context of Semantic Web [15], it is an apt choice. If the knowledge of software development distributed across multiple sites has to be captured, Semantic Web becomes an added boon. OWL provides a rich set of constructs to define classes as named or anonymous using property restrictions, logical operations or class axioms [16]. The semantics of OWL ontology is from Description Logic, which is a derivative of First Order Predicate Logic, enabling sound and decidable reasoning facility [17]. OWL has been further enhanced to incorporate rules about concepts and properties using Semantic Web Rule Language (SWRL), so that the expressivity of it can be improved [18].

The knowledge base of OSEE is defined as a quintuple {T, I, ≤, *conf*, B} [19] where T is the set of concepts (T_C) and relation types (T_R) such that $T = T_C \cup T_R$; I is the set of instances of T_C; "≤" is the subsumption relation of T, defined as ≤: $(T_C \times T_C) \cup (T_R \times T_R) \rightarrow$ {true, false} such that "c ≤ d" means "c is a subtype of d"; *Conf* is the "conformity" relation that relates each instance in T_C to one, and one only, concept type in T_C; B is the Canonical Basis function associating each relation type with the concept types that may be used in that relation, so that, B: $T_R \rightarrow \varphi(T_C)$ where: $\forall r \in T_R \quad B(r) = \{c_{1,\ldots} c_n\}$ where n is a variable associated with r and the set $\{c_{1,\ldots} c_n\}$ is an ordered set of arguments for r. Using this scheme, ontology for the software process and each application domain are developed in a formal manner with OWL as the representation language.

D. Inference Engine

Inference engine is an integral component of OSEE knowledge base. Queries expressed in natural language must match with the existing ontologies. The given query (Q) is categorized as a set of terms $\{q_1, q_2, \ldots q_n\}$, which in turn is expressed as a sub-ontology. This sub-ontology must find a match in OSEE ontology. It is a great challenge to match semantic similarities between concepts and to represent them logically. A probabilistic method can be applied to do ontology mapping and find the most similar concepts. An algorithm for ontology mapping given by Zhongli Ding et al suits the requirement [20]. Two ontologies can be mapped by converting them into BayesOWL Network (BN) to which probabilistic measures are added. The translated BNs facilitate ontology reasoning, both within and across ontologies, as Bayesian inferences. The query to the knowledge base is taken as onto1 and the main ontology as onto2. Based on the measure of similarity in mapping between both ontologies, the required information can be retrieved by the seeker. Three types of ontology mapping are supported in BN method: one-to-one, one-to-many or many-to-many concepts between onto1 and onto2.

3. The OSEE Model

The OSEE model is envisaged to obtain reusability of application and process domain knowledge using ontologies as the driving force. Hence, ontologies, project environment and query handling become important components of the model. This is proposed in five stages as shown in figure 1.

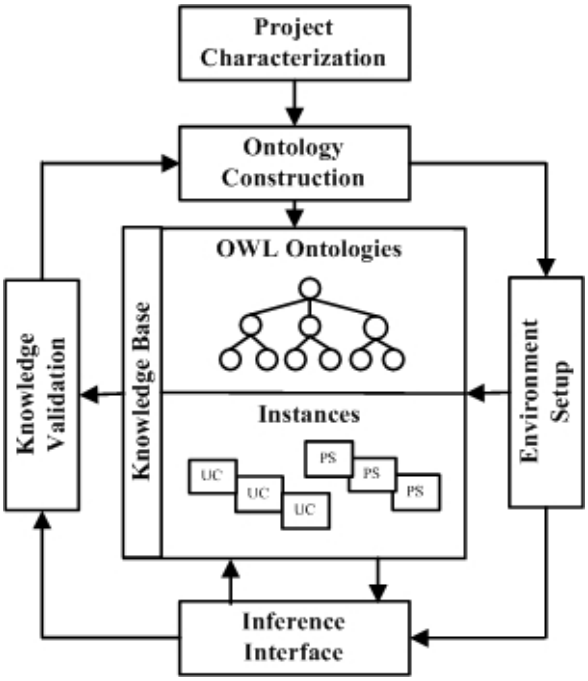


Fig. 1. OSEE Model

i. Project Characterization. As the scope of a project is defined and user requirements specification is agreed upon, the key concepts (T_C) and their relationships (T_R) from requirements specification and application domain knowledge are identified. The factors for process tailoring are decided based upon the outputs from project planning: effort estimation, deliverables and specific conditions for project execution. These characteristics provide inputs to select or create pertinent ontologies for the problem and process domains and to set up the project environment for knowledge-based software engineering.

ii. Ontology Construction. The knowledge base is built on a single ontology (known as OSEE) to which process and application domain ontologies are linked as sub-ontologies. Initially, the software engineering process of the organization is defined as the process ontology. For every project in a new domain, the domain ontology is created. There are already ontology libraries available for various domains as knowledge resources [21]. It is recommended to adapt an existing one from the available ontology collections. Ontologies for a specific domain can also be created using existing standards or process models of the system. For instance, the

ontology for Health Care can be defined using HL7-Reference Information Model [22], a standard for Health Care Information Systems. If the ontology for an application domain already exists, the concepts and relations of the new project are added to the ontology of the application domain.

iii. Environment Setup. Every software project is defined as a new OWL class inheriting from OSEE:Project class (see Figure 2). To the new class, the necessary concepts and relations from the application domain knowledge are linked, so that it describes the problem domain adequately. The tailoring of the organizational process based on the expectations for project management is also added to the class. Thus, the created class describes the target project in terms of the problem and process using OWL property restrictions. OWL syntax suits well to represent any knowledge structure formally. The specific user requirements of the problem domain are included as instances in the relevant classes of the application domain ontology. Thus, a formal knowledge environment for the project is set up so that the developers can access the right information or update based on their experience and insights.

iv. Inference Interface. This is the inference cum user interface of this model where a user can update tacit knowledge as instances or retrieve knowledge snippets from the repository. User can submit a query in the context of software engineering or the application domain. The query is converted into an ontology, which is mapped, with the ontology of the knowledge base, and the most matching cluster of concepts are presented based on probabilistic measure (p) using BN method.

v. Knowledge Validation. The quality of knowledge base depends on the accuracy and relevance of information retrieved from the knowledge base. The resultant of a query is obtained based on the nearest probabilistic measure applying the inference approach. In order to validate the knowledge base, we categorize three types of outputs {N, P, E} based on the probability value (p) associated with a query and compare it with the assumed similarity threshold value (τ).

i. N: Not Relevant (when $p=0$)
ii. P: Partial Result (when $0 < p \leq \tau$)
iii. E: Exact Result (when $p > \tau$)

Test cases are selected on different conditions such as input size, complexity, hierarchy of information and level of logical inference. To apply Chi-Square test, the results of test cases are obtained from domain experts as expected value $\{N_e, P_e, E_e\}$. Similarly, the results through ontology mapping are taken as observed values $\{N_o, P_o, E_o\}$. Using Chi-Square test for the selected sets of test cases, with degree of freedom as 2, the consistency and efficiency of the knowledge base is established.

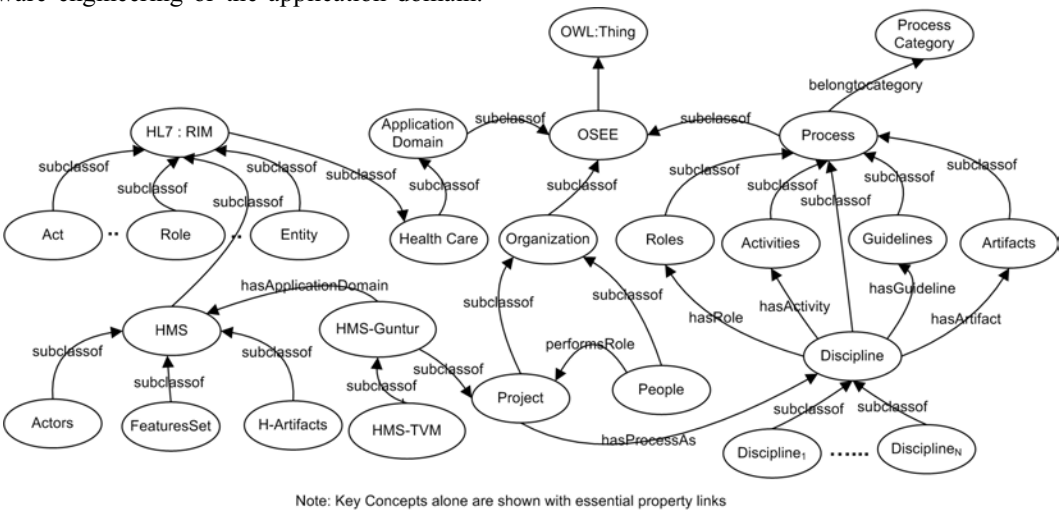


Fig. 2. Partial View of Ontologies in an OSEE implementation

4. Experiment

Hospital automation software (called as HMS), which is developed and deployed for three hospitals in South India by our Software Development Centre, was taken as pilot project for OSEE implementation. A partial view of concepts and relations in the developed OSEE ontology is shown in figure 2. Basing on HL7 Reference Information Model, ontology for the domain knowledge of Hospital-1 $\{Wd_1\}$ was done classifying the concepts and relationships from user requirements specification. This was built on three main concepts (or classes): feature-set, actors and artifacts. Where, actors signify the individuals who perform activities in the hospital; feature-set includes all features of the software application with its distinct activities such as Registration, Billing, Pharmacy, Nursing Station and so on and artifacts are the products of the information system (both input and output). The requirements $\{R_1\}$ (as use cases for each feature) were included as instances to the respective feature classes. While developing ontology for Hospital-2 $\{Wd_2\}$, it fitted exactly with $\{Wd_1\}$. The individual features $\{R_2\}$ were differing and they were added as new instances to the respective classes. Similarly, Hospital-3 was handled. Thus, {Wd+R} was built up based on the domain inputs from three hospitals where the knowledge of the domain is made explicit using ontological structures, and the unique activities of each hospital are stored as instances. It has proven to be a single point reference for domain knowledge, facilitating domain analysis for any system analyst even without prior exposure to the particular domain.

The software engineering process of the organization is a tailored version of Rational Unified Process. When the ontology was created for the organization's process, the descriptive model of the process fitted well into the ontologies. It paved way for a formal representation of the process. Analyzing the process implementation issues, which are stored as instances in the knowledge base, it enabled process improvement. The ontologies were created using Protégé [23] as the editor and knowledge items were stored as instances using a software prototype developed with Jena toolkit [24]. An integrated CASE tool is being developed to update and retrieve knowledge items based on concept matching. This tool will facilitate building up knowledge base with instances of tacit knowledge from various software projects. A larger repository will validate the efficacy of OSEE model. However, the preliminary results have established the proof-of-concept that ontology-driven software engineering knowledge base can enhance domain modeling and process improvement. And it can provide context-sensitive assistance to software developers based on the accumulated and structured knowledge.

5. Conclusion and Future Work

We have shown how a knowledge-base can be devised for software engineering environment where explicit knowledge of the application domain and software processes are represented as ontologies and tacit knowledge as instances. This model is effective if an ontology engineer in an organization drives every software project with ontological commitment. Our future works are focused on the development of CASE tool to incorporate all features of the model, which includes ontology mapping and testing the validity of the model with large volume of data. If rules (using SWRL) are incorporated into OWL classes and properties, it will facilitate better expressivity and automated reasoning of application domain and/or process knowledge. Integrating the knowledge base into Semantic Web will extend OSEE model catering to knowledge management of distributed software development activities at multiple locations. This concept can be easily extended to any branch of engineering where explicit and tacit knowledge need to be organized and reused.

Acknowledgement

This work was supported by University Grants Commission of the Government of India under Faculty Improvement Programme of Teacher Fellowship (Fellow Code: XTFTNMD094).

References

[1] Aybuke Aurum, Ross Jeffery, Claes Wohlin and Meliha Handzic (Eds.), *Managing Software Engineering Knowledge*, Springer-Verlag, 2003.

[2] Sergio Bandinelli, Luciano Baresi, Alfonso Fuggetta, and Luigi Lavazza, "Experiences in the Implementation of a Process-centered Software Engineering Environment using Object-Oriented Technology", *Theory and Practice of Object Systems*, vol. 24, no. 4, 1994, pp. 1-18.

[3] P. Mi and W. Scacchi, “A Meta-Model for Formulating Knowledge-Based Models of Software Development”, *Decision Support Systems*, vol. 17, no. 3, 1996, pp. 313-330.

[4] Rodrigo Reis, Carla Reis and Daltro José Nunes, “Automated Support for Software Process Reuse: Requirements and Early Experiences with the APSEE model”, *in Proceedings of the Seventh International Workshop on Groupware (CRIWG-01)*, Darmstadt (Germany), September-2001.

[5] Kacem Zeroual and Pierre N, “KBMS: A Knowledge-Based System for Modeling Software System Specifications”, *IEEE Transactions on Knowledge and Data Engineering*, vol. 4, no. 3, 1992, pp. 238-252.

[6] Victor Basili, G. Caldiera, and H.D. Rombach, “Experience Factory” in *Encyclopedia of Software Engineering*, vol. 1, Ed. John J. Marciniak, John Wiley & Sons, 1994, pp. 476-496.

[7] Scott Henninger, “A Tool Support for Experience-Based Software Development Methodologies”, in *Advances in Computers*, vol. 59, 2003, pp. 29-82.

[8] Carl A Gunter, Elsa L Gunter, Michael Jackson and Pamela Zave, “A Reference Model for Requirements and Specifications”, *IEEE Software,* vol. 17, no. 3, 2000, pp. 37-43.

[9] J.L. Diaz-Herrera, “Domain Engineering”, in *Handbook of Software Engineering & Knowledge Engineering, Vol. 1 Fundamentals*, Ed. S K Chang, World Scientific Publishing, 2001, pp. 311-312.

[10] Grady Booch, James Rumbaugh and Ivar Jacobson, *The Unified Modeling Language User Guide*, Addison- Wesley, 1999, pp. 219-221.

[11] Silvia T Acuna, Angelica de Antonio, Xavier Ferre, and Martia Lopez, “The Software Process: Modeling, Evaluation and Improvement”, in *Handbook of Software Engineering & Knowledge Engineering, Vol. 1 Fundamentals*, Ed. S K Chang, World Scientific Publishing, 2001, p. 200.

[12] A Aamodt and E Plaza, “Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches”, *Artificial Intelligence Communications*, vol. 7, no. 1, 1994, pp. 39-59.

[13] T.R. Gruber, “A Translation Approach to Portable Ontology Specifications”, *Knowledge Acquisition*, vol. 5, no. 2, 1993, pp. 199-220.

[14] I. Horrocks, P. F. Patel-Schneider and F. van Harmelen. “From SHIQ and RDF to OWL: The making of a Web Ontology Language”, *Journal of Web Semantics*, vol. 1, no. 1, 2003, pp. 7–26.

[15] Tim Berners-Lee, James Hendler and Ora Lassila, “The Semantic Web”, *Scientific American*, vol. 284, no. 5, 2001, pp. 35–42.

[16] http://www.w3.org/TR/owl-guide/

[17] Franz Baader and Werner Nutt, “Basic Description Logics” in *The Description Logic Handbook: Theory, Implementation and Applications*, Eds. Diego Calvanese, Deborah McGuinness, Daniele Nardi, Peter Patel-Schneider, Cambridge University Press, 2003, pp. 43-95.

[18] I. Horrocks, “OWL Rules: A Proposal and Prototype Implementation”, *Journal of Semantic Web*, vol. 3, no. 1, 2005, pp. 23-40.

[19] Philip H.P. Nguyen and Dan Corbett, “A Basic Mathematical Framework for Conceptual Graphs”, *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 2, 2006, pp. 261-271.

[20] Pan R., Ding Z., Yu Y. and Peng Y., "A Bayesian Network Approach to Ontology Mapping", *in Proceedings of the Fourth International Semantic Web Conference*, November 2005.

[21] http://www.daml.org/ontologies/keyword.html.

[22] http://www.hl7.org/Library/

[23] Holger Knublauch, “Protégé-OWL and the Semantic Web”, *8th International Protégé Conference, Madrid*, July 2005.

[24] Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., and Wilkinson, K., “Jena: Implementing the Semantic Web Recommendations”, in *Proc. of the Thirteenth International World Wide Web Conference*, May-2004.

Performing Requirements Elicitation Activities Supported by Quality Ontologies

Taiseera Hazeem Al Balushi, Pedro R. Falcone Sampaio, Divyesh Dabhi, Pericles Loucopoulos
School of Informatics, University of Manchester, PO Box 88, Manchester M60 1QD, UK
Taiseera.Al-balushi@postgrad.manchester.ac.uk
[P.Sampaio, P.Loucopoulos]@manchester.ac.uk
D.Dabhi@student.manchester.ac.uk

Abstract

The requirements elicitation phase is often regarded as the most critical stage of the entire software engineering effort with strong evidence suggesting that increasing the effectiveness of requirements analysts and reducing requirements elicitation errors may be the key to improve project outcomes and deliver high quality software. This paper presents a requirements elicitation approach and associated tool aimed at empowering requirements analysts with a knowledge repository that helps in the process of capturing precise non-functional requirements specifications during elicitation interviews. The approach is based on the application of functional and non-functional domain ontologies (quality ontologies) to underpin the elicitation activities. We also discuss how the approach and tool are being used to effectively support the requirements elicitation stage of the new student intranet project of the University of Manchester (Manchester Unity Web Project).

Keywords: non-functional requirements, requirements engineering, requirements elicitation, ontologies.

1. Introduction

Requirements elicitation is the first step in the requirements engineering process and is often considered to be one of the most critical activities of software development [1]. Requirements elicitation presents several challenges to developers. According to [2] requirements elicitation challenges can be classified into three main themes: (1) problems of scope, in which the requirements may address too little or too much information; (2) problems of understanding, where communication and semantic interpretation of requirements may differ within groups as well as between groups such as users and developers; and (3) problems of volatility, i.e., the changing nature of requirements.

Problems of scope typically result from the fact that requirements analysts often have limited knowledge of the application domain which in turn limits their effectiveness in asking the right questions towards capturing a complete set of requirements during requirements elicitation interviews. Problems of understanding typically result from the fact that requirements elicitation involves many people's expressed needs across different communities as apposed to the result of a single person's perspective. These multiple perspectives present challenges with regard to redundancy of information, inconsistency, and point of view alignment [3]. Problems of understanding also arise in connection with non-functional requirements (NFR) or quality requirements elicitation. NFRs are viewed as system properties and constraints e.g. reliability, response time, storage requirements, safety etc. Some of the key difficulties relating to capturing a sound, complete and agreed upon set of quality requirements relate to:

- Quality requirements are of abstract and subjective nature with stakeholders having different perceptions of quality factors across different applications. Thus there is need for methods, techniques and tools to foster a shared collective understanding of the domain.
- A large scale project may involve hundreds of quality requirements, therefore, there is a need for a knowledge base of quality requirements linked to different application domains to remind requirements analysts of the breadth of NFRs that may be critical to a specific domain as well as the potential trade-offs that may be caused by conflicting quality requirements.

This paper presents a requirements elicitation approach and associated tool aimed at empowering requirements analysts with a knowledge repository that helps in the process of asking the right questions and capturing precise non-functional requirements specification during elicitation interviews. The approach is based on the application of functional and non-functional domain ontologies to underpin the elicitation activities. In particular, the non-functional ontology (quality ontology) helps to ensure all relevant quality requirements related to a domain are addressed during interviews (thus, reducing problems scope), and also helps in the process of creating standardized quality criteria that can be uniformly used by different analysts during interviews (thus, reducing problems of understanding).

For the purpose of this study we illustrate the effectiveness of the approach/tool in supporting the requirements elicitation stage of the university helpdesk functionality relating to a new student intranet development project of the University of Manchester (Manchester Unity Web Project).

The remainder of this paper is divided as follows: Section 2 discusses the benefits of using quality ontologies in requirements elicitation. Section 3 presents the related work in addressing quality factors in requirements elicitation. Section 4 describes the development of a quality ontology guided requirements elicitation approach. Section 5 discusses the process of requirement elicitation supported by the tool. Section 6, provides an elicitation process/tools evaluation. Section 7, summarizes the paper, discusses the key contributions, and the future work.

2. Benefits of Using Quality Ontologies in Connection with Requirements Elicitation

Ontologies, according to [4] is defined as "a formal, explicit specification of a shared conceptualization". Ontologies can be used to capture quality dimensions of a domain of discourse. For example, the TOVE Quality Ontology-VB is the representational basis upon which formalised quality knowledge can be used to integrate quality-related decision making throughout an enterprise [5]. More recently, the Qurator project [6] uses ontologies to describe the quality of curated e-Science information resources assisting scientists and data curators in managing the quality of information. They provide scientists with the means of annotating information with explicit descriptions of its quality in terms that are relevant to the domain of interest and the specific application in hand. The ontologies were built around medical domains to support quality knowledge sharing of lab results between scientists.

Quality ontologies are also used in specifying quality of service (QoS) requirements of service-centric systems [7]. The approach described in [7] is based on defining quality characteristics that will be used in the process of matching functional and non-functional requirements during service sourcing and composition.

With regard to supporting requirements elicitation, ontologies provide the following benefits:

- Ontologies promote a shared domain vocabulary that can be used to avoid ambiguities arising in projects involving teams of multiple requirements analysts and stakeholders. This shared vocabulary can also help to develop a common language that can be applied in several projects [8], across several teams to foster requirements reuse and uniform understanding across requirements elicitation activities.
- Ontologies are often used to encode specialized knowledge that is used to formulate competency questions [9]. In particular, questions with regard to the quality requirements relevant to a particular domain, facilitating the task of asking relevant questions to elicit a complete set of quality requirements during stakeholders' interviews.

Quality ontologies, therefore, can have a positive impact on the requirements engineering phase when applied as a knowledge base during elicitation activities to focus on key quality characteristics of a domain.

3. Addressing Quality Factors in Requirements Elicitation: Related Work

In order to successfully integrate quality concepts and practices into the requirements engineering phase, satisfying the quality requirements of a software system, quality should be formally identified and addressed at very early stages of system development. There are two potentially complementary approaches for incorporating quality concerns into requirements elicitation activities:

Product-oriented: in this approach, quality criteria are made specific using quality models (e.g., ISO/IEC 9126[10]), NFR catalogues or patterns to enable precise descriptions of the characteristics and metrics relevant to an application domain. This is approach is concerned with modelling NFRs and measuring the extent to which a software system is in accordance with the set of NFR's.

Process-oriented: in this approach, quality is regarded as goals to be achieved throughout the software design process. Methods such as softgoal analysis and design trade-offs are employed to discuss and reason about system quality characteristics.

One of the pioneer process-oriented approaches in capturing quality requirements during requirements elicitation is the NFR-goal framework [11] for NFRs elicitation and negotiation activities of the RE process, which views NFRs as goals that might conflict with each other during the elicitation process. The process-oriented approach discussed in [12] considers quality characteristics to be goals that need to be elicited from various stakeholders to achieve a certain level of service quality. A recent product-oriented approach by [13] emphasizes on the emerging field of reusability in requirements engineering. The work develops a knowledge base catalogue using i* framework to express aspects of usability that can be considered during requirements engineering.

4. Developing a Quality Ontology Guided Requirements Elicitation Approach

Despite the support in dealing with non-functional requirements during the elicitation phase, all approaches discussed above, however, have not adopted a standard

quality model (e.g., ISO/IEC 9126) to integrate the non-functional requirements with the functional requirements, covering a wide range of quality characteristics important for several application domains. Moreover, to the best of our knowledge, no requirements elicitation method has investigated the effectiveness of using automated knowledge bases such as ontology management systems to support the elicitation stage. Thus, our motivation is to develop an ontology driven requirements elicitation method, guided by a standard quality model. The quality model is encoded as quality ontology, and automated by a requirements elicitation tool, helping to address quality factors during elicitation interviews. Figure 1 illustrates the proposed approach.

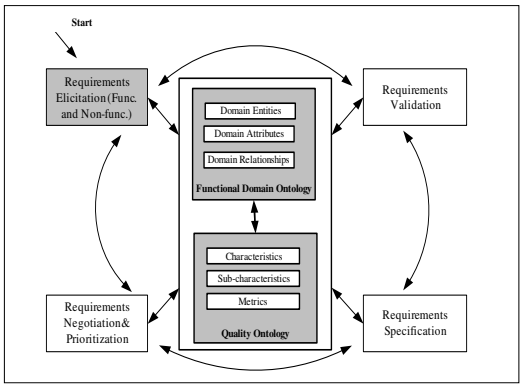


Figure 1. Ontology Guided Requirements Elicitation

There are two types of ontologies needed to guide the process of capturing and addressing quality concerns in requirements engineering. These ontologies are:

- *Quality ontology*, which is based on software quality models, and it represents reusable knowledge about different quality characteristics, sub-characteristics, and metrics. For example the quality characteristic reliability embraces the sub-characteristics maturity that can be measured by the level of accessibility anytime anywhere, fault tolerance measured by mean time between failure, and recoverability which can be measured by the mean time to repair. These quality factors are general and can be applied to any application domain, however, the level of quality required and the order of importance of these quality factors may vary from a domain to another and will be further detailed during elicitation interviews. Section 4.1 describes the quality ontology in more details.
- *Functional domain ontology*, which provides a conceptual structure of the domain (e.g. university helpdesk, in this paper) that information is collected and processed about. The domain entities are acquired (students, centers, services, labs, faculty, etc.) and properties or attributes attached to each entity (student-ID, name, address, etc.) and the relationship between these entities.

The cornerstone element of the proposed approach described in this paper (illustrated in Figure 1) is the use of a quality ontology representing quality characteristics found in a standard quality model (the ISO/IEC 9126 quality model) addressing quality as first class requirement in the requirements engineering process. The quality ontology explicitly implements the features of a formal quality model that is applied throughout the requirements engineering effort (in our case the requirements elicitation stage). Non-functional requirements are represented as quality requirements and quality metrics are explicitly specified and associated with the functional domain ontology. The second pivotal element is the process of using the quality ontology to guide requirements elicitation activities and the supporting tool to help the requirements analysts in the interviewing process.

The proposed approach is a combination of product-oriented and process-oriented quality driven requirements elicitation approaches supported by a quality knowledge base (ontology). The quality knowledge base constructed helps in the process of capturing and validating requirements. The quality ontology is used to drive the process, and act as selection criteria. The elicitation tool offers functions and corresponding interfaces to assist in the requirements elicitation steps.

4.1. Adopting and Implementing a Baseline Quality Model Ontology

To develop a quality ontology to support requirements elicitation we chose to adopt the ISO/IEC 9126 software product quality model as the baseline quality model for developing the foundations for integrating quality engineering factors onto requirements engineering activities. To automate the ISO/IEC 9126 quality model, quality characteristics and sub-characteristics identified in the model are translated into a set of hierarchical classes in the Protégé ontology knowledge management environment as shown in Figure 2. The left-hand side of the figure shows how the quality characteristics, sub-characteristics and metrics are mapped into a hierarchy. The right-hand side of the figure defines each class, relationships, and asserted conditions.

Once the baseline quality ontology is specified Figure 2 (A) the next challenge is to associate the general quality ontology with the functional application domain ontology, as illustrated in Figure 2 (B) (a university helpdesk ontology for the example described in this paper). For example, how useable is the FAQ under the helpdesk services. This is achieved through OWL expressions denoting domain restrictions/constraints [14, 15] these

expressions allow cross-ontology assertions enabling the association of quality model characteristics with functional domain characteristics, as illustrated in Figure 2 (C). The advantage of using an ontology management system such as Protégé is that due to the ontology interoperability features supported, other functional domain ontologies can be rapidly imported and associated with the quality ontology; this can help with customizing the use of quality factors and their applications across a multitude of application domains.

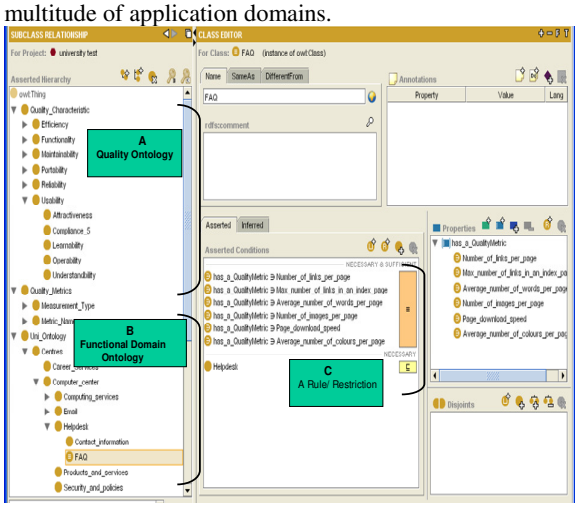


Figure 2. Quality Ontology Implemented in Protégé

5. Performing Requirements Elicitation Activities: Process and Tool Usage

Pre-requisite: Implementing the ontology – the ontology development process begins with the identification and structuring of the domain knowledge, describing the relevant functional and non-functional characteristics. In our case we chose to study a university helpdesk. To structure the university domain knowledge and its quality characteristics we used textbooks, standards, and interviews with domain experts (e.g. head of information services and five help desk operators with more than 5 years of experience each). The functional domain ontology was then mapped into Protégé as discussed in section 4. In addition to the university functional domain analysis, the quality ontology was developed by including all quality characteristics pertaining to the ISO/IEC 9126 quality model. Finally, the quality model was mapped into a Protégé hierarchy as discussed in section 4. Once all the necessary quality requirements are captured and stored in the protégé knowledge base, the final step is to develop a tool for supporting requirements analysts in the requirements elicitation phase. The process of requirements elicitation using the ontologies illustrated in Figure 3 using OMG's Software Process Engineering Metamodel (SPEM)[16], starts when the requirements analyst performs interviews with the stakeholders towards capturing detailed functional/quality requirements relevant to the application domain.

The requirements elicitation tool – The quality ontology-guided elicitation tool supports the process by retrieving from the ontology knowledge base relevant quality characteristics attached to the domain that will underpin potential questions that the requirements analyst will need to ask when eliciting quality aspects relevant to a particular functional characteristic of the domain of discourse. Once all the requirements are identified, a list of detailed quality requirements is generated.

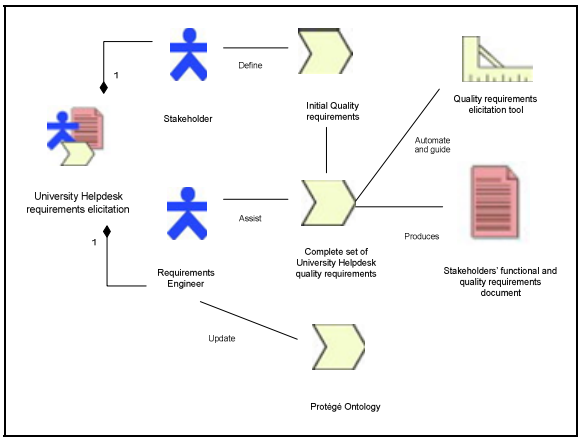


Figure 3. University Quality Requirements Elicitation

The main benefit of the elicitation tool is to help the requirements analysts in the process of requirements elicitation disregarding his/her level of expertise about the functional and non-functional requirements of a given domain. Having a list of functional and quality requirements for a specific application domain helps to decrease the occurrence of problems of scope (e.g., finding all relevant requirements) and understanding (e.g., enabling that all non-functional requirements are uniformly treated across different elicitation interviews conducted by different requirements analysts), thus, reducing the chances of missing out important requirements or not treating requirements uniformly.

In addition, the explicitly defined quality characteristics and metrics associated to that domain will help during requirements negotiation and prioritization since precise and non-ambiguous specifications are developed. Figure 4 illustrates the tool front-end interface.

The requirements elicitation tool user interface is built using the Protégé API, which interacts with the objects stored in the knowledge base (functional and non-functional domain ontologies)

As illustrated in Figure 4, the tool supports the task of eliciting requirements and also facilitates the communication process between the stakeholders and the

requirements analysts. The top of the screenshot is the functional domain ontology representation, where the stakeholders can choose what part of helpdesk activity they need to specify with regard to quality requirements.

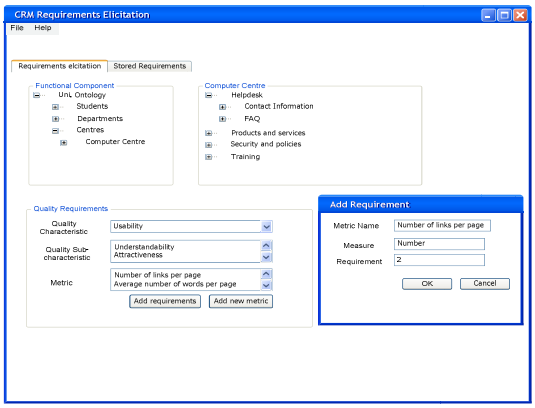


Figure 4. Requirements Elicitation Tool GUI

Once a certain activity is selected (e.g. FAQ) relevant quality characteristics that can be discussed with stakeholders towards developing NFR specifications are presented. The tool displays a list of quality characteristics, sub-characteristics, and metrics that will underpin the formulation of a precise requirement statement. The add requirements button allows the stakeholders to detail a quality requirement, in the given example, the tool provides the requirements analyst with the quality characteristics (usability and efficiency) and their associated sub-characteristics (understandability, attractiveness and time behavior) related to the functional activity FAQ. The tool also allows the requirements analyst to ask more specific questions about their quality requirements through metrics such as (number of links per page, number of words per page, and etc.). By specifying the metrics, the requirements can be measured and controlled. The tool also allows stakeholders to specify additional requirements that are not listed by the tool via add new metric button which can always be added and updated as a new class or knowledge in the quality ontology, thus helping to increase the knowledge about a particular domain.

6. Elicitation Process/ tool Evaluation

To evaluate the proposed elicitation tool, the authors attended a focus group session which was one of the ongoing sessions at the University of Manchester for the purpose of enhancing the current helpdesk website of the university. The participants were asked for what they want to see in the website and what the problems they come across using the website. It was a two hours interview session and the requirements elicited are listed in Table 1.

Table 1. Requirements Captured without the Tool Support

	User Requirements
R1	Provide information/pathway onto how to access web services (i.e. web mail, network drive, etc.)
R2	FAQ should be clear and simple in answering users technical problems
R3	Make the websites among different schools consistent
R4	Provide campus map when required
R5	Make the university regulations and policies easy to access
R6	Make students user names accessible to faculty when using WebCT (e-learning) to register students
R7	Provide information on how to report a problem and to whom
R8	Provide information about exam timetables and venues
R9	Provide links to the outside world
R10	Highlight important events or alerts
R11	Update the staff directory frequently

As it is illustrated in the table above, the amount of requirements collected were very limited and some of them were very general and not clear enough such as (provide links to outside world). In addition these requirements can be of different categories such as (email, regulations, computer services, presentation of the website, and security of the website) which are a mix of functional and non-functional requirements.

The main objective in the evaluation was to assess the effectiveness of the tool in complementing the outcome of the focus group towards improving the quality of the requirements captured.

Table 2 presents the set of requirements obtained after using the ontology based requirements elicitation tool after interviewing two of the participants (Intranet project manager and the IT services manager) from the focus group for two hours (same amount of time used during the focus group). It is evident from Table 2 that the ontology based tool has complemented the requirements captured during the focus group by helping to enhance the level of precision and the scope of requirements captured. This is due to the fact that the tool leverages the knowledge repository of functional and non-functional requirements relevant to the domain of discourse. The requirements analyst doesn't have to be an expert in the domain as all needed expertise is stored in the ontology. The knowledge encoded in the ontology has a positive impact in reducing the problem of scope (helping requirements analysts to focus on the relevant aspects of the domain) and reducing the chances of missing out important aspects of quality requirements. The tool also helps to promote effective communications as the quality/functional requirements are better communicated with the stakeholders as they are defined and broken down into a set of measurable metrics. To illustrate this feature, consider for example one of the requirements elicited using the traditional approach (e.g. R2). The precision of the same requirement captured using the ontology based tool is highlighted in Table 2, in which this requirement is considered as one of the helpdesk services.

Table 2. Requirements Derived from Ontology Based Elicitation Tool

Functional department services	User requirements	Quality Characteristic	Quality sub-characteristic	Metrics
Computer centre: Helpdesk activities	FAQ	Usability	Understandability	• Number of links per page (2 links) • Maximum number of links in an index page (20 links per page) • Average number of words per page (200 words) • Average depth of pages (distance between the homepage to a page with no further links (5 pages) • Number of External links (3 links)
			Attractiveness	• Interactive objects (search buttons, text box, pulldown menu) per page (all options) • Number of images per page (3 images) • Average number of colours per page (3 colours)
		Efficiency	Time behaviour	• Page download speed (1 second)
Web services	Web mail	Reliability	Fault tolerance	• Mean Time to Failure (604800 seconds, i.e. 1 week) • Mean Time to Recover (3600 seconds, i.e. 1 hour)
		Efficiency	Time behaviour	• Response time (2 second)
		Functionality	Security	• Number of failed authentication attempts (2) • Automatic virus check (twice a day)

7. Conclusions and Future work

This paper proposes a requirements elicitation approach and associated tool aimed at supporting requirements analysts with a knowledge repository that helps in the process of capturing a comprehensive and precise set of quality requirements during elicitation interviews. The approach is based on the application of functional and non-functional domain ontologies (quality ontologies) to underpin the elicitation activities. The ontology guided requirements elicitation process has key substantial elements: (1) the quality ontology that is based on the ISO/IEC 9126 standard; (2) a functional domain ontology which describes all related objects and operations attached to a certain domain. The ontology-guided elicitation tool was constructed by developing a requirements elicitation application layer on top of Protégé's application programming interface. We also reported on the application of the elicitation process/tool to support requirements elicitation activities of a student helpdesk project of the University of Manchester, where it complements the focus group session by helping the requirements analyst to develop detailed quality specifications with a superior level of precision. Future work will be made at an experimental level, as this tool will be tested by subject groups to check for its usability and functionality. We will also investigate how the ontology can help with requirements prioritization and negotiation.

8. References

[1] F. Brooks, "No sliver bullet-Essence and accidents of software engineering," in *Computer*, vol. 20, 1987, pp. 10-19.

[2] M. G. Christel and K. C. Kang, "Issues in Requirements Elicitation. Technical Report CMU/SEI-92-TR-12," Software Engineering Institute, Carnegie Mellon University, Pennsylvania 1992.

[3] D. W. Cordes and D. L. Carver, "Evaluation Method for User Requirements Documents," *Information and Software Technology*, vol. 31, pp. 181-188, 1989.

[4] T. R. Gurber, "Toward Principles for the Design of Ontologies Used for Knowledge Sharing," in *Formal Ontology in Conceptual Analysis and Knowledge Representation*. Deventer, The Netherlands: Kluwer Academic Publishers, 1993.

[5] H. M. Kim, M. S. Fox, and M. Gruninger, "An ontology of quality for enterprise modelling," presented at Proceedings of the Fourth Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 1995, 1995.

[6] Qurator, "http://www.qurator.org," 2005.

[7] G. Dobson, R. Lock, and I. Sommerville, "Quality of Service Requirements Specification using an Ontology," presented at Proc. Workshop on Service-Oriented Computing Requirements (SOCCER), Paris, 2005.

[8] T. Lau and Y. Sure, "Introducing Ontology-based Skills Management at a large Insurance Company," *In Modellierung 2002, Modellierung in der Praxis - Modellierung für die Praxis, Tutzing, Deutschland, 25.-27*, pp. 123-134, 2002.

[9] S. Staab, R. Studer, H.-P. Schnurr, and Y. Sure, "Knowledge processes and ontologies," *IEEE Intelligent Systems*, vol. 16, pp. 26-34, 2001.

[10] "ISO/IEC 9126-1:2001 Software engineering -- Product quality -- Part 1: Quality model.."

[11] J. Mylopoulos, L. Chung, and B. Nixon, "Representing and Using Non-Functional Requirements: A Process Oriented Approach," *IEEE Transactions on Software Engineering*, vol. 18, pp. 483-497, 1992.

[12] P. Loucopoulos, K. Zografos, and N. Prekas, "Requirements Elicitation for the Design of Venue Operations for the Athens2004 Olympic Games," presented at Proceedings of 11th IEEE International Requirements Engineering Conference, Monterey Bay, California, U.S.A., 2003.

[13] L. M. Cysneiros, V. M. Werneck, and A. Kushniruk, "Reusable Knowledge for Satisficing Usability Requirements," presented at Proceedings of the13th IEEE International Conference on Requirements Engineering, 2005., 2005.

[14] "World Wide Web Consortium," OWL Web Ontology Language References. W3C Recommendation 10 Feb, 2004.

[15] H. Knublauch, R. W. Fergerson, N. F. Noy, and M. A. Musen, "The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications," presented at Third International Semantic Web Conference, Hiroshima, Japan, 2004.

[16] "Object Management Group. Software Process Engineering Metamodel Specification. Version 1.1. January 2005, Technical Report 05-01-06,OMG."

KOntoR: An Ontology-enabled Approach to Software Reuse

Hans-Jörg Happel*), Axel Korthaus†), Stefan Seedorf†) and Peter Tomczyk*)

*) *FZI Research Center for Information Technologies*
Research Group Information Process Engineering (IPE)
Haid-und-Neu-Str. 10-14
D-76137 Karlsruhe, Germany
{happel|tomczyk}@fzi.de

†) *University of Mannheim*
Lehrstuhl für Wirtschaftsinformatik III
Schloss, L 5,5
D-68131 Mannheim, Germany
{korthaus|seedorf}@wifo.uni-mannheim.de

Abstract

Research on software reuse libraries has extensively dealt with representation and retrieval issues of software artifacts. While representation in terms of metadata is a key issue, most systems neglect the possibilities of leveraging knowledge about the corresponding problem domain. In this paper, we present KOntoR—an ontology-enabled approach to software reuse. We show how background knowledge provided in the form of ontologies can increase the value of reuse libraries. This is achieved by integrating explicit and implicit metadata semantically, thus providing means for deriving new facts. Additionally, we give three examples that show how software library users can benefit from formalized knowledge, e.g. about software licenses or programming technologies.

1. Introduction

Although the topic of reuse has been widely discussed in the area of software engineering for many years, many researchers and practitioners are not yet satisfied with the current state of practice [1]. Component libraries, service registries or artifact repositories constitute an indispensable part of software reuse systems. They all have in common that they manage artifact descriptions for supporting the process of retrieval, adaptation and integration. However, current approaches often fall short of providing adequate support for typical reuse tasks. The paper thus particularly addresses the following problems:

- In many cases, the relevant information resides isolated in multiple heterogeneous descriptions of an artifact, covering different aspects each. It should therefore be examined how these descriptions can be integrated.
- A purely metadata-based approach is often not powerful enough to answer declarative queries (see Q.1 - Q.3) and thus requires additional background knowledge. To date, most reuse systems do not incorporate formalized knowledge about application domains or artifacts.

We argue that semantic web technologies provide the means for addressing these issues. In this paper, we present *KOntoR*, an infrastructure for software reuse employing technologies from the emerging field of semantic web research. We show that codifying knowledge in an ontology-enabled infrastructure can significantly improve the value of a reuse environment.

A typical usage scenario of a reuse environment would include the retrieval of particular software components fitting a specific application development need. However, realistic reuse scenarios require a wider range of supported functionality. The multifaceted requirements on such enhanced software reuse systems can be illustrated in the form of competency questions (cf. [2]).

These are example queries that should be supported by our ontology-enabled approach, such as:

Q.1 Find all artifacts dealing with the *customer* business object.
Q.2 Tell me if componentA and componentB can be used in my product under a proprietary license.
Q.3 Find a developer who is experienced in building banking applications in Java.

The paper is structured as follows: We first analyze the shortcomings of current reuse approaches. In chapter 3, the architecture of the KOntoR approach is derived. Chapter 4 describes the prototype implementation and gives an example for the realization of Q.1 – Q.3, before summarizing the key advantages of our approach.

2. Related work

Approaches to software reuse target different kinds of reuse artifacts—ranging from classical binary, *off-the-shelf* components to source code fragments, process knowledge and *experience* [3, 4, 5]. However, to manage and retrieve these artifacts, an appropriate infrastructure has to be in place. In this chapter we will therefore review metadata-oriented and intelligent reuse systems.

2.1 Metadata-based reuse

Unlike text documents, most reuse artifacts in software engineering are not human-readable. This applies especially to binary code, but also the serializations of design models or test cases are not necessarily understandable in their raw form [6]. Also, access might be unfeasible due to the artifact's complexity. Thus, some representation of an artifact is needed that can be matched against a user's request [7, 8]. Since this representation is describing the actual reuse artifact's data, we call it metadata. A general setting of software reuse would therefore involve three essential success factors as depicted in Figure 1.

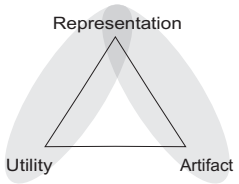


Figure 1: Dimensions of reuse (derived from [6])

For the user to maximize utility a library containing suitable artifacts and an appropriate representation in the form of queryable metadata describing those artifacts is required [6]. Traditional libraries for reuse artifacts serve as an intermediary between component providers and component requesters. Accordingly, they have two clearly defined interfaces to the software development process—providers use the repository once when adding new components and buyers access the repository to search for components. As a result, component libraries focus on effective retrieval mechanisms and are based on so-called descriptive metadata. For example, components can be classified by using a fixed set of facets [9] or fixed taxonomies, both assuming a stable body of knowledge. In this sense, software is regarded as a kind of document—like in *library science*—that can be handled by classification and indexing methods. Thus, most reuse libraries allow for different kinds of artifacts, but are mostly limited to a fixed number of representation formats and users [6, 10].

In contrast to component libraries, collaborative development platforms aim at supporting distributed development teams. They offer different features, e.g. bug-tracking, communication tools, project management or version control. Although some platforms include basic project and artifact descriptions, they deal with administrative metadata in the first place. While primarily designed for supporting software development, sites like *SourceForge*[1] have become popular places to search for reusable components. However, metadata and retrieval mechanisms are less powerful than in component libraries. SourceForge for example is limited to keyword based search and browsing a simple classification taxonomy (TROVE-scheme[2]).

While metadata-based reuse has shown some success—especially in intra-organizational settings [11] and for infrastructure tools [3]—many authors claim that it has failed a breakthrough [1, 3, 11]. Especially in distributed development settings spanning multiple sites or time zones, a common conceptualization of development tasks and some background knowledge is required [12], which the presented tools do not provide.

2.2 Intelligent reuse systems

The rising number of artifacts, metadata formats and actors in modern software engineering processes imposes additional requirements on the design of reuse systems. Vitharana et al. [10] tackle this by differentiating among several user groups and representations of

[1] http://sourceforge.org
[2] http://sourceforge.net/softwaremap/index.php

structured and semi-structured data, as well as providing basic support for inferences. However, additional knowledge about the semi-structured data or its querying possibilities is not provided. Intelligent reuse systems try to remedy this by formally capturing context of an artifact or the user. Together with background knowledge, more powerful reuse results can be obtained. A knowledge-based reuse infrastructure providing personalized views for different stakeholders with different needs can add value to all three dimensions depicted in Figure 1.

Within the *utility* dimension, the structuring and capturing of user context can inform the reuse system. Recommender systems like Hipikat [13] or CodeBroker [14] gain information about the context of a developer or his task and recommend information that seems to be suitable in this context. However, these tools are integrated in a development environment (Eclipse resp. Emacs) and they are primarily designed to work in an intra-project setting. Additionally, the knowledge used for recommendation is not declaratively stored, but hard-coded.

Development information systems aim to enhance the *representation* of artifacts. Tools like the LaSSIE system [15] use description logic formalisms to store knowledge about the application domain and code structure into a knowledge base [16]. Thus, the system is able to answer declarative requests about the code using links between the domain model and the source code. Falbo et al. [17] work towards an ontology-based software development environment (ODE). Their vision includes the entire software development lifecycle by capturing process, tool and quality in ontologies.

Background knowledge can also be leveraged in the *artifact* dimension. Oberle [18] proposes an ontology-enabled application server for managing distributed systems. Here, semantic technologies are also used to describe relationships between artifacts in an application server, thus enabling use-cases such as the automatic discovery and loading of dependent libraries at run-time. While this system focuses on run-time application management instead of reuse, it clearly demonstrates the benefit of combining ontologies with existing metadata. Thus, the approach proposed by Oberle addresses both the representation dimension and the artifact dimension of the reuse triangle in Figure 1.

We think that combining "intelligent" approaches with the features of metadata-based reuse systems may lead to a powerful solution. Such an integrated approach is proposed and described in detail below.

3. The KOntoR solution approach

The identified shortcomings of existing software reuse systems, namely the low integration of artifact metadata and insufficient utilization of background knowledge, are addressed by two key elements in our architecture (see Figure 2):

1. An XML-based *metadata repository component* to describe software artifacts independently from a particular format
2. A *knowledge component* which comprises an ontology infrastructure and reasoning to leverage background knowledge about the artifacts

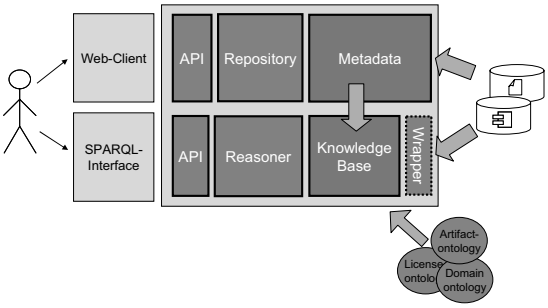


Figure 2: KOntoR high level architecture

In order to allow for the management of arbitrary metadata, the repository component is based on a particular metaschema (see Figure 3). Every metadata set is an XML document which describes a software artifact in one or more information aspects. For example, a WSDL[3] document describes how to communicate with a component's service and covers the information aspect *Interface*. The Dublin Core standard[4] may be used to specify the aspects *Authorship* and *Licensing* of a component.

Furthermore, different formats may be applied to represent an information aspect of a software artifact. This feature is also covered by the proposed metaschema. For example, the interface of a software component may be described using WSDL, Corba IDL or a UML profile serialized in XMI. For being able to fulfill the requirements stated above, the artifact metadata needs to be automatically homogenized, e.g. by transforming a WSDL definition into a simplified, consistent interface description, thereby unifying different formats.

[3] http://www.w3.org/2002/ws/desc/
[4] http://dublincore.org/

The artifact metadata is managed using a dedicated API. Artifact retrieval is supported by a SQL-style XML-based query language (cf. [19]). It can be used for creating structured queries (via XPath expressions[5]) as well as for keyword-based search.

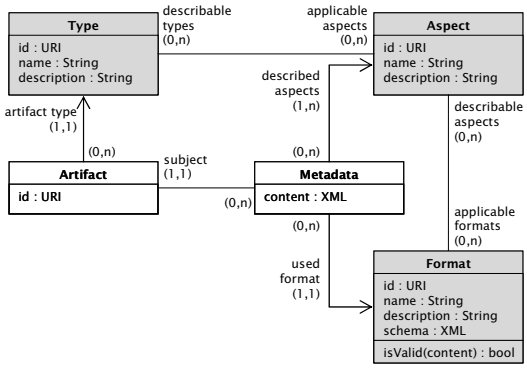


Figure 3: Metaschema

The second building block of the KOntoR architecture, the knowledge component, comprises a knowledge base which imports the transformed metadata. This is achieved by writing a mapping ontology and an export plug-in for each data source. In this way, assertional knowledge can be combined with background knowledge provided by ontologies. In the center, there is a "shallow" ontology for describing structural aspects of software artifacts (see Figure 4). This ontology is extended by optional domain ontologies, e.g. an ontology for describing standard software licenses. The background knowledge contained within the domain ontologies can be utilized to assist the user in a wide range of reuse problems. In the process of component selection, a system integrator might want to know whether an open source component (e.g. derived from GPL) can be reused in a commercial project under a proprietary license. The proposed architecture supports such queries since the necessary background knowledge for testing the compatibility of free and proprietary licenses is provided by a domain ontology which can be plugged in easily.

Semantic queries on top of the knowledge base are supported by a different API. The functionality of this interface concentrates on processing SPARQL[6] queries executed by an interpreter and returning the results. Finally, simple clients for both APIs are required to provide end users access to the system.

[5] http://www.w3.org/TR/xpath
[6] http://www.w3.org/TR/rdf-sparql-query/

4. Implementation and example

The architecture presented in the preceding section has been implemented in the context of the CollaBaWue[7] project. We will first describe selected implementation aspects and then give a concrete example to demonstrate how the identified use cases are realized.

4.1 Prototype implementation

The two main components of the KOntoR architecture have been realized as a prototype in Java. The repository component provides an API for managing artifact metadata and an instantiation of the metaschema. The metadata is stored in a Hypersonic relational database (hsqldb)[8] and indexed with Lucene, by using individual analyzers (e.g. one for XMI). Moreover, a web interface makes it possible to manage artifact descriptions.

For the prototype implementation of the knowledge component, we set up an RDFS/OWL[9] based infrastructure. The KAON2 API[10] was used for both storing the ontologies and reasoning. Declarative requests are supported using a simple front-end for KAON2's SPARQL interface.

In order to explain the example in the subsequent section, we need to derive an instantiation of the metaschema and specify ontologies for providing background knowledge. Table 1 shows the excerpt of a metaschema instance which can be extended to describe various aspects of software components. The metadata is specified by using standardized (e.g. WSDL, Dublin Core) as well as custom XML-based formats.

Type	Component
Information aspects	Interface, Behavior, License, Authorship
Formats	WSDL, XMI, DublinCore

Table 1: Instantiation of the metaschema

For our example presented below, we define an artifact ontology and additional domain ontologies, i.e. a banking, technology and software license ontology, which are part of the prototype's knowledge component.

[7] http://www.collabawue.de
[8] http://hsqldb.org/
[9] http://www.w3.org/2001/sw/
[10] http://kaon2.semanticweb.org/

4.2 Example

In order to demonstrate the applicability of our approach, we introduce the example of a banking component (*AccountBalance*) which returns the balance of a private customer's account. The component is realized as Web service and comprises a WSDL description of the component interface. In the excerpt below, a domain reference of the input element to the *PrivateCustomer* business object is established following an annotation approach comparable to the WSDL-S proposal (cf. [20]):

```
<wsdl: description ...>
<wsdl:types>
 <xs:schema ...>
  <xs:element name="c" type="Customer"
   bo:businessObjectRef="fin#PrivateCustomer">
        ...
  </xs:element>
  ...
 </xs:schema>
</wsdl:types>
<wsdl:interface name="GetAccountBalance">
 <wsdl:operation name="getBalance"
  pattern="http://www.w3.org/.../wsdl/in-out">
    <wsdl:input element="my:Customer"/>
    <wsdl:output element="my:Balance" />
 </wsdl:operation>
</wsdl:interface>
</wsdl:description>
```

Further, we specify that the license held on the component is GPL. To this end, we adopt the Dublin Core rights management element. The following XML document describes the component authorship and uses a qualified name to reference a concept in our license ontology (*lic:GPL*), which is a taxonomy of software licenses.

```
<metadata xmlns=...>
   <dc:rights> lic:GPL </dc:rights>
   <dc:creator> Dave Developer </dc:creator>
</metadata>
```

These XML-based artifacts belonging to the *AccountBalance* component are described according to the metaschema instance and stored in the repository using the repository component's API.

Subsequently, these artifacts can be extracted, provided that a format wrapper has been defined, and inserted into the knowledge base. The ontologies together with the concepts used in our example are visualized in Figure 4. The artifact ontology provides the skeleton to describe components and related artifacts. It is supported by three domain ontologies providing the background knowledge for realizing the three use cases. As an example, the finance ontology (*fin:*) defines that *PrivateCustomer* is subsumed by *Customer*.

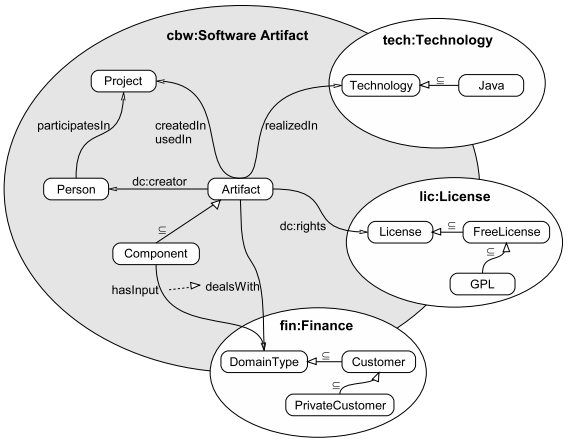


Figure 4: Part of the KOntoR ontology

The knowledge component can be applied to define a wide range of queries as required by the competency questions Q.1-Q.3 formulated in section 1. The following SPARQL query returns all artifacts dealing with the *Customer* business object as postulated in use case scenario Q.1:

```
SELECT ?cbw:artifact WHERE
{?cbw:artifact <cbw:dealsWith>
<fin:customer>}
```

In this query, the *AccountBalance* component, which uses an annotation to specify *PrivateCustomer* as its input parameter, is returned. This is possible since the knowledge base is aware of *AccountBalance* being an *Artifact*, *PrivateCustomer* being a specialization of *Customer* and *hasInput* being a subproperty of *dealsWith*.

According to example Q.2, we specify another query in order to verify if component A and component B can be used in the product to be developed under a proprietary license:

```
SELECT ?compA, ?compB WHERE {
 ?compA <dc:rights> ?lic1;
 ?compB <dc:rights> ?lic2;
 ?lic1 <lic:isCompatible> ?lic2}
```

The third example query Q.3 is about finding a developer who is experienced in building banking applications in Java:

```
SELECT ?developer WHERE {
 ?developer <dc:creator> ?comp;
 ?comp <cbw:realizedIn> <tech:Java>;
 ?comp <cbw:dealsWith> <fin:DomainType>}
```

The information that the component is realized in Java can be extracted from the enclosed binary package not described in this example.

5. Summary and conclusion

In this paper, we addressed two major problems underlying current software reuse systems: the low integration of reusable assets and the insufficient utilization of knowledge about the artifact's domain. The presented KOntoR architecture and prototype implementation takes a two-step approach for an evolutionary adoption in software processes. First, we contributed a repository component which achieves the requested integration of artifact metadata. Second, the knowledge component uses ontologies to capture the background knowledge required for declarative queries. As shown in the examples, the system is capable of supporting a wide range of tasks that are currently not supported by other reuse systems.

For future work we plan to develop additional user interfaces for different interactions with the repository. The options under consideration are a semantic wiki and a context extraction module for an active repository (cf. [Ye01]). Moreover, there are some open tasks in the core of the system. The interplay between artifact metadata, knowledge base and external data sources should be generalized. Not only does this include a mechanism for mappings between equal concepts and instances with different representations but also an efficient strategy for fetching just the required metadata from the data sources when a declarative query is executed. Finally, we are planning to evaluate the system in a real-world environment within the scope of the CollaBaWue project.

6. References

[1] W.B. Frakes and K. Kang, "Software Reuse Research: Status and Future", in: IEEE Trans. on Softw. Eng., vol 31, no. 7, 2005, pp. 529-536.

[2] M. Uschold and M. Gruninger, "Ontologies: principles, methods, and applications", Knowledge Engineering Review, vol. 11, 1996, pp. 93-155..

[3] C. Szyperski, "Component Software - Beyond Object-Oriented Programming". Addison-Wesley, 2nd ed., London, 2002.

[4] A. Mockus and J. D. Herbsleb, "Expertise browser: a quantitative approach to identifying expertise", in Proc. of the 24th Int. Conf. on Software Engineering (ICSE), Orlando 2002, pp. 503-512.

[5] V. R. Basili and G. Caldiera, "Improve Software Quality by Reusing Knowledge and Experience", Sloan Management Review, 1995, pp. 55-64.

[6] A. Mili, R. Milli and R.T.: Mittermeir, "A survey of software reuse libraries", in: Annals of Software Engineering, vol. 5, 1998, pp. 349-414.

[7] T. J. Biggerstaff and C. Richter, "Reusability framework, assessment, and directions", in Software reusability: vol. 1, concepts and models: ACM Press, 1989, pp. 1-17.

[8] C.W. Krueger, "Software Reuse", ACM Comp. Surv., vol. 24, 1992, pp. 131-183.

[9] R. Prieto-Diaz, "Implementing Faceted Classification for Software Reuse", in: Comm. ACM, vol. 34, no. 5, 1991, pp. 88-97.

[10] P. Vitharana, F. Zahedi and H. Jain, "Knowledge-Based Repository Scheme for Storing and Retrieving Business Components", in: IEEE Trans. on Softw. Eng., vol. 29, no. 8, 2003, pp. 649-664.

[11] J. Bosch,"Software Product Lines: Organizational Alternatives", In Proc. of the 23rd Int. Conf. on Software Engineering (ICSE), Toronto, 2001, pp 91-100.

[12] J. A. Espinosa, R. E. Kraut, J. F. Lerch, S. Slaughter, J. D. Herbsleb and A. Mockus: "Shared Mental Models and Coordination in Large-Scale, Distributed Software Development", In Proc. of the Int. Conf. in Information Systems (ICIS), New Orleans, 2001, pp. 513-518.

[13] D. Cubranic, G.C. Murphy, J. Singer, K. S. Booth, "Hipikat: A Project Memory for Software Development", in: IEEE Trans. on Softw. Eng. vol 31, no. 6, 2005, pp. 446-465.

[14] Y. Ye and G. Fischer, "Reuse-Conducive Development Environments", Autom. Softw. Eng. vol 12, no. 2, 2005, pp. 199-235.

[15] P. T. Devanbu, R. J. Brachman, P. G. Selfridge and B. W. Ballard, "LaSSIE: A Knowledge-Based Software Information System", in Comm. ACM, vol 34, no. 5, 1991, pp. 34-49.

[16] C.A. Welty, "Software Engineering", in: Description Logic Handbook, 2003, pp. 373-387.

[17] R.A. Falbo, D. O. Arantes and A.C.C. Natali, "Integrating Knowledge Management and Groupware in a Software Development Environment", In Proc. of the 5th Int. Conf. on Practical Aspects of Knowledge Management (PAKM), Vienna, 2004, pp. 94-105.

[18] D. Oberle, "Semantic Management of Middleware", Springer, New York. February 2006.

[19] H.-J. Happel, A. Korthaus, S. Seedorf and P. Tomczyk, "Ein Ansatz zur formatneutralen Verwaltung von Metadaten in komponentenorientierten Softwareprozessen", in: Proc. of Software Engineering 2006, Leipzig, March 28-31, 2006, pp 181-192.

[20] R. Akkiraju et al.: "Web Service Semantics - WSDL-S", W3C Member Submission, 7 Nov., 2005.

Automatic Monitoring of Control-flow Through Inheritance Hierarchies

Benjamin Tyler and Neelam Soundarajan
Computer Science & Eng., Ohio State University, Columbus, OH 43210
tyler, neelam@cse.ohio-state.edu

ABSTRACT

Polymorphism, based on inheritance and dynamic binding in standard object-oriented languages allows designers to customize the behavior of functions defined in particular base classes by suitably redefining, in derived classes, other functions that they invoke. But at the same time, polymorphism, especially when used in conjunction with the `super` mechanism that most OO languages provide, can result in complex control-flow among methods defined in various classes. We develop an approach that can be used by the designer to automatically trace this control-flow; we also present results from a prototype implementation.

1. INTRODUCTION

A key aspect of the Object Oriented (OO) approach is *polymorphism*[1]. Polymorphism enables a derived class designer to construct varied derived classes by redefining an appropriate set of functions of the base class. Given such redefinitions, not only will the redefined functions exhibit new behavior, but since polymorphism ensures that calls in other functions of the base classes to these functions are dispatched to their redefined versions (assuming that the objects in question are instances of the derived classes), these other functions will also exhibit suitably enriched behavior. But polymorphism also poses serious difficulties. A fundamental problem [10, 2] has to do with the way that control flows between methods defined in different classes of an OO program. Our goal in this paper is to present an approach that exploits polymorphism in helping analyze this control flow.

There are two issues that we must consider. The first involves the mechanisms of inheritance and dynamic binding. Thus in the example presented in [2], the problem shows up as follows: There are five classes, `C1` through `C5`, with `C5` being a derived class of `C4`, which is a derived class of `C3`, which in turn is a derived class of `C2`, etc. A method `c()` is defined in the class `C2`, inherited by `C3`, and redefined in `C4`, and inherited by `C5`. Another method `a()`, defined in `C1` contains, in its body, a call to `c()`. When this `a()` is applied to an object that is an instance of `C5`, dynamic binding will ensure that the call to `c()` that is made from within the body of `a()` will be dispatched to the `c()` defined in `C4`. On the other hand, if this `a()` were to be applied to an object that is an instance of `C3`, the same call will go to the `c()` defined in `C2`. This makes it difficult to follow the control-flow that results from applying a method such as `a()`; although that method is inherited by the various derived classes, what it actually does, in particular which method bodies it invokes as it executes, depends critically on the particular class of which the object in question is an instance.

The second aspect that contributes to the complexity of control-flow is, somewhat paradoxically, one designed to avoid the dynamic binding. Thus one of the methods, say, `c()` that is redefined in a derived class such as `C4` may contain, in the body of that redefinition, a call such as `super.c()`. This is because, often the redefinition of a method such as `c()` in a derived class has to perform all of the tasks carried out by the base class definition of the method, plus some additional activities typically related to the additional state (in the form of new member variables) of the derived class. And the former task is best achieved by the call `super.c()`. But this means that control transfers to the base class definition of the method –which was supposedly superseded by the derived class definition– which might invoke other methods that will be dynamically dispatched, unless those invocations also use `super`, etc.

We use the term *up-call* to refer to calls using the `super` mechanism since control will go from the current method to a method defined in an ancestor class. Similarly, we use *down-call* to refer to calls that are dispatched based on the class of the object. This is not always an accurate description; if a method `m()` defined in `C4` and not redefined in `C5` invokes `n()` in its body and `n()` is defined in `C3` and not redefined in `C4` or `C5`, and `m()` is applied to an instance of `C5`, then this call will be handled by `C3.n()`; thus control flows from `C4.m()` up to `C3.n()`. By contrast, the call `super.n()` necessarily results in control flowing up.

Taenzer, Ganti, and Podar [10] coined the term "yo-yo problem" to convey the effect of dynamically dispatched calls that typically transfer control to methods defined in derived classes, alternating with calls using the `super` mechanism that transfer control to methods in ancestor classes. Binder [2] argues that the "[l]oss of intellectual control that results from spaghetti polymorphism (the yo-yo problem) . . . " is one of the unique bug hazards of the OO approach.

Given this complexity, a graphical representation –which we call a *yo-yo graph*– of control flow through the inheritance hierachy would clearly be useful. Further, the control flow must be monitored automatically by a tool as the system executes; the information collected can then be used to generate the yo-yo graph. Similar graphs, generated by hand have been used by various authors.

In this paper, we present an approach *that exploits polymorphism* that allows us to automatically track control flow through inheritance hierarchies. In our approach, *no* changes are made to the classes whose methods are to be monitored as far as tracking *down*-calls are concerned, and only minimal changes are made for tracking *up*-calls. We have implemented our approach in a prototype tool which, given a system and the list of classes and methods to be tracked, makes the needed changes to the system, monitors the system at runtime, and generates the system's yo-yo graph.

[1]Throughout, by 'polymorphism' we mean *inclusion* or *subtype* polymorphism [3] achieved in standard OO languages such as *Java* via *inheritance* and *dynamic binding*.

To appear in SEKE 2006.

In Section 2, we present a fleshed-out version of an example from [10, 2]. In Section 3, we develop our approach to tracing both down-calls and up-calls; provide some details of our prototype implementation based on our approach; and present results of using it on the case-study. In Section 4, we discuss related work. In Section 5, we consider directions for future work.

2. CONTROL FLOW

Consider the program[2] shown in Fig. 1 consisting of classes C1 through C5, with each class (except C1) being a derived class of

```
abstract class C1 {
   protected int x = 0;
   public void a() { x++; x = b(); c(x-1); }
   abstract public int b();
   abstract public void c(int k);  }
class C2 extends C1 {
   protected int y = 0;
   public int b() { y = 2*x; int j = d(y); return y+j; }
   public void c(int k) { x = x - k; }
   public int d(int k) { c(k+1); return x; }  }
class C3 extends C2 {
   protected boolean p;
   public void a() { p = !p; super.a(); }
   public int b() { return super.b(); }   }
   public void c(int k) { x = x + k; }
   public int d(int k) { return x - 1; }  }
class C4 extends C3 {
   protected int z;
   public void a() { super.a(); z++; }
   public void c(int k) { super.c(k); z=z+x; }
class C5 extends C4 {
   protected boolean q = true;
   public int d(int k) { q = !q; return super.d(k); }
   public static void main(String[] args)
                { C5 c5 = new C5(); c5.a(); }   }
```

Figure 1: Program to be traced (original source code)

the one immediately above it. This program is based on the one in [2]; the only changes are that the individual method bodies have been defined to perform specific actions. But these actions are not really relevant; our focus is on the control flow.

The main() function defined in C5 creates an instance of C5 and invokes a() on it. Since the closest ancestor of C5 that has a definition of a() is C4, that definition will be invoked. That method invokes super.a() which calls C3.a(), which also invokes super.a() which calls C1.a(). That method invokes b() and then c() and these calls will be dispatched to their respective definitions applicable to instances of C5, i.e., C3.b() and C4.c(), etc.

Fig. 2, based on the one in [2], represents the static inheritance structure of our program as well as the control flow. The left side of the figure depicts, for each class, the methods inherited from the parent class (these methods are labeled inh), defined or redefined in the class (these methods have no labels), or as being redefined but the redefinition including a super call (labeled ref for "refined). On the right is the yo-yo graph representing the control flow when the call c5.a() is executed. This graph seems to contain some errors. First, d() is redefined in C3, not inherited by it. Therefore the arrow from C5.d to C2.d representing the super.d() call in the body of C5.d() should instead go to C3.d. To further add to the confusion, the discussion in [2] states that C5.d() invokes super.a(). If that were the case, the arrow from C5.d should go to

[2]For concreteness, we use *Java* in our discussion.

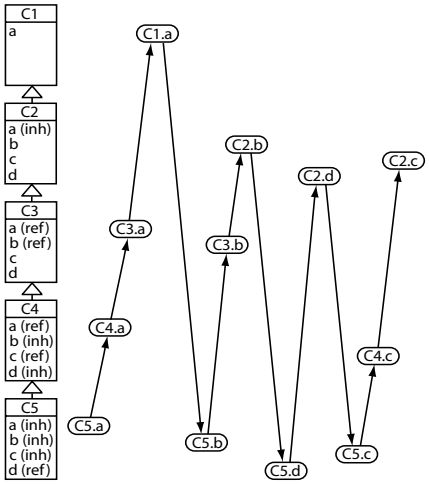


Figure 2: Yo-yo graph (generated by hand)

C3.a, not C2.d. Similar problems can be seen with c(). c() is redefined in C3, not inherited. Therefore, the super.c() call in C4 should go to C3.c, not C2.c; and, as in the case of C5.d(), an accompanying table states that C4.c() invokes super.a() (which would be inconsistent with the arrow from C4.c to C2.c). It is possible that the problem lies with the inheritance diagram. Thus according to the table accompanying the figure, c() and d() are *inherited* by C3, not redefined in this class. In that case, the arrow from C5.d to C2.d would indeed be correct as would the one from C4.c to C2.c. But the confusion regarding calls to super.a() that, according to the table, appear in C5.d() and C4.c(), still remains. In any case, the example demonstrates that even in simple systems, control flow can be complex.

3. AUTOMATIC MONITORING

Corresponding to each class Ci, we will introduce a derived class TCi and define certain methods in TCi. If we are interested in seeing what control-flow would result when a method m() is applied to an object that is an instance of, say, C4, we create an object of type TC4 and apply m() to it. The TCi classes will be defined in such a way that the resulting control-flow will be essentially the same as if we had applied m() to an instance of C4; the only difference is that the "down" calls will be "intercepted" by the methods we define in TC4 which will record information about the call and then forward the call to the actual method that would have received the call if the object had been an instance of C4.

The up-calls present a more difficult challenge. When a call such as super.n() is made from within, say, C4.m(), control flows up to the n() defined in the closest ancestor of C4, *independent of the class of the object*. Therefore, there is no way to intercept such a call. Given this, we will have to make certain minor modifications in the classes Ci. With this, we will be able to record information about both down-calls and up-calls.

3.1 Tracking Down-calls

Consider a call to m() in the body of a method n() of a class Cj. Suppose the object on which m() is invoked is an instance of Ci. When this call is executed, it will be dispatched to the definition of m() that is in Ci or, if m() is not (re-)defined in Ci, the one in the closest ancestor of Ci that has such a definition. To intercept such calls, in TCi, we will redefine every method that is applicable to objects of type Ci. Consider the class TC4 in Fig. 3. The methods applicable to C4 objects are a(), b(), c(), and d(). We have rede-

```
class TC4 extends C4 {
  public void a() {
    //. . . save information about this call . . .
    super.a();
    //. . . save information about return from call . . .  }
  public int b() {
    //. . . save information about this call . . .
    int x = super.b();
    //. . . save information about return from call . . .
    return x;  }
  public void c(int k) {. . . similar to a(); super.c(k);. . . }
  public int d(int k) { . . . similar to b() . . . }
```

Figure 3: TC4 class (first version)

fined these in TC4. The redefinitions save information about the method call, invoke the method defined in the parent class (C4), and when that call returns, save information about the results, etc., and then return to the caller.

Now method b() applied on an instance C4 will behave exactly as before. If the method is applied on an instance of TC4, the call will also behave in a similar manner except that information about the calls and returns to the various methods will be recorded by the methods defined in TC4. We have not indicated how the information about the calls and the returns is saved, but those are primarily matters of detail that we will briefly address in Section 3.3.

3.2 Tracking Up-calls

Consider the call super.a() that appears in the definition of C3.a(). When this call is executed, control will transfer to C1.a() (since C2 does not redefine a()), independent of the type of the object. Hence we cannot intercept this call. Instead, we must rewrite these calls in such a manner that they can be intercepted.

```
class C4 extends C3 {
  public int z;
  public void a() { C4_super_a(); z++; }
  public void c(int k) { C4_super_c(k); z=z+x; }
  public void C4_super_a() { super.a(); }
  public void C4_super_c(int k) { super.c(k); }
```

Figure 4: Modified class C4

The C4 in Fig.4 differs from the original in two respects. First, we have introduced two methods, C4_super_a() and C4_super_c() each of which simply calls the corresponding super method. Second, the super calls that appeared in the original C4 have been replaced by calls to these new methods. We have not introduced methods C4_super_b() or C4_super_d() since there are no calls of the form super.b() or super.d() in the original C4.

The modified C4 will still behave in the same way as the original C4 as far as instances of C4 are concerned. To track the up-calls, we need to modify TC4. In the TC4 defined in Fig. 5,

```
class TC4 extends C4 {
  // a(), b(), c(), d() as in Fig. 3
  public void C4_super_a() {
    //. . . save info about this *super* call . . .
    super.C4_super_a();
    //. . . save info about return from call . . .  }
  public void C4_super_c(int k) { . . . similar . . . }
```

Figure 5: TC4 class (second version)

we have redefined C4_super_a() and C4_super_c() in the same way as we redefined the original methods a(), b(), etc., in Fig. 3. Therefore, if we use an object that is an instance of TC4, rather than an instance of C4, and apply the method a() to it, the call to C4_super_a() in the modified C4 will be dispatched to the redefinition of C4_super_a() in TC4. This method, as usual, saves information about this call, then forwards the call to C4.C4_super_a(), which in turn forwards the call to the a() defined in C3, which was the method we called from the original C4.a().

But consider what happens when C3.a(), executes. We would have modified C3 in the same way as C4. So the super.a() call in the original C3.a() would now be the call C3_super_a(). This method would be defined to call super.a(). Moreover, C3_super_a() is *not* redefined in TC4, so this will be handled by C3.C3_super_a() which will call super.a(), i.e., the one defined in C1. That is what we want but we *did not intercept* this super call.

```
class TC4 extends C4 {
  // a(), b(), c(), d() as in Fig. 3
  public void C4_super_a() {// as in Fig. 5 }
  public void C4_super_c(k) { . . . similar . . . }
  public void C3_super_a() {
    //. . . save info about *this* super call (to C2.a()) and
    // the current state of object. . .
    super.C3_super_a();
    //. . . save info about return from call and
    // current state of object, results . . .  }
  public void C3_super_b(k) { . . . similar . . . }
  public void C3_super_c(k) { . . . similar . . . }
  public void C2_super_c(k) {
    // this one is not needed since there are no super.c()
    // calls in the methods of C2, but it would be more
    // uniform to have all of these . . . }
  // rede nitions  of other _super_ methods  }
```

Figure 6: TC4 class (final version)

Hence, in TC4, in Fig. 6, we redefine methods such as C3_super_a(), and by extension, C2_super_a(), etc. With this addition to TC4, if the method a() is applied to an instance of TC4, not only is the call to C4_super_a() call intercepted, but also the call to C3_super_a(). Therefore, this final version of TC4 will allow us to intercept all down-calls and up-calls invoked upon objects of type TC4 and save the necessary information about these calls.

3.3 Implementation Details and Results

We have implemented our approach in a prototype tool, *PolyTracker*[3]. The tool takes as input, the original program, i.e., the classes we have referred to as C1, C2 etc. It modifies these classes, replacing the super calls by the corresponding Ci_super_ call. It also introduces the definitions for the Ci_super_ methods in these classes, similar to the ones in Fig. 4. Next it produces the tracing classes, TC1, TC2, etc. As we saw, these classes do not depend on the details of the methods in the original classes.

In the modified program, *PolyTracker* creates an instance of the appropriate trace class TCi. To log the information about the various calls and returns that are intercepted, all of the TCi classes use an instance of Tracker, a *singleton* class that *PolyTracker* uses for this purpose. A Tracker object is essentially a *sequence* of elements, each of which contains information about a single call or a return. We omit the details.

Once the modified Ci and the tracing classes have been produced, the resulting set of classes (as well as Tracker and related classes) are compiled using the standard *Java* compiler. The program is then executed. During execution, the various TCi classes

[3] *PolyTracker*, its documentation, and examples are available at: www.cse.ohio-state.edu/~tyler/pTracker/index.html

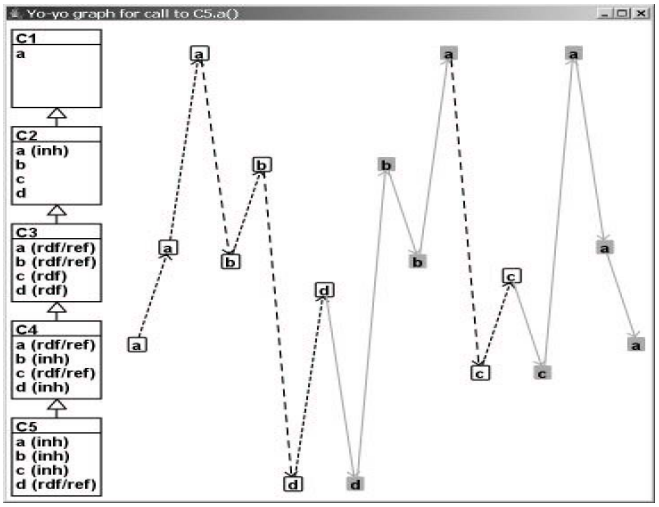


Figure 7: Yo-yo graph (generated by *PolyTracker*)

collect and save information about the control flow in the `Tracker` object. Once the execution completes, the tool renders the yo-yo graph. Again we will omit the details (which are available at the *PolyTrakcer* site). Fig. 7 shows the graph generated by *PolyTracker* for our simple case-study in Fig. 1. On the left side of the figure, we have the classes along with the inheritance relations. Inherited methods are labeled `inh`. Methods that are redefined and, in their redefinition, include a `super`-call are labeled `rdf/ref`. Methods that are redefined but do not include such a call are labeled `rdf`. Methods that are defined for the first time do not have any label.

In the yo-yo graph, *PolyTracker* uses different styles of arrows (short versus long dashes) to distinguish between up-calls and down-calls. Also, no *call nodes* are created for class methods that are inherited since, in this case, control simply "passes through" to the first ancestor class that implements the method. The yo-yo graph also depicts, by the *grey* nodes and arrows, the flow of control *as it returns* to the calling method. This results in a more informative graph especially when a method makes *multiple* calls.

4. RELATED WORK

We have already mentioned the work of Taenzer *et al.* [10] and Binder [2]. Lange and Nakamura [5, 6] present a technique for tracing the execution of an OO program. This is based on accessing, at the machine level, specific information contained in the run-time structures. They also discuss various graphical ways to display the information. De Pauw *et al.* [8] present an approach to visualizing the execution of OO programs. Their technique requires insertion of substantial amounts of code in the individual classes, and depends on the *RTTI* mechanism of *C++* to access information about the types of given objects. Jerding [4] discusses ways in which the execution of OO programs can be visualized. His main concern is to extract the most relevant pieces from the large amount of information that may be obtained.

Several authors have addressed problems related to testing of polymorphic interactions [7, 1, 9] in OO systems and related questions concerning *coverage*. Although this question is clearly important, it is orthogonal to our work which is concerned with automatically tracing the flow of control among the various methods.

5. DISCUSSION

The use of polymorphism/dynamic binding and the `super` mechanism can lead to relatively complex control flow in OO systems. Representing this graphically in the form of yo-yo graphs can be of considerable help to designers and implementers but the very complexity of the flow means that generating the graphs by hand can result in subtle mistakes in the graphs. Our work shows how the power of these same mechanisms can be exploited to build a tool that can track the control flow automatically and use the data collected during the monitoring to generate the yo-yo graphs.

We conclude with some pointers for future work. So far in our work, we have only dealt with a single object and the control-flow that results from applying a particular method on it. In practical systems we will have to deal with multiple objects. Our approach should be directly applicable to such situations. Indeed, we can *selectively* trace methods invoked on *some* objects and ignore those invoked on others. To do this, we simply have to ensure that the objects for which calls should be traced, should be of the appropriate `TC` type, while the others should be of their original `C` type.

A more complex issue has to do with the fact that the program under study may have a class `C1` that has a member variable `x` of type, say, `C2` rather than the simple types such as `int`s. Suppose the program has an object `o1` of type `C1`. What if we wish to trace not just method invocations on `o1` but those invoked on `o1.x`? That is an object of type `C2` and normally we would simply create and use an instance of type `TC2` for this purpose. But that will not work here since this is a part of the `o1` object, rather than being an independent object. We could create an `o1` object that is of type `TC1` but the `x` component of this object will still be of type `C2` rather than `TC2`. Finding a suitable solution to this problem is important since practical systems are likely to have such objects that have other objects as components and designers and analysts are likely to be interested in the behaviors of those components. Less challenging but still of practical importance is the question of finding suitable ways to display information about the control-flow when the system has several objects that we are interested in; here, we should be able to build on previous work in visualization [4, 6].

6. REFERENCES

[1] R. Alexander and J. Offutt. Criteria for testing polymorphic relationships. In *Int. Symp. on Softw. Reliability Eng.*, pages 15–23, 2000.

[2] R. Binder. *Testing "OO" systems*. Addison-Wesley, 1999.

[3] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *ACM Comp. Surveys*, 1985.

[4] D. Jerding. Visualizing patterns in the execution of "oo" programs. In *Proceedings of CHI '96*. ACM, 1996.

[5] D. Lange and Y. Nakamura. Interactive visualization of design patterns. In *Proc. of OOPSLA '95*, pages 342–357. ACM, 1995.

[6] D. Lange and Y. Nakamura. Object-oriented program tracing and visualization. *Computer*, pages 63–70, Oct. 1997.

[7] T. McCabe, L. Dreyer, A. Dunn, and A. Watson. Testing an object-oriented application. *J. of the Quality Assurance Institute*, 8(4):21–27, 1994.

[8] W. De Pauw, R. Helm, D. Kimelman, and J. Vlissides. Visualizing the behavior of object-oriented systems. In *Proc. of OOPSLA '93*, pages 326–337. ACM, 1993.

[9] A. Rountev, A. Milanova, and B. Ryder. Fragment class analysis for testing of polymorphism in java software. In *Int. Conf. on Softw. Eng.*, pages 210–220, 2003.

[10] D. Taenzer, M. Ganti, and S. Podar. Problems in object-oriented software reuse. In S. Cook, editor, *Proceedings of ECOOP '89*, pages 25–38. British Computer Workshop Series, 1989.

A Constraint-based Correct Call Pattern Semantics for Prolog as an Abstraction of Decorated Tree Semantics

Lingzhong Zhao, Tianlong Gu, Junyan Qian, and Guoyong Cai
Department of Computer Science, Guilin University of Electronic Technology, Guilin, China 541004
gllzzhao@gliet.edu.cn

Abstract

Decorated tree semantics [7] is a goal-independent denotational semantics for Prolog that is designed with the aim to provide a general framework for the abstract analysis of generic properties of logic programs. Correct call patterns (ccp) are a useful observable for optimization of Prolog programs. Designing appropriate ccp semantics is the base for semantics-based correct call patterns analysis of Prolog programs. By introducing the height of ccp constraint we present a new constraint-based semantic domain, and successfully achieve ccp semantics as an abstraction of decorated tree semantics. The obtained semantics enjoys the feature of goal-independence and is suitable for goal-independent analysis of Prolog programs.

1. Introduction

Static analysis of Prolog programs is essential in optimizing Prolog compilers and program verification systems. Information about call patterns (cp) and correct call patterns of a goal is useful for the optimization of Prolog programs [1,4]. Call patterns are the atoms that are selected during the SLD-derivation of a goal. Correct call patterns are those call patterns that belong to successful derivations. The development of appropriate cp or ccp semantics is the base for semantics-based call patterns or correct call patterns analysis of Prolog programs.

Decorated tree (DT) semantics is a goal-independent denotational semantics proposed by Spoto [7]. A feature of this semantics is that it uses syntactic objects (substitution with control information), rather than functions, as the denotational domain, which makes it suitable for abstract interpretation [1,8].

Semantics for computed answers (ca) and call patterns have been given as specializations of DT semantics. Both of the semantics use as semantic domain the sequences of constraints, which have been shown to be useful and promising in the analysis of Prolog programs [5,6]. Further abstractions of ca and cp semantics have been proposed to get finitely computable abstract semantics for program analysis [8]. Yet it's not clear how the semantics for correct call patterns can be obtained from DT semantics.

Our work shows that the task of achieving ccp semantics from DT semantics is not trivial. Following the approach of abstract interpretation [2,3], we propose a new semantic domain consisting of four kinds of constraints. Based on this domain we obtain ccp semantics as an abstraction of DT semantics. For the limit of space, we refer to [7] for details about DT semantics.

2. Preliminaries

We assume the familiarity with the basic algebraic structures and Prolog language. The basic notations are used in the usual way. A sequence is an ordered collection of elements possibly with repetitions. The set of all sequences of elements of E is denoted by Seq(E) and the set of non-empty sequences of elements of E is denoted by $\mathrm{Seq}^{+}(E)$. :: represents sequence concatenation. A sequence is denoted by a variable with a tilde sign on it. The empty sequence is explicitly written as ε.

Abstract interpretation is a general theory of semantics approximation. A common method for formalizing abstract interpretation is by means of a Galois connection.

Definition 2.1 Given two partial ordered sets (posets) $<P,\sqsubseteq>$ and $<P^a,\sqsubseteq^a>$, a Galois connection between them is a pair $<\alpha,\gamma>$ such that α is a total map from P to P^a, γ is a total map from P^a to P, and $\forall p\in P$: $\forall p^a\in P^a$: $\alpha(p)\sqsubseteq^a p^a \Leftrightarrow p\sqsubseteq\gamma(p^a)$. α and γ are respectively called the abstraction and the concretization maps of the connection.

The abstract interpretation using Galois connection can be formalized as follows.

Proposition 2.2 (Proposition 23 in [2]) Let $<P,\sqsubseteq>$ and $<P^a,\sqsubseteq^a>$ be two posets with $\perp$ and $\perp^a$ as the minimal elements, respectively, $<\alpha,\gamma>$ is a Galois connection between P and P^a, $\perp^a=\alpha(\perp)$, φ: $P\mapsto P$ is such that $lfp(\varphi)=\mathrm{lub}_{n\geqslant 0}\,\varphi^n(\perp)$ (where lub is the least upper bound operator) and φ^a: $P^a\mapsto P^a$ is monotonic, then $\alpha\circ\varphi=\varphi^a\circ\alpha$ implies $\alpha(lfp(\varphi))=lfp(\varphi^a)$. The condition $\alpha\circ\varphi=\varphi^a\circ\alpha$ is called local correctness condition and $\alpha(lfp(\varphi))=lfp(\varphi^a)$ is called global correctness condition.

By Tarski's fixpoint theorem the following proposition obviously holds as a consequence of proposition 2.2.

Proposition 2.3 Let $<P,\sqsubseteq,\perp>$ and $<P^a,\sqsubseteq^a,\perp^a>$ be two complete lattices with $\perp$ and $\perp^a$ as the minimal elements,

respectively, $<\alpha,\gamma>$ is a Galois connection between P and P^a, φ: $P \mapsto P$ and φ^a: $P^a \mapsto P^a$ are two operators such that φ is continuous and φ^a is monotonic, then $\alpha\circ\varphi=\varphi^a\circ\alpha$ implies $\alpha(lfp(\varphi)) = lfp(\varphi^a)$.

In this paper proposition 2.3 is used to prove the correctness of abstract semantics. Moreover, the proof can be further simplified using the following result.

Proposition 2.4 Let $<P,\sqsubseteq>$ and $<P^a,\sqsubseteq^a>$ be two posets with $\perp$ and $\perp^a$ as the minimal elements, respectively, and φ: $P \mapsto P$, φ^a: $P^a \mapsto P^a$ and α: $P \mapsto P^a$ be three monotonic maps such that $\alpha(\perp)=\perp^a$ and $\alpha\circ\varphi=\varphi^a\circ\alpha$. Then P and P^a can be extended to two complete lattice P' and $P^{a\prime}$, respectively, and α, φ and φ^a to continuous maps α', φ' and $\varphi^{a\prime}$ such that $<\alpha',\gamma>$, for a suitable γ, is a Galois connection between P' and $P^{a\prime}$, $\alpha'(\perp)=\perp^a$ and the correctness condition $\alpha'\circ\varphi' = \varphi^{a\prime}\circ\alpha'$ holds.

By the above proposition we only have to define semantic operators and abstraction map on posets instead of complete lattices when showing the correctness of abstract semantics.

An abstract syntax was used for Prolog programs in [7] to simplify the semantic operators. Without loss of generality all the predicates in Prolog programs are assumed to be unary. The clause has the form $p(x)$:- G_1 or…or G_n with $n\geqslant 1$, where $G_1,\ldots,G_n$ are goals defined by the grammar:

$G ::= c \mid p(x) \mid G \text{ and } G \mid \text{exists } x.\ G \mid \text{cut}(G) \mid \text{cut}_d(G)$,

where c is a constraint, x is a program variable, d is a non-negative integer. The expression $p(x)$, where p is a predicate symbol, is called *procedure call*.

The basis constraints domain from which c is taken is defined as a lattice $<\mathcal{B}, \leq, \vee, \wedge, true, false>$ [7]. It is assumed that $\mathcal{B}$ contains the element $\delta_{x,z}$ for each pair of variables x and z, which represents the constraint identifying the variables x and z. Moreover, there is a family of monotonic operators $\exists_x$ on the set of constraints, representing the restriction of a constraint obtained by hiding all the information related to the variable x.

A goal is called *divergent* if and only if it contains a procedure call. The denotation $\text{div}(G)$ means G is divergent. A goal which is not divergent is said to be *convergent*. The map $con(G)$ associates to G the conjunction of the constraints that precede the first procedure call in G. The definition of $con(G)$ is as follows.

$$con(c) = c, \quad con(p(x)) = true$$
$$con(exists\ x.G) = \exists_x.con(G)$$
$$con(cut(G)) = con(cut_d(G)) = con(G)$$
$$con(G_1\ and\ G_2) = \begin{cases} con(G_1) & \text{if } \text{div}(G_1) \\ con(G_1) \wedge con(G_2) & \text{if not } \text{div}(G_1). \end{cases}$$

3. Semantics domain

In this section we propose a semantic domain for ccp semantics, which consists of four kinds of constraints.

We define $\mathbb{CCP} = C^c \cup C^d \cup C^! \cup C^{ccp}$, where

1) $C^c = \{o +^c b \mid o \in \mathcal{O}^1,\ b \in \mathcal{B}\}$ (convergent constraints),
2) $C^d = \{o +^d b \mid o \in \mathcal{O},\ b \in \mathcal{B}\}$ (divergent constraints),
3) $C^! = \{o +^! b \mid o \in \mathcal{O},\ b \in \mathcal{B}\}$ (cut constraints), and
4) $C^{ccp} = \{o +^x b + sc, p \mid o \in \mathcal{O},\ sc \in \mathcal{O},\ b \in \mathcal{B} \text{ and } x \in \mathbb{N}\} \cup \{^x sc \mid sc \in \mathcal{O} \text{ and } x \in \mathbb{N}\}$ where p is a procedure call and $\mathbb{N}$ is the set of non-negative integers which are intended to denote the heights of SLD-derivation of a goal when call patterns are obtained (correct call pattern constraints).

In the following sections we call x the height of constraint $(o +^x b + sc, p)$ or $(^x sc)$. Every decorated tree will be abstracted into a sequence of constraints in $\mathbb{CCP}$. Next we formally present our approach for achieving ccp semantics.

Let $\mathbb{DT}$ be the set of decorated trees defined in [7]. The abstraction of a sequence of decorated trees is defined as follows.

Definition 3.1 We define a map $\alpha_{\mathbb{CCP}}$: $\text{Seq}^+(\mathbb{DT}) \mapsto \text{Seq}^+(\mathbb{CCP})$ as follows. Let p be the leftmost procedure call in G, then

$$\alpha_{\mathbb{CCP}}(o + G, \varepsilon) = \begin{cases} o +^0 con(G) + false_{\mathcal{O}}, p :: o \odot^{\mathbb{CCP}} cutsseq(G) \\ \quad :: o +^d con(G) & \text{if } \text{div}(G) \\ {}^0(o \sqcap con(G) \propto obs) :: o \odot^{\mathbb{CCP}} cutsseq(G) \\ \quad :: o +^c con(G) & \text{if not } \text{div}(G) \end{cases}$$

$$\alpha_{\mathbb{CCP}}(o + G, \tilde{t}) = o +^0 con(G) + \sqcup \circ^{\mathbb{DT}}(\tilde{t}), p :: o \odot^{\mathbb{CCP}} cutsseq(G) :: inc^1(\alpha_{\mathbb{CCP}}(\tilde{t}))$$

$$\alpha_{\mathbb{CCP}}(\tilde{t}_1 :: \tilde{t}_2) = \alpha_{\mathbb{CCP}}(\tilde{t}_1) :: \alpha_{\mathbb{CCP}}(\tilde{t}_2),$$

where, given $o' \in \mathcal{O}$:

$$o' \odot^{\mathbb{CCP}} (\tilde{s}_1 :: \tilde{s}_2) = o' \odot^{\mathbb{CCP}} \tilde{s}_1 :: o' \odot^{\mathbb{CCP}} \tilde{s}_2$$
$$o' \odot^{\mathbb{CCP}} (o +^z b) = (o' \sqcap o +^c b), z = c, d, !.$$
$$o' \odot^{\mathbb{CCP}} ({}^n sc) = {}^n (o' \sqcap sc),$$
$$o' \odot^{\mathbb{CCP}} (o +^n b + sc, p) = o' \sqcap o +^n b + o' \sqcap sc, p,$$

inc^x is used to increase the heights of ccp constraints by x and is defined as:

$$inc^x(\tilde{s}_1 :: \tilde{s}_2) = inc^x(\tilde{s}_1) :: inc^x(\tilde{s}_2), inc^x({}^n sc) = {}^{n+x} sc,$$
$$inc^x(o +^n b + sc, p) = o +^{n+x} b + sc, p,$$
$$inc^x(o +^z b) = o +^z b,\ z = c, d, !.$$

and $cutsseq(G)$ is the sequence of cut constraints associated to a goal G and is defined the same way as that for Spoto's ca semantics.

[1] The symbol $\mathcal{O}$ denotes the set of observability constraints with $true_{\mathcal{O}}$ and $false_{\mathcal{O}}$ as the top and bottom element in $\mathcal{O}$, respectively; The functions $\propto obs$, "$\bullet$" and "-" are all operators defined on $\mathcal{O}$.

The following maps are needed to define the abstract operators used in ccp semantics.

$\circ^{\mathbb{CCP}}(\tilde{s}) = \sqcup_{o +^c b \in \tilde{s}} (o \sqcap b \propto obs)$

$k^{\mathbb{CCP}}(\tilde{s}) = \sqcup_{o +^! b \in \tilde{s}} (o \sqcap b \propto obs)$

$\delta^{\mathbb{CCP}}(\tilde{s}) = \sqcup_{o +^d b \in \tilde{s}} (o \sqcap b \propto obs)$

$\beta^{\mathbb{CCP}}(\tilde{s}) = k^{\mathbb{CCP}}(\tilde{s}) \sqcup \delta^{\mathbb{CCP}}(\tilde{s})$

$\xi^{\mathbb{CCP}}(\tilde{s}_1, \tilde{s}_2) = \sqcup_{o +^c b \in \tilde{s}_1} (o \sqcap (b \bullet \beta(\tilde{s}_2)))$

In order to define abstract operators for ccp semantics we present the following concepts.

Definition 3.2 A minimal regular subsequence (MRS) $\tilde{s}'$ of $\tilde{s}$ is the longest subsequence of $\tilde{s}$ such that $\tilde{s}'$ begin with a correct call pattern constraint and contains no other correct call pattern constraint. The correct call pattern constraint contained in $\tilde{s}'$ is denoted by *ccp*($\tilde{s}'$).

Definition 3.3 A regular subsequence $\tilde{s}'$ of $\tilde{s}$ is a subsequence of $\tilde{s}$ that is the concatenation of the MRSs of $\tilde{s}$.

Definition 3.4 The affecting area $\tilde{s}''$ of a MRS $\tilde{s}'$ of $\tilde{s}$ is the longest regular subsequence of $\tilde{s}$ beginning at the constraint next to the last constraint of $\tilde{s}'$ such that the height of *ccp*($\tilde{s}'$) is less than those of correct call pattern constraints in $\tilde{s}''$. The affecting area of a MRS $\tilde{s}'$ of $\tilde{s}$ is denoted by $\Gamma_{\tilde{s}}(\tilde{s}')$.

Now we are ready to give the basic operations on Seq$^+$($\mathbb{CCP}$).

Definition 3.5

Product operator $\otimes^{\mathbb{CCP}}$:

Let $\tilde{s} = \tilde{s}_1 :: \tilde{s}_2$ where $\tilde{s}_1$ and $\tilde{s}_2$ are regular subsequences of $\tilde{s}$, then:

$\tilde{s} \otimes^{\mathbb{CCP}} \tilde{s}' = \tilde{s}_1 \otimes^{\mathbb{CCP}} \tilde{s}' :: -\xi^{\mathbb{CCP}}(\tilde{s}_1, \tilde{s}') \odot^{\mathbb{CCP}} (\tilde{s}_2 \otimes^{\mathbb{CCP}} \tilde{s}')$

Let $\tilde{s}_i$ be a MRS of $\tilde{s}$, then:

1) if $\tilde{s}_i$ ends with a convergent constraint, e.g. $\tilde{s}_i = ^x sc :: \tilde{s}_i' :: o +^c b$, if $\tilde{s}' = o' +^n b' + sc, p :: \tilde{s}''$, then

$\tilde{s}_i \otimes^{\mathbb{CCP}} \tilde{s}' = inc^x(o \odot^{\mathbb{CCP}} (b \circledcirc^{\mathbb{CCP}} (o' +^n b' + sc, p :: \tilde{s}_i' :: \tilde{s}'')))$.

if $\tilde{s}' = ^n sc :: \tilde{s}''$, then

$\tilde{s}_i \otimes^{\mathbb{CCP}} \tilde{s}' = inc^x(o \odot^{\mathbb{CCP}} (b \circledcirc^{\mathbb{CCP}} (^n sc :: \tilde{s}_i' :: \tilde{s}'')))$.

2) if $\tilde{s}_i$ ends with a divergent constraint, then $\tilde{s}_i \otimes^{\mathbb{CCP}} \tilde{s}' = \tilde{s}_i$;

3) if $\tilde{s}_i$ has only one constraint or ends with a cut constraint, e.g. $\tilde{s}_i = o +^x b + sc, p :: \tilde{s}_i'$ (By definition of $\alpha_{\mathbb{CCP}}$, in this case *ccp*($\tilde{s}_i$) cannot be of the form $^x sc$), then: $\tilde{s}_i \otimes^{\mathbb{CCP}} \tilde{s}' = o +^x b + sc', p :: \tilde{s}_i'$, where

$sc' = \sqcup_{o +^c b \in \Gamma_{\tilde{s}}(\tilde{s}_i)} (o \sqcap (b \bullet \circ^{\mathbb{CCP}}(\tilde{s}')) \sqcap (- \sqcup_{o' +^c b' \in \Gamma_{\tilde{s}}(\tilde{s}_i) \text{ precedes } o +^c b} o' \sqcap (b' \bullet \beta^{\mathbb{CCP}}(\tilde{s}'))))$.

Cut operator $!^{\mathbb{CCP}}$:

Let $\tilde{s} = \tilde{s}_1 :: \tilde{s}_2$ where $\tilde{s}_1$ and $\tilde{s}_2$ are regular subsequences of $\tilde{s}$, then: $!^{\mathbb{CCP}} \tilde{s} = !^{\mathbb{CCP}} \tilde{s}_1 :: - \circ^{\mathbb{CCP}} (\tilde{s}_1) \odot^{\mathbb{CCP}} !^{\mathbb{CCP}} \tilde{s}_2$.

Let $\tilde{s}_i$ be a MRS of $\tilde{s}$, then:

1) if $\tilde{s}_i$ ends with a convergent constraint, e.g. $\tilde{s}_i = \tilde{s}_i' :: o +^c b$, then $!^{\mathbb{CCP}} \tilde{s}_i = \tilde{s}_i' :: o +^! b :: o +^c b$;

2) Otherwise, if $\tilde{s}_i$ ends with other constraint, then $!^{\mathbb{CCP}} \tilde{s}_i = \tilde{s}_i$.

Sum operator $\oplus^{\mathbb{CCP}}$:

Let $\tilde{s}_1 = o +^x b + sc_1, p :: \tilde{s}_1'$ and $\tilde{s}_2 = \tilde{s}_2' :: \tilde{s}_2''$, where $\tilde{s}_2'$ is the first MRS of $\tilde{s}_2$ and $ccp(\tilde{s}_2') = o +^x b + sc_2, p$, then

$\tilde{s}_1 \oplus^{\mathbb{CCP}} \tilde{s}_2 = o +^x b + sc_1 \sqcup (-\beta^{\mathbb{CCP}}(\tilde{s}_1) \sqcap sc_2), p :: \tilde{s}_1' :: -\beta^{\mathbb{CCP}}(\tilde{s}_1) \odot^{\mathbb{CCP}} \tilde{s}_2''$.

Uncut operator $\downarrow^{\mathbb{CCP}}$ is defined as:

$\downarrow^{\mathbb{CCP}} (\tilde{s}_1 :: \tilde{s}_2) = \downarrow^{\mathbb{CCP}} (\tilde{s}_1) :: \downarrow^{\mathbb{CCP}} (\tilde{s}_2)$

$\downarrow^{\mathbb{CCP}} (o +^c b) = o +^c b, \quad \downarrow^{\mathbb{CCP}} (o +^d b) = o +^d b$

$\downarrow^{\mathbb{CCP}} (o +^! b) = \varepsilon, \qquad \downarrow^{\mathbb{CCP}} (^n sc) = ^n sc$

$\downarrow^{\mathbb{CCP}} (o +^n b + sc, p) = o +^n b + sc, p$

Other operators are defined as follows.

$\exists_x^{\mathbb{CCP}}(\tilde{s}_1 :: \tilde{s}_2) = \exists_x^{\mathbb{CCP}}(\tilde{s}_1) :: \exists_x^{\mathbb{CCP}}(\tilde{s}_2)$, where

$\exists_x^{\mathbb{CCP}}(o +^z b) = \exists_x o +^z \exists_x b, \ z = c, d, !.$

$\exists_x^{\mathbb{CCP}}(^n sc) = ^n (\exists_x sc)$

$\exists_x^{\mathbb{CCP}}(o +^n b + sc, p) = \exists_x o +^n \exists_x b + \exists_x sc, p$

Given $b' \in \mathcal{B}$: $b' \circledcirc^{\mathbb{CCP}} (\tilde{s}_1 :: \tilde{s}_2) = b' \circledcirc^{\mathbb{CCP}} \tilde{s}_1 :: b' \circledcirc^{\mathbb{CCP}} \tilde{s}_2$

$b' \circledcirc^{\mathbb{CCP}} (o +^z b) = (b' \bullet o +^z b' \wedge b), z = c, d, !.$

$b' \circledcirc^{\mathbb{CCP}} (^n sc) = ^n (b' \bullet sc)$

$b' \circledcirc^{\mathbb{CCP}} (o +^n b + sc, p) = b' \bullet o +^n b' \wedge b + b' \bullet sc, p$

$\phi^G_{\mathbb{CCP}}(\tilde{s}) = true_{\mathcal{O}} +^0 con(G) + \circ^{\mathbb{CCP}}(\tilde{s}), p :: cutsseq(G) :: inc^1(\tilde{s})$, where p is the leftmost procedure call present in G;

$\psi^G_{\mathbb{CCP}}(\tilde{s}) = \tilde{s}$; $\tilde{s}[x/\alpha] = \exists_\alpha^{\mathbb{CCP}}(\delta_{x,\alpha} \circledcirc^{\mathbb{CCP}} \tilde{s})$.

It can be shown that the operators defined above are correct w.r.t. their counterparts for DT semantics.

Let I be a *correct call pattern interpretation* (*ccp interpretation* in short), i.e. a map from predicate symbols into Seq$^+$($\mathbb{CCP}$), we can define the immediate consequence operator $T_P^{\mathbb{CCP}}$ as follows.

$T_P^{\mathbb{CCP}}(I)(p) = \exists_y^{\mathbb{CCP}}(\delta_{y,\alpha} \circledcirc^{\mathbb{CCP}} \downarrow^{\mathbb{CCP}} (\phi^{p(y)}_{\mathbb{CCP}}(\mathcal{J}_P^{\mathbb{CCP}} [\![G_1]\!] I) \oplus^{\mathbb{CCP}} \cdots \oplus^{\mathbb{CCP}} \phi^{p(y)}_{\mathbb{CCP}}(\mathcal{J}_P^{\mathbb{CCP}} [\![G_n]\!] I)))$,

where p(y):-G_1 or…or G_n is the definition of p in program P, α is a variable not allowed in all programs and $\mathcal{J}_P^{\mathbb{CCP}}$ is defined as

$\mathcal{J}_P^{\mathbb{CCP}} [\![c]\!] I = ^0 c :: true_{\mathcal{O}} +^c c, \mathcal{J}_P^{\mathbb{CCP}} [\![p(x)]\!] I = I(p)[x/\alpha]$

$\mathcal{J}_P^{\mathbb{CCP}} [\![exists\ x.G]\!] I = \exists_x^{\mathbb{CCP}} \mathcal{J}_P^{\mathbb{CCP}} [\![G]\!] I$

$\mathcal{J}_P^{\mathbb{CCP}} [\![cut(G)]\!] I = !^{\mathbb{CCP}} \mathcal{J}_P^{\mathbb{CCP}} [\![G]\!] I$

$\mathcal{J}_P^{\mathbb{CCP}} [\![G_1\ and\ G_2]\!] I = \mathcal{J}_P^{\mathbb{CCP}} [\![G_1]\!] I \otimes^{\mathbb{CCP}} \mathcal{J}_P^{\mathbb{CCP}} [\![G_2]\!] I$

Let $\mathcal{J}_P^{\mathbb{DT}}$ and $T_P^{\mathbb{DT}}$ be the counterparts of $\mathcal{J}_P^{\mathbb{CCP}}$ and $T_P^{\mathbb{CCP}}$ in DT semantics, respectively. By the correctness of basic operators, we have the following theorem:

Theorem 3.6 Let *I* be a DT interpretation and *G* be a goal. We have:

1) $\alpha_{\mathbb{CCP}}(\mathcal{J}_P^{\mathbb{DT}}[\![G]\!]I) = \mathcal{J}_P^{\mathbb{CCP}}[\![G]\!]\alpha_{\mathbb{CCP}}(I)$
2) $\alpha_{\mathbb{CCP}}(T_P^{\mathbb{DT}}(I)) = T_P^{\mathbb{CCP}}(\alpha_{\mathbb{CCP}}(I))$.

4. Correct call pattern semantics and its correctness

Definition 4.1 On $\text{Seq}^+(\mathbb{CCP})$ we define the relation $\preceq$ as follows:

1) $o +^c b \preceq o +^c b$
2) $o +^d b \preceq \tilde{s}$ *iff* every $o' +^c b'$, $o' +^d b'$, $o' +^! b'$, $o' +^n b' + sc'$ and ${}^n sc'$ in $\tilde{s}$ is such that $o' \leq o$, $b' \leq b$, $sc' \leq o$ and $sc' \leq b$
3) $o +^! b \preceq o +^! b$
4) $o +^n b + sc_1,\ p \preceq o +^n b +^n sc_2,\ p$ *iff* $sc_1 \leq sc_2$
5) ${}^n sc_1 \preceq {}^n sc_2$ *iff* $sc_1 \leq sc_2$
6) $\tilde{s}_1 \preceq \tilde{s}_2$ if for every partition $\tilde{s}_{11} :: \tilde{s}_{12}$ of $\tilde{s}_1$, we can find a partition $\tilde{s}_{21} :: \tilde{s}_{22}$ of $\tilde{s}_2$ such that $\tilde{s}_{11} \preceq \tilde{s}_{21}$ and $\tilde{s}_{12} \preceq -\delta^{\mathbb{CCP}}(\tilde{s}_{11}) \odot^{\mathbb{CCP}} \tilde{s}_{22}$.

It can be shown that $\preceq$ is a partial ordering relation on $\alpha_{\mathbb{CCP}}(\alpha_{\mathbb{DT}}(\text{Seq}^+(\mathbb{T})))$.

This ordering relation can be lifted to the domain of ccp interpretations by defining $I_1 \preceq I_2$ *iff* $I_1(p) \preceq I_2(p)$ for every predicate p in a program. The minimal ccp interpretation $I_0^{\mathbb{CCP}}$ is such that:

$I_0^{\mathbb{CCP}}(p) = true_{\mathcal{O}} +^0 true + false_{\mathcal{O}}, p :: true_{\mathcal{O}} +^d true$.

Let $\leqslant^{\mathbb{DT}}$ be the partial order relation defined for DT semantics (definition 36 in [7]) and $I_0^{\mathbb{DT}}$ be the bottom element of $\leqslant^{\mathbb{DT}}$, we have

$\alpha_{\mathbb{CCP}}(I_0^{\mathbb{DT}}(p)) = \alpha_{\mathbb{CCP}}(true_{\mathcal{O}} + p(\alpha), \varepsilon)$
$= true_{\mathcal{O}} +^0 true + false_{\mathcal{O}}, p :: true_{\mathcal{O}} +^d true = I_0^{\mathbb{CCP}}(p)$

The following theorem shows that $\alpha_{\mathbb{CCP}}$ is a monotonic map from $\leqslant^{\mathbb{DT}}$ to $\preceq$.

Theorem 4.2 Given $\tilde{t}_1, \tilde{t}_2 \in \text{Seq}^+(\mathbb{DT})$ such that $\tilde{t}_1 \leqslant^{\mathbb{DT}} \tilde{t}_2$, we have $\alpha_{\mathbb{CCP}}(\tilde{t}_1) \preceq \alpha_{\mathbb{CCP}}(\tilde{t}_2)$.

Consider the sequence $\{(T_P^{\mathbb{DT}})^i\}$ of interpretation over $\alpha_{\mathbb{DT}}(\text{Seq}^+(\mathbb{T}))$. We know it is increasing w.r.t. relation $\leqslant^{\mathbb{DT}}$. By the strictness of $\alpha_{\mathbb{CCP}}$ and theorem 3.6 we conclude $\alpha_{\mathbb{CCP}}((T_P^{\mathbb{DT}})^i) = (T_P^{\mathbb{CCP}})^i$ for every $i \geqslant 0$. By monotonicity of $\alpha_{\mathbb{CCP}}$ we conclude $(T_P^{\mathbb{CCP}})^i \preceq (T_P^{\mathbb{CCP}})^{i+1}$ for every $i \geqslant 0$. Here $(T_P^{\mathbb{CCP}})^i$ is a shorthand for $(T_P^{\mathbb{CCP}}(I_0^{\mathbb{CCP}}))^i$. Therefore in the completion of the partial orders $\alpha_{\mathbb{CCP}}(\alpha_{\mathbb{DT}}(\text{Seq}^+(\mathbb{T})))$ there exists $\text{lub}_i(T_P^{\mathbb{CCP}})^i$. Given a program P we can define $\mathcal{F}_P^{\mathbb{CCP}} = \text{lub}_i(T_P^{\mathbb{CCP}})^i$.

Since the extension of $\alpha_{\mathbb{CPT}}$ is continuous, by proposition 2.3 and 2.4 we have

Theorem 4.3 Given a program P, the following equalities hold:

$\alpha_{\mathbb{CCP}}(\mathcal{F}_P^{\mathbb{DT}}) = \alpha_{\mathbb{CCP}}(lfp(T_P^{\mathbb{DT}})) = lfp(T_P^{\mathbb{CCP}}) = \text{lub}_i(T_P^{\mathbb{CCP}})^i = \mathcal{F}_P^{\mathbb{CCP}}$

By the correctness of DT semantics, theorem 3.6 and 4.3 we have

Theorem 4.4 Given a program *P* and a goal G, the set of correct call patterns for *G* in *P* is given by

$\{b,p \mid b \neq false$, o and sc is true and $(o +^n b + sc, p)$ belongs to $\mathcal{J}_P^{\mathbb{CCP}}[\![G]\!]\mathcal{F}_P^{\mathbb{CCP}}\}$.

Note that we refer to [7] for the definition of an observability constraint to be true or to be false.

5. Conclusion

This paper presents a constraint-based correct call pattern semantics for Prolog language, which deals with Prolog control rules and cut operator. The theory of abstract interpretation is used to prove the correctness of our semantics. Future work includes the derivation of computable abstract ccp semantics from the semantics proposed in this paper and their use in the analysis of non-trivial applications.

6. Acknowledgment

Supported by National Natural Science Foundation of China (No.60563005) and Guangxi Natural Science Foundation of China (No. 0542036).

7. References

[1] A. Bossi, M. Gabbrielli, G. Levi, M. Martelli. The s-Semantics Approach: Theory and Applications, Journal of Logic Programming, 19-20: 149-197, 1994.

[2] P. Cousot, R. Cousot, Abstract Interpretation and Applications to Logic Programs, Journal of Logic Programming 13 (23): 103-179, 1992.

[3] P. Cousot. Constructive Design of a Hierarchy of Semantics of a Transition System by Abstract Interpretation, Theoretical Computer Science, 277(1-2): 47-103, 2002.

[4] M. Gabbrielli, G. Levi, M.C. Meo. Observable Behaviors and Equivalences of Logic Programs, Information and Computation, 122(1): 1-29, 1995.

[5] G. Levi, D. Micciancio, Analysis of Pure Prolog Programs, in: Proceedings GULP-PRODE '95, 1995, pp. 521-532.

[6] G. Levi, F. Spoto, Accurate Analysis of Prolog with Cut, in: Proceeding APPIA-GULP-PRODE'96, pp. 481-492.

[7] F. Spoto, Operational and Goal-independent Denotational Semantics for Prolog with cut. The Journal of Logic Programming, 42: 1-46, 2000.

[8] F. Spoto, Giorgio Levi, Abstract Interpretation of Prolog Programs, LNCS, 1548: 455-470, 1999.

Incrementally Inferring Context-Free Grammars for Domain-Specific Languages

Faizan Javed
Department of Computer and Information Sciences
University of Alabama at Birmingham
1300 University Boulevard
Birmingham, AL 35294-1170, USA
javedf@cis.uab.edu

Marjan Mernik
Faculty of Electrical Engineering and Computer Science
University of Maribor
Smetanova 17
2000 Maribor, Slovenia
marjan.mernik@uni-mb.si

Alan Sprague, Barrett Bryant
Department of Computer and Information Sciences
University of Alabama at Birmingham
1300 University Boulevard
Birmingham, AL 35294-1170, USA
{sprague, bryant}@cis.uab.edu

Abstract

Grammatical inference (or grammar inference) has been applied to various problems in areas such as computational biology, and speech and pattern recognition but its application to the programming language problem domain has been limited. We propose a new application area for grammar inference which intends to make domain-specific language development easier and finds a second application in renovation tools for legacy software systems. We discuss the improvements made to our core incremental approach to inferring context-free grammars. The approach affords a number of advancements over our previous genetic-programming based inference system. We discuss the beam search heuristic for improved searching in the solution space of all grammars, the Minimum Description Length heuristic to direct the search towards simpler grammars, and the right-hand-side subset constructor operator.

1. Introduction

In [1], Kugel makes a case for programming computers the same way children learn – from examples. This idea is known as "programming by examples" and involves providing the computer with examples (or samples) from which a program which correctly classifies the input examples can be output by the computer without the programmer providing the computer a detailed algorithm on how to do so. To accomplish this, it would require the computer to *compute in the limit* - i.e., we would take the last output of the computer as its result without requiring it to announce when an output is its last. Our work on grammar induction, or "programming by examples", elaborates on these ideas; we provide language samples to our computer, which then uses the incremental learning algorithm to come up with a CFG which can classify or parse these language samples without overgeneralization or overspecialization.

One of the open problems in the area of domain-specific languages (DSLs) [2], also called little languages, is how to make domain-specific language development easier for domain experts not experienced in programming language design. van Deursen et al. [3] state a terse and apt definition for DSL's : "A DSL is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain". The defining characteristics of DSLs are that they are usually declarative, and smaller in size than general-purpose programming languages. A few of the possible approaches to building a DSL are to use parameterized building blocks, or by grammar induction. Grammatical inference (grammar induction or GI), a subfield of machine learning, is the process of learning of grammar from training data. Machine learning of grammars finds application in diverse domains such as software engineering, syntactic pattern recognition and computational biology. Gold's theorem [4], one of the seminal results in this area, states that it is impossible to identify any of the four classes of languages in the Chomsky hierarchy in the limit using only positive samples (unless a statistical distribution over all the positive samples is used). However, using both negative and positive samples, the Chomsky hierarchy languages can be identified in the limit.

Our research focus is on designing DSLs using Context-Free Grammar (CFG) induction techniques. In the software engineering and programming language domain, such a technique would find application in cases where legacy DSLs have been running for many years and their specifications no longer exist to assist with evolution of their implementations (e.g., as was needed to solve the Y2K problem). A survey of programming language usage in commercial and research environments has shown that more than 500 general-purpose and proprietary programming languages are in use today [5]. The Y2K-like problems not withstanding, many commercial installations use in-house DSLs and a variety of situations can arise (e.g., software company went bankrupt) where source implementations either need to be recovered or translated to a different language dialect. We expect that DSLs would usually be small enough so that GI techniques may be applied in a tractable manner to recover the underlying grammar from sample sources. While semi-automatic

techniques as in [5] can be used to recover grammar in situations where compiler sources and manuals are available, our technique is also applicable when such sources are not available, whether it be in or outside the domain of software engineering (e.g. neural networks, structured data and patterns) [6]. In such situations, the grammar needs to be extracted solely from artifacts represented as sentences/programs written in some unknown language.

The importance of CFGs is paramount in programming language syntactic design and various other domains. This, along with the fact that there has so far been no successful solution to the CFG induction problem (see section 2) makes for a promising research area to explore. In this paper, we briefly discuss our Genetic-Programming [7] approach to grammar induction in the next section, and then in section 3 we introduce some improvements to our incremental approach to inferring grammars (along with some initial results) for facilitating DSL development for domain experts not conversant with programming language development and for recovering legacy DSLs.

2. Related Work

Our research problem can be reduced to the problem of inferring CFGs from language samples. The grammar inference research community has so far been successful only in inferring regular languages. Various algorithms like RPNI[8] and EDSM[9] have been developed which can learn regular languages from positive (the set of strings belonging to the target language) and negative samples (the set of strings not belonging to the target language). In [10] a genetic-programming approach was used for inferring regular grammars and compared with the RPNI algorithm.

Successful learning of CFGs has proved to be more difficult than learning regular grammars. Despite various and differing attempts at solving the problem of CFG induction [11] [12] [13] [14] [15], there has been no one convincing solution to the problem as of now. In all the work cited so far, experiments were performed on theoretical sample languages instead of on real or even toy programming languages. Based on the work done in this area so far, we conclude that learning CFGs is still an unsolved problem.

Our previous work in this area focused on using a genetic-programming based system called GenParse to infer CFGs for DSLs. It used genetic operators and parameters, and encoded the grammar into a chromosome as a list of BNF production rules [16]. Chromosomes are evaluated using the LISA compiler generator [17] at the end of each generation by testing each grammar on a sample of positive and negative samples. The system was augmented with basic data-mining techniques such as frequent sequences [18] in order to approximately infer the sub-languages first. These improvements increased the inference capability of the GenParse system, and allowed bigger DSLs to be inferred. For more details and discussion on the GenParse system, please see [7][19]. In the area of Domain-Specific Modeling, we have applied grammar inference techniques to the problem of metamodel drift, which occurs when instance models in a repository are separated from their defining metamodel. The resulting system, called MARS[1], is a semi-automatic inference system which makes use of already existing tools along with new grammar inference algorithms to recover metamodels which correctly define the mined instance models.

3. Incrementally Inferring Grammars

In the current GenParse implementation, whenever a positive sample is not accepted (or a negative sample is accepted) by the current CFG, the induction engine takes these violating samples into account and infers a new CFG which then successfully accepts the positive sample (or rejects the negative sample). Adapting this behavior so that only the minimum number of new production rules are added to the existing CFG (which would enable the positive/negative sample to be accepted/rejected) would allow the search for a suitable CFG to be carried out in an incremental and more efficient way. So far, all attempts at using incremental grammar construction for CFG inference have only succeeded in inferring simple toy grammars [11][15]. In this section, we describe improvements to the incremental algorithm introduced in [19]. The approach makes use of positive samples only since the absence of negative examples often arises in practice. We first elaborate on the theoretical foundations of the approach, and then discuss the improvements made to the algorithm.

Input to the algorithm is a set of positive samples (or programs) $Pos = \{S_1, S_2, ..., S_N\}$, and the following auxiliary functions are used:

- *Position*($Token_i$, S_i): returns the position of $Token_i$ in sample S_i.
- *Prefix*(S_i, k): returns the first k tokens in sample S_i.
- *Suffix*(S_i, k): returns the last k tokens in sample S_i.
- *Length*(S_i): returns the length of sample S_i
- *Diff*(S_i, j, k): returns k tokens in program S_i starting from position j.

A ordering on the samples is described by the relation "simpler than" (<*) where S_i is simpler than S_{i+1} ($S_i <^* S_{i+1}$) iff the following holds:

- *All tokens in S_i are also in S_{i+1}*
- *If Position ($Token_x$, S_i) ≤ Position ($Token_y$, S_i), then Position ($Token_x$, S_{i+1}) ≤ Position ($Token_y$, S_{i+1}).*

The simplest case for incremental learning occurs when the following condition holds:

Case 1: Prefix(S_i, Length (S_i)) = Prefix (S_{i+1}, Length (S_i))

[1] Http://www.cis.uab.edu/softcom/GenParse/mars.htm

This statement denotes that the sample S_{i+1} is a continuation of sample S_i, i.e., S_{i+1} is sample S_i with some new tokens appended at the end. The initial grammar is constructed using the first sample and the algorithm described in [20]. The reason for this is to have a better grammar structure as a starting point for the incremental inference process as this can facilitate a better search process.

Case 2 describes the case when the new tokens aren't appended at the end of S_{i+1}, but rather somewhere in the middle. Case 3 is the most general of all the cases and describes the situation where the samples don't have any specific ordering on them.

Case 2:

- S_i = Prefix(S_i, k) + Suffix (S_i, j) and Length(S_i)= k + j
- S_{i+1} = Prefix(S_i, k) + Diff(S_{i+1}, k +1, Length(S_{i+1}) -k-j) + Suffix(S_i, j)

Case 3:

- S_{i+1} = Prefix(S_i, k) + Diff(S_i + 1, k+1, Length(S_{i+1})-k-j) + Suffix (S_i, j)
- S_i = Prefix (S_i, k + k_1) + Suffix (S_i, j + j_1) and Length(S_i) = k + k_1 + j + j_1 , where k_1 >0 or j_1 > 0.

Our current work focus is on designing an improved algorithm for case 1. These improvements will also be valid for the other cases, and all of the cases will be incorporated into one cohesive algorithm. Table 1 details the improved algorithm. Due to space restrictions, we only provide a short description of these advancements to the algorithm. Our first improvement is the introduction of the beam search heuristic to the algorithm. At any point in time, a set (determined by the user specified variable þ) of suitable candidate grammars is stored and used for the next iteration of the grammars. Initially, the beam contains only the initial grammar.

We next introduce the use of the Right Hand Side (RHS) subset construction operator (step 2d in Table 2). If the new increment exists as a subset of a set of RHS terminals in the grammar rules, then those terminals are replaced by a new non-terminal and a new rule (NewNT → #subset) is appended to the grammar. The entire grammar is scanned for similar subsets, all of which are replaced by the new non-terminal. Table 1 shows an example of an increment (#item #price) which, when encountered, modifies grammar (a) to grammar (b). The increment is found as a subset in rule 2 of grammar (a). The subset is then merged into a new non-terminal and 2 new rules (NewNT→ #subset) are added.

Table 1. RHS subset operator

NT1 → #stock NT2 NT3 NT2 → #item #price #qty NT2 | ε NT3 → #sales | ε (a)
NT1 → #stock NT2 NT3 NT2 → NT4 #qty NT2 | ε NT3 → #sales NT4 | ε NT4 → #item #price (b)

When only positive examples are used for inferring a grammar (as in our case), overgeneralization can be controlled by *either: i) focusing on inferring a restricted class of formal languages (which have been proved to be learnable from positive examples only), or ii) using a heuristic.* For our inference algorithm, we follow the works in [21][22] and use the Minimum Description Length (MDL) [23] heuristic to direct the search towards compact grammars (i.e. a few bits are required for encoding). The concept of MDL involves encoding a set of data, and then transmitting it to a receiver where it can be decoded. MDL compresses the grammar as well as encodes the training samples using that grammar. Thus, it offers a way to compare grammars and choose the one that is able to be compressed and encodes the examples using the least number of bits.

Our MDL heuristic is the sum of two components:

i) Grammar Description Length (GDL): the bits required to encode the grammar rules,

ii) Samples Description Length (SDL): the bits required to encode all examples using a grammar.

We would like to minimize the values of both the components. For computing the GDL, we assume that the total number of bits for encoding a rule is the sum of the bits required to encode the head of the rule, the body of the rule and the end of the rule. Since the rule size can be variable, a STOP non-terminal will be appended to each rule to indicate the end of the rule. If a grammar has U_{NT} unique non-terminals (excluding the STOP non-terminal), then the total number of bits required to encode a single non-terminal is given by: $\beta_{NT} = log\ (U_{NT} + 1)$.

For T unique terminals, the number of bits required is given by: $\beta_T = log\ (T)$.

Our approach involves dividing the rules into three subsets.

Table 2. The Incremental Learning Algorithm

1. Create initial Grammar G_1 from the first sample S_1 using the algorithm described in [20].

2. While more samples exist, do :

   a. Generate an LR(1) parser for all the grammars (G_i's) in the Beam and try to parse the next sample. If þ or more than þ grammars successfully parse the samples (where þ is the maximum number of grammars to be held in a beam), then skip steps b-e.

   b. If G_i fails to parse a sample S_j, reconstruct G_i by making the use of the increment in tokens from S_{j-1} to S_j and the following sub-steps (c,d and e).

   c. If the increment already exists as the right hand side (RHS) of a rule in the grammar with non-terminal Nx as left hand side (LHS), then append Nx to the rule and add the rule Nx → epsilon to the grammar. In some cases, this will introduce recursion in the grammar.

   d. Else, if the increment does not exist as the RHS of a rule, perform the RHS subset construction procedure on G_i, if possible. If the procedure succeeds, try the appending of the new non-terminal to existing rules as described in the latter part of sub-step e.

   e. Else, add the rule Nx → #increment to the grammar. Append a new non-terminal Nx to the first rule and then create an LR(1) parser from the grammar and then try to parse all the samples up to and including the violating sample. This process is repeated, and Nx appended to all rules in succession (except the epsilon rules) until all the samples (up to and including the violating sample) are successfully parsed. Successful grammars are stored in the beam.

   f. Calculate the MDL scores of all the grammars in the beam, and choose þ grammars with the highest scores.

3. If all samples are parsed successfully, output the grammar with the highest MDL score. Else, indicate failure and output the sample which can't be parsed.

i) Start rules: these rules have the special property that they always have the same head, and as a result the following expression gives the number of bits required for encoding one rule R of the start subset:

$$\beta_R = (NT_R + 1)\ (\log (U_{NT} + 1)) + (T_R)\ (\log T).$$

NT_R and T_R are the number of non-terminals and terminals in the rule body, respectively.

ii) Epsilon rules: these rules have a fixed length of 1, that is, the LHS non-terminal only. We don't encode the epsilon symbol, and there is no need for a STOP symbol since all the rules are of the same size. The number of bits to encode an epsilon rule is given by:

$$\beta_R = \log (U_{NT} + 1).$$

iii) All other rules : All other rules are rules which have a non-terminal on the LHS and any number of non-terminals or terminals on the right hand side. The following equation gives the number of bits to encode a rule of this type:

$$\beta_R = (NT_R + 2)\ (\log (U_{NT} + 1)) + (T_R)\ (\log T).$$

For computing SDL, we need to estimate the derivation power of the grammar. This can be done by counting all the sentences which can be generated by a grammar. However, this is not possible since a grammar usually generates an infinite language. To overcome this, we use a calculation method similar to the variability calculation of feature diagrams [24], with additional rules to handle recursion. The number of all possible different sentences is calculated by the rules in Table 3, where *NT* stands for non-terminals, *a* is a terminal symbol, ***B*** denotes a non-terminal or terminal symbol, and *Var* stands for *Variability*.

$$SDL = \log (Var (G)),\ where\ G\ is\ the\ grammar.$$

Table 3. Rules for SDL calculation

Var (NT ::= $\mathbf{B}_1 \ldots \mathbf{B}_n$)	= Var($\mathbf{B}_1$) * … * Var($\mathbf{B}_n$)	
Var (NT ::= $\mathbf{B}_1 \mid \ldots \mid \mathbf{B}_n$)	= Var($\mathbf{B}_1$) + … + Var($\mathbf{B}_n$)	
Var (NT ::= a)	= 1	(single terminal)
Var (NT ::= eps)	= 1	(empty production)
Var (NT ::= NT **B**)	= Var(**B**)	(left recursion)
Var (NT ::= **B** NT)	= Var(**B**)	(right recursion)

An experimental run of the DESK DSL is shown in Table 4. The second column shows the 3 samples used to infer the grammar, and the third column shows the incrementally constructed grammar after each iteration. The grammar inferred after sample 3 successfully parses all samples and provides a reasonable generalization of the language being inferred. In table 4, the increment during iteration 3 is (#where, #id, #oper=, #int). At this point, two suitable LR(1) grammars can be created and stored in the beam. Table 5 shows these grammars and their MDL calculations.

Table 4. Experimental Run on the DESK DSL

i	S_i	G_i
1	print %b	NT1 → #print #id
2	print %b + %b	NT1 → #print #id NT2 NT2 → ε NT2 → #oper+ #id
3	print %b + %b where %b = 20	NT1 → #print #id NT2 NT2 → #oper+ #id NT3 | ε NT3 → ε NT3 → #where #id #oper= #int

Table 5. MDL calculations for the DESK DSL

U_{NT} = 2 (excluding NT1 and STOP) , T = 6

Separator = *log (U_{NT} + 1)*

NT1 → #print #id NT2 NT3
NT2 → #oper+ #id | ε
NT3 → ε
NT3 → #where #id #oper= #int

GDL = 1 start rule + separator +
2 epsilon rules + separator + 2 other rules

MDL = GDL + SDL = 38.09 + $\log_2(4)$ = 40.096 bits

(a)

NT1 → #print #id NT2
NT2 → #oper+ #id NT3 | ε
NT3 → ε
NT3 → #where #id #oper= #int

GDL = 1 start rule + separator +
2 epsilon rules + separator + 2 other rules

MDL = GDL + SDL = 38.09 + $\log_2(3)$ = 39.68 bits

(b)

Both the grammars contain 1 start rule, 2 epsilon rules and 2 rules of other kinds, but they differ in the location of non-terminal NT3. The SDL calculation shows that grammar (a) has a higher derivation power than grammar (b) because it can generate one more sentence and is more general than grammar (b). Grammar (b) however is compressed better, as the MDL score reveals. Overall, the difference in the MDL score between the two grammars isn't vast. However for bigger experiments where a larger number of grammars would need to be scored, a bigger variability in the score can occur.

We are also currently experimenting with an optimal value for the beam size. For the DESK DSL, a beam size of 1 is suitable since it is a small DSL However, for bigger DSLs where the search space would be expansive with a bigger set of possible solution grammars, a larger beam size value might be required.

4. Future Work and Conclusion

Solving the problem of CFG inference can lead to solutions to many programming language related problems such as renovation problems and development of domain-specific languages. In this paper we describe and discuss improvements to our incremental learning algorithm. Our focus was on case 1 of the learning algorithm, and we augment that algorithm with the beam search heuristic to better search in the solution space, as well as the MDL simplicity heuristic to direct the beam search towards simpler grammars to control overgeneralization. A RHS subset constructor operator is also introduced.

Our future work involves continuing work on the incremental learning algorithm, specifically developing an all inclusive algorithm which can handle all the cases described, and further investigating the MDL heuristic for bigger DSLs. We are also experimenting with the size of the beam to observe its affects on the quality of the inferred grammars for DSLs of varying size.

Since CFGs are widely used in many other domains, the results of this work will be directly applicable in many different fields such as software system renovation, development of domain-specific languages, syntactic pattern recognition, computational biology and natural language acquisition.

5. REFERENCES

[1] Kugel, P. It's time to think outside the computational box. *Communications of the ACM,* 48(11): pp. 32-37, 2005.

[2] Mernik, M., Heering, J., and Sloane, A.M. When and How to Develop Domain-Specific Languages. *ACM Computing Surveys*, 37(4): pp. 316-344, December 2005.

[3] van Deursen, A., Klint, P., and Visser, J. Domain-Specific Languages: An Annotated Bibliography. *ACM SIGPLAN Notices*, 35(6): pp. 26-36, 2000.

[4] Gold, E. M. Language Identification in the Limit. *Information and Control*, 10(5): pp. 447-474, 1967.

[5] Lämmel, R., and Verhoef, C. Semi-automatic grammar recovery. *Software –Practice & Experience*, 31(15): pp. 1395-1438, December 2001.

[6] Mernik, M, Črepinšek, M., Kosar, T., Rebernak, D., and Žumer, V. Grammar-based systems: Definition and Examples, *Informatica*, 28(3): pp. 245-255, 2004.

[7] Črepinšek, M., Mernik, M., Bryant, B., Javed, F., and Sprague, A. Inferring Context-Free Grammars for Domain-Specific Languages. *In Proceedings of the Fifth Workshop on Language Description, Tools and Applications (LDTA 2005),* pp. 64 - 81, 2005.

[8] Oncina, J., and Garcia, P. Inferring regular languages in polynomial update time. In *Pattern Recognition and Image Analysis, volume 1 of Series in Machine Perception and Artificial Intelligence*, pp. 49 – 61. World Scientific, Singapore, 1992.

[9] Lang, K.J., Pearlmutter, B. A., and Price, R. A. Results of the Abbadingo One DFA Learning Competition and a new Evidence-Driven State Merging Algorithm, *Fourth International Colloquium on Grammatical Inference, Lecture Notes In Computer Science,* Vol. 1433, pp. 1-12, Ames, IA, Springer-Verlag, July 1998.

[10] Dupont, P. Regular Grammatical Inference from Positive and Negative Samples by Genetic Search: the GIG method. In *Proceedings of the Second International ICGI Colloquium on Grammatical Inference and Applications*, of *Lecture Notes in Artificial Intelligence*, vol. 862 , pp. 236-245, Berlin, September, Springer-Verlag, 1994.

[11] Nakamura K., and Ishiwata, T. Synthesizing Context-Free Grammars from Sample Strings based on Inductive CYK Algorithm. In *Proceedings of Grammatical Inference: Algorithms and Applications, 5th International Colloquium, ICGI 2000*, *Lecture Notes in Artificial Intelligence*, vol. 1891, pp. 186-195, Lisbon, Portugal, September 11 – 13, Springer-Verlag, 2000.

[12] Laxminarayana, J. A., and Nagaraja, G. Inference of a Subclass of Context-Free Grammars using Positive Samples. In *ECML/PKDD 2003 Workshop on Learning Context-Free Grammars*, 2003.

[13] Oates, T., Armstrong, T., Harris, J., and Nejman, M. Leveraging Lexical Semantics to Infer Context-Free Grammars. In *ECML/PKDD 2003 Workshop on Learning Context-Free Grammars*, 2003.

[14] Nakamura, K., and Matsumoto, M. Incremental learning of Context Free Grammars based on Bottom-Up Parsing and Search. *Pattern Recognition* 38(9): pp. 1384-1392, 2005.

[15] Sakakibara, Y. Learning Context-Free Grammars using Tabular Representations. *Pattern Recognition* 38(9): pp. 1272-1383, 2005.

[16] Wyard, P. Representational Issues for Context Free Grammar Induction Using Genetic Algorithm. *Proceedings of the 2nd International Colloquium on Grammatical Inference and Applications*, *Lecture Notes in Artificial Intelligence,* vol. 862, pp. 222 -235, Springer-Verlag, 1994.

[17] Mernik, M., Lenič, M., Avdičaušević, E., Žumer, V., LISA: An Interactive Environment for Programming Language Development, *Proceedings of the 11th International Conference on Compiler Construction, CC'2002, Lecture Notes in Computer Science*, vol. 2304, pp. 1 – 4, Springer-Verlag, 2002.

[18] Han, J., and Kamber, M., Data Mining: Concepts and Tehniques, *Morgan-Kaufmann Publishers*, 2001.

[19] Črepinšek, M., Mernik, M., Bryant, B., Javed, F., and Sprague, A. Context-Free Grammar Inference for Domain - Specific Languages. http://www.cis.uab.edu/softcom/GenParse/SCP05.ps, *Technical Report UABCIS-TR-2006-0301-1,* UAB, 2006.

[20] Nevill - Manning, C. G., and Witten, I. H. Compression and Explanation using Hierarchical Grammars. *The Computer Journal*, 40:103-116, 1997.

[21] Langley, P., and Stromsten, S. Learning Context-Free Grammars with a Simplicity Bias. In *Proceedings of Machine Learning: ECML 2000, 11th European Conference on Machine Learning*, *Lecture Notes in Artificial Intelligence*, vol. 1810, pp. 220-228, Barcelona, Catalonia, Spain, Springer-Verlag, 2000.

[22] Petasis, G., Paliouras, G., Spyropoulos, C. D. and Halatsis, C. eg-GRIDS: Context-Free Grammatical Inference from Positive Examples using Genetic Search". *In Proceedings of the 7th International Colloquium on Grammatical Inference (ICGI 2004), Lecture Notes in Artificial Intelligence,* vol. 3264*,* pp. 223 – 234, Springer, 2004.

[23] Rissanen, J. *Stochastic Complexity in Statistical Inquiry*, World Scientific Publishing Co., Singapore, 1989.

[24] Kang, K.C., Cohen, S.G., Hess, J.A., Novak W.E., and Peterson, A.S. Feature-Oriented Domain Analysis (FODA) Feasibility Study, *Technical Report, CMU/SEI-90-TR-21, ADA 235785*, Software Engineering Institute, CMU, Pittsburgh, PA, 1990

Supporting Connector in Programming Language*

Bo Chen[1], ZhouJun Li[2], and HuoWang Chen[1]
[1]Computer College of National University of Defense Technology,
Changsha, Hunan, P.R. China
{chenbo,hwchen}@nudt.edu.cn
[2] School of Computer Science and Engineering, Beihang University,
Beijing, P.R. China
zhoujun.li@263.net

Abstract

The importance of connector as a first-class entity in software architecture description and design has been increasingly recognized. However, how to realize connector in the implementation is still a challenge. In this paper, we argue that the most straightforward solution to realize connector is to offer explicit support in programming language to it. We have researched, designed and developed SAJ (Software Architecture-based Java) which integrates the concepts of software architecture into the java language. Both connector and component are first-class entities In SAJ, which makes the connector in the implementation able to conform to the connector in software architecture description and design. Therefore, the consistency among description, design and implementation of software architecture is able to be held. We formalize our language giving both the type system and operational semantics and prove type soundness property.

1. Introduction

Connector which explicitly describes the interactions among components is one of the important contributions of the research on the software architecture. During the phase of software architecture description, all the existing Architectural Description Languages (ADL) offer support to connector, and they can be used to describe, analyze and verify connector formally[1][2][3].However, how to realize connector in the implementation is still a challenge. At present, methods for realizing connector include: scattering the connector implementation in components among which interactions happen, adopting mechanism of operating system (such as pipeline, shared storage and etc), realizing connector with special component, and synthetically using the above methods. Most of the realization methods make connector invisible in the implementation, the traceability, intelligibility and maintainability of the software system are lost to a great extent, while the constraints and non-functional properties the ADL described, analyzed and verified are not likely to be held in the implementation.

In this paper, we argue that at present the most straightforward solution for connector realization is to offer explicit support to connector in programming language. Thereby, the connector in the implementation is conformed to that in the software architecture description and design, which can enhance the consistency, traceability, intelligibility and maintainability of the system. Meanwhile if the type system of programming language possesses soundness property, it will be in favor of the derivation of the various properties of the software system, communication integrity [4], for example. Moreover, it will be easier to hold the system properties described by ADL in the implementation. From the perspective of component-oriented programming language, the explicit support for connector is also crucial for the following reasons: Firstly, a component is a computational unit, while a connector describes interactions. Thus, component and connector have different semantics and play different roles in programming language. Secondly, the explicit support for connector makes communication, control and computation separated, which is favorable for the reusing of component and connector independently. Thirdly, a connector plays the role as composition operator in composite components. Compared with some simple composition operators, such as Mixin [5], connector has much richer semantics. Based on such an idea, the programming language SAJ has been researched, designed and developed, in which component, connector and other architecture concepts are integrated into the Java language. According to the classification of connector, the basic connectors (procedure call, pipeline, data stream and the like) were pre-defined, and the composition method for composite connector has been advanced.

The contribution of this paper is twofold. On one hand, we propose a new component-oriented programming language SAJ in which connector is a first-class entity as well as component. Conformance between architectural specification and implementation can be enforced through using SAJ to construct software system. On the other hand, the component model and connector model underlying SAJ provide the basis for constructing complex component and complex connector.

The rest of this paper is organized as follows. The SAJ language is introduced in section 2, where the component

* The work reported in this paper has been funded by the National Natural Science under Grant Nos. 90104026, 60473057, 90604007.

model and connector model are discussed and the syntax, semantics and type system of SAJ are presented in detail. In section 3, we compare our approach with related work. Finally, in section 4, we conclude and consider further work.

2. SAJ Language

Developing SAJ with the integrating concepts of software architecture into the Java language aims at naturally mapping the architecture model to the implementation, and making SAJ a new component-oriented programming language. In SAJ, both connector and component are first-class entities, and SAJ type system possesses soundness property. The following is an introduction of the SAJ. Firstly comes the introduction of the component model and the connector model in SAJ.

2.1 Component Model and Connector Model

Component represents the computational unit and data store of a system. Each component may have multiple interfaces called ports. Each port represents a logical point of interactions between component and external world. Port declares two kinds of interfaces: provided interfaces and required interfaces. Required interfaces clearly declared can make the dependencies explicit, reduce coupling between components and promoting understanding of component in isolation. Since the concept of port is favorable for information hiding and interface segregation principle, we provide support for the concept of port in SAJ, which is different from other component-oriented programming language.

Connector represents interactions among components. Each connector has a group of roles and interaction protocols. A role, which represents the external behavior that the component participating in the interaction should have, is the abstraction of the component that is capable of playing the role: any component that can play the role (strictly speaking, play the role through port) is able to connect to the connector and participate in the interaction. Interaction protocol, which describes how roles collaborate, localizes information about interactions: the interactions among components happen in the connector. In SAJ, the role is the interface in general meaning, and the interactions are implemented by the isolated codes in the connector.

The concept of active connector [6] or exogenous connector [7] is supported in our connector model. Connector describes collaborations among roles, and component plays the corresponding role through port, which has substantively formed the active/passive relationship between connector and component. We are able to encapsulate control in the active connector and computation in the component. Through separating control and computation with connector, the problem of mixing control and computation in the component is solved. Thereby, the coupling between components is reduced greatly, and simultaneously reusing component and connector can be realized. Of course, when it is not suitable to use active connector, we use the communication and data transfer functions of connector. Connector is responsible for transparently delivering the services component required to the corresponding component port and returning the result. To be exact, we encapsulate control and computation in the component and encapsulate communication in the connector.

SAJ provides support for the concepts of composite component and composite connector. Composite component is different from basic component in that it has internal architecture, and that the services it provides for the external world are accomplished through the collaborations among the sub-components connected by the connector. Although composite connector has internal architecture, which is the same as composite component, the inner of composite connector is composed of methods, component instances and connector instances, which are used to implement crosscutting feature services, while most of these services, such as security, log and so on, are common services which are orthogonal to business logic [8].

We adopt a method of inheriting and incremental composition in constructing composite connector. We believe that however complex the interaction among components is, the interaction always belongs to a certain type as a whole. After the user-defined connector inherits the pre-defined connector (such as ConProcCall, ConPipeLine) of SAJ in such a type, various common services are added to the user-defined connector in order to implement complex interaction and communication protocol. Since these common services possess the property of universal applicability, generic programming and aspect-oriented programming are adopted in the implementation of the common services. The services we have implemented include interface adaptation, data-transform and the like.

2.2 Formalization

The core of SAJ is based on RelJ[9], a "middleweight" subset of Java. The reason for our choosing RelJ as our basis from a number of calculi proposed for java is that it is big enough to include the essential imperative features of Java and yet small enough that formal proofs are still feasible (We should point it out that many definitions in the following sections are directly derived from RelJ). On the basis of RelJ, component, connector, port, role (interface) and other architecture concepts are added.

2.2.1 Syntax

The syntax of SAJ is presented in Figure 1. The meta-variables c ranges over component class names; d ranges over connector class names; r, r', R, P and I range over interface names, where r and r' are typical used to represent roles ; n ranges over component class names , connector class names and interface names; z ranges over port names; t ranges over types; f ranges over fields; m ranges over the set of method names; and x ranges over the set of variable names; u and v range over values; l ranges

over addresses in the heap; e ranges over expressions; s ranges over statements; as a shorthand, an overbar is used to represent a sequence.

$p\in$ Program	::=$\overline{\text{com}}$; $\overline{\text{con}}$
com	::=Component Class c extend c' $\{\overline{t\ f},\overline{\text{M}},\overline{z}\}$
con	::=Connector Class d extend $d'(r,r')\{\overline{t\ f},\overline{\text{M}}\}$
Primtype	::=int | boolean|void|....
$n\in$ RefType	::=$c\|d\|I$ (*interface-name*)
$t\in$Type	::= Primtype| RefType
M	::=$t\ m(\overline{t\ x})\ mb$
$mb\in$MethBody	::=$\{s$ *return* $e;\}$
$mt\in$Methtype	::=$\overline{t}\to t$
r,r',R,P,I	::=$\{\overline{m:mt}\}$
z	::={Required R,Provided P}
$u,v\in$Value	::=pv|null|l
$pv\in$PrimValue	::=*intvalue* | *boolvalue* |....
var	::=$x\,|\,e.f$
$e\in$ Expression	::=v| *var*|*se*
$se\in$StatementExp	::=*var*=$e|e.m(\overline{e})|e.z.m(\overline{e})$|new $c(\,)$|new $d(r,r')$
$s\in$Statement	::=$\varepsilon\ |\ se;s_1$ | if (e) then $\{s_1\}$ else$\{s_2\};s_3$

Figure 1. The syntax of SAJ

$C\in$ComponentTable	: CompontClassName $\to$ CompontClassName $\times$ FieldMap $\times$ MethMap $\times$ PortMap
$D\in$ConnectorTable	: ConnectorClassName $\to$ ConnectorClassName $\times$ RoleType $\times$ RoleType $\times$ FieldMap $\times$ MethMap
$F\in$FieldMap	: FldName $\to$ Type
$M\in$MethMap	: MethName $\to$ ParameterList $\times$ LocalImap $\times$ TypeList $\times$ Type $\times$ MethBody
$Z\in$PortMap	: PortName $\to$ ReqInterface $\times$ ProInterface
$L\in$LocalMap	: VarName $\to$ Type

Figure 2. Signatures of component and connector class tables

As usual ,for such language formalizations, we assume that given a SAJ program P, the component class and connector class declarations give rise to component and connector tables that are denoted by C_p and D_p, respectively. A component (connector) table is a map from component (connector) class names to their definitions.

A component class definition is a tuple, (c',F,M,Z), where c' is the superclass; F is a map from field names to field types; and M is a map from method names to method definitions. Z is a map from port names to port definitions. Method definitions are tuples $(\overline{t\ y},L,t',mb)$, where y is the parameter; t is the parameter type; L is a map from local variable names to their types; t' is the return type; and mb is the method body. For brevity, we write F_c, M_c and Z_c for the field, method and port definition maps of class c.

A connector class definition is a tuple, (d',r,r',F,M), where d' is the superclass, r,r' represent the roles in connector (to be simple, we assume that there are two roles in each connector and that each connector connects two components, role is played by the pair (c,z) in which c represents the type of the component participating in the interaction and z represents component port that connected to the connector); F,M are the same to the above. Signatures for these maps are presented in Figure 2.

Values represent irreducible computational results, including addresses and primitive values (*e.g.* literals such as true, false, 1, *etc*). Expressions include component instance creation expressions, connector instance creation expressions, filed accesses, method invocations. Only conditional statements, assignments, statement sequences are considered for compactness.

2.2.2 Semantics

We specify reduction rules for a small-step operational semantics. The meta-variables used in the semantics range over addresses, instances and stores are given below.

$o\in$ Instance

σ : Address $\to$ Instance

ρ : Address $\times$ Port $\times$ Address $\times$ Port $\to$ Address

λ : VarName $\to$ Value

Instances, ranged over by o, are either component instances or connector instances. Component instance is written as an annotated pair $<<c\parallel f_1:v_1,f_2:v_2,....f_n:v_n>>$, where c represents the dynamic type of component instance, f_i and v_i represent the name and the value of field respectively.

Connector instance is written as an annotated 6-tuple $<<d,l_1,z_1,l_2,z_2\parallel f_1:v_1,f_2:v_2....f_n:v_n>>$, d represents the dynamic type of connector instance; l_1,l_2 represent the addresses of the component instances participating in the connection, while z_1,z_2 represent the ports through which the component instances do.

A store σ, is a map from addresses to instances (component instances and connector instances), while local variables are given values by a locals store λ. A connection relationship heap ρ maps connection relationship tuples to addresses.

A program configuration *config* $<\Gamma,\sigma,\rho,\lambda,s>$ in the semantics is a 5-tuple of typing environment Γ (which is a finite mapping from variables to types), heap σ, connection relationship heap ρ, locals map λ, and a statement s. An error configuration is a configuration $<\Gamma,\sigma,\rho,\lambda,Error>$, with an error in a statement sequence. Program execution is described with $<\Gamma,\sigma,\rho,\lambda,s>\rightsquigarrow<\Gamma',\sigma',\rho',\lambda',s'>$.

Due to space limitations, only the more interesting reduction rules in Figure 3 are discussed here.

OSNewCon gives the semantics to the creation of a connector instance. The OSNewCon rule reduces a new expression to a fresh location. The heap is updated at that location to refer to a new connector instance with its fields set to the initial values and the connection relationship heap is added an entry which represents the map from the connection relationship tuple newly created to address of the new connector instance.

The OSCallReqMeth rule gives the semantics for calling required method in port. Only the component instance itself is

able to call the required method in the connected port which is then delivered by the connector instance to the corresponding port of another component instance.

The OSCallProMeth rule is simple. The semantics for the method invocation of component instance is given by OSCallComMeth. OSCallComMeth determines the correct method body to invoke. Then, the method invocation is replaced with the appropriate method body. In the method body, all occurrences of the formal method parameters and *this* are replaced with the actual arguments and the receiver, respectively.

$$\text{(OSNewCon)}\quad <\Gamma,\sigma_1,\rho_1,\lambda,New\,d(l_1,z_1,l_2,z_2)> \rightsquigarrow <\Gamma,\sigma_2,\rho_2,\lambda,l>$$
$$where\quad \sigma_2=\sigma_1[l\mapsto ll d,l_1,z_1,l_2,z_2\mid f_i:Inital(FD_c(f_i))\gg]$$
$$\rho_2=\rho_1[(l_1,z_1,l_2,z_2)\mapsto l]\quad l\notin dom(\sigma_1)$$
$$\text{(OSCallReqMeth)}\quad <\Gamma,\sigma,\rho,\lambda,l_1.z_1.m(\bar{u})> \rightsquigarrow <\Gamma,\sigma,\rho,\lambda,l_2.z_2.m(\bar{u})>$$
$$where\ \sigma(l_1)=ll c\|...\gg\quad m\in\{c.z_1.R_1\mid z_1\in dom(Z_c)\}$$
$$\rho(l_1,z_1,l_2,z_2)=l\Rightarrow l\neq Null\wedge l_2\neq Null$$
$$\sigma(l_2)=ll c'\|...\gg\ m\in\{c'.z_2.P_2\mid z_2\in dom(Z_{c'})\}$$
$$\text{(OSCallProMeth)}\quad <\Gamma,\sigma,\rho,\lambda,l.z_i.m(\bar{u})> \rightsquigarrow <\Gamma,\sigma,\rho,\lambda,l.m(\bar{u})>$$
$$where\ \sigma(l)=ll c\|...\gg\quad m\in\{c.z_i.P_i\mid z_i\in dom(Z_c)\}$$
$$\text{(OSCallConMeth)}\quad <\Gamma_1,\sigma,\rho,\lambda_1,l.m(\bar{u})> \rightsquigarrow <\Gamma_2,\sigma,\rho,\lambda_2,\{s_3\ return\ e_3;\}>$$
$$where$$
$$\sigma(l)=ll d,l_1,z_1,l_2,z_2\|...\gg\ MD_d(m)=(\overline{t\,y},L,t',s_1\ return\ e_1)$$
$$dom(L)=\bar{x}\qquad \bar{u}',x'_{this},\bar{x}'\notin dom(\lambda_1)$$
$$\Gamma_2=\Gamma_1[\overline{u'\mapsto t}][x'_{this}\mapsto d][\overline{x'\mapsto L(x)}]$$
$$\lambda_2=\lambda_1[\overline{u'\mapsto u}][x'_{this}\mapsto l][\overline{x'\mapsto Inital(L(x))}]$$
$$s_2=s_1[\overline{u'/y}][x'_{this}/this][\overline{x'/x}]$$
$$e_2=e_1[\overline{u'/y}][x'_{this}/this][\overline{x'/x}]$$
$$s_3=s_2[l_1.z_1/x'_{this}.r][l_2.z_2/x'_{this}.r']$$
$$e_3=e_2[l_1.z_1/x'_{this}.r][l_2.z_2/x'_{this}.r']$$

Figure 3. Some interesting reduction rules

Finally, we consider method invocations in connector whose methods describe interactions among components. The rule OSCallConMeth is the same as the rule OSCallComMeth except that the roles of connector instance are replaced with actual component instances (when the component instance plays a role through a certain port, only its provided methods at this port can be invoked in connector methods). Here, only the OSCallConMeth rule is given.

2.2.3 Type System

We provide **Component** for the root of the component class hierarchy, and **Connector** as its counterpart in the connector class hierarchy. The subtyping rules are omitted here because they are similar to that of RelJ. We type expressions and statements in the presence of a typing environment Γ and only some typing rules deserving particular attention are discussed here.

The connector instance creation typing rule TSNewCon verifies that when two component instances are connected with a connector instance, the type of the provided interface in connected port of each component instance should be the subtype of role $r's$ type and $r''s$ type respectively, while the type of the required interface should be the supertype of role $r''s$ type and $r's$ type respectively.

$$\text{(TSNewCon)}$$
$$\frac{d\in dom(D_p)\quad D_p(d)=(d',r,r',F,M)\quad \Gamma\vdash e_1:c_1\quad \Gamma\vdash e_2:c_2\quad \Gamma\vdash c_1.z_1.P_1\le r\quad \Gamma\vdash c_2.z_2.P_2\le r'\quad \Gamma\vdash r'\le c_1.z_1.R_1\quad \Gamma\vdash r\le c_2.z_2.R_2}{\Gamma\vdash New\ d(e_1.z_1,c_2,z_2):d}$$

The rule TSCallMeth looks up the invoked method's type using *mtype* function, and verifies that the actual argument types are subtypes of the method's parameter types. If the invocation is through a port and the invoked method belongs to required interface, the rule TSCallPortMeth verifies that the instance expression must be *this* .

$$\text{(TSCallMeth)}\quad \frac{\Gamma\vdash e:n\quad \Gamma\vdash \bar{e}:\bar{t}'\quad mtype(m,n)=\bar{t}\to t\quad \bar{t}'\le\bar{t}}{\Gamma\vdash e.m(\bar{e}):t}$$

$$\text{(TSCallPortMeth)}\quad \frac{\Gamma\vdash e:c\quad \Gamma\vdash\bar{e}:\bar{t}'\quad c\in dom(C_p)\quad z\in dom(Z_c)\quad m\in c.z.R\Rightarrow e=this\quad mtype(m,c,z)=\bar{t}\to t\quad \bar{t}'\le\bar{t}}{\Gamma\vdash e.z.m(\bar{e}):t}$$

Next, the rules for well-formed component class definition and connector class definition are given in Figure 4. Firstly, ports are checked in the presence of their enclosing component class and then fields and methods are checked in the presence of their enclosing component class or connector class (Only TSPort rule is presented here).

$$\text{(TSPort)}\quad \frac{C_p(c)=(c',F,M,Z)\quad z\in dom(Z_c)\quad z.R\cap z.P=\emptyset\quad z.P\subseteq dom(MD_c)\quad c'\neq\textbf{Component}\Rightarrow z\in dom(Z_{c'})}{P,c\vdash z}$$

$$\text{(TSCom)}\quad \frac{C_p(c)=(c'\neq c,F,M,Z)\quad P\vdash c'\quad \forall f\in dom(F)\Rightarrow\ P,c\vdash f\quad \forall m\in dom(M)\Rightarrow P,c\vdash m\quad \forall z\in dom(Z)\Rightarrow\ P,c\vdash z}{P\vdash c}$$

$$\text{(TSCon)}\quad \frac{D_p(d)=(d'\neq d,r,r',F,M)\quad P\vdash d'\quad r,r'\in ValidType\quad \forall f\in dom(F)\Rightarrow\ P,d\vdash f\quad \forall m\in dom(M)\Rightarrow P,d\vdash m}{P\vdash d}$$

$$\text{(TSProg)}\quad \frac{\forall n\in dom(C_p)\cup dom(D_p)\Rightarrow P\vdash n}{\vdash P}$$

Figure 4. Port, component class, connector class and program typing

TSPort checks that a port is well-formed by verifying that only subclass of **Component** can define new ports and that the intersection of the required interface and the provided interface at the same port is empty.

TSCom specifies that a component class type is well-formed if its superclass is well-formed, its ports are well-formed, and if all of its methods and fields are well-typed. TSCon imposes many of the same restrictions as

TSCom except without checking ports.

TSProg specifies that a program is well-formed if all of its component classes and connector classes are well-formed.

Finally, the rules for well-formed heap σ, connection relationship heap ρ and the locals map λ are given. For heap σ, we ensure all the runtime instances (component instance and connector instance) stored in it are well formed. The rule for well-formed component instance is the same as WFFOBJECT2 in RelJ. The rule for well-formed connector instance WFConInst is presented bellow:

(WFConInst)

$$\frac{\begin{array}{l} D_p(d) = (d', r, r', F, M) \qquad \forall f \in dom(FD_d) \Rightarrow P, \sigma, o \vdash f_{\Diamond fld} \\ \sigma(l_1) = << c \parallel ... >> \Rightarrow P, \sigma \vdash << c \parallel ... >>_{\Diamond inst} \\ \sigma(l_2) = << c' \parallel ... >> \Rightarrow P, \sigma \vdash << c' \parallel ... >>_{\Diamond inst} \\ \vdash dynType(\sigma(l_1)).z_1.P \le r \qquad \vdash dynType(\sigma(l_2)).z_2.P \le r' \end{array}}{P, \sigma \vdash << d, l_1, z_1, l_2, z_2 \parallel f_1 : v_1 .... f_n : v_n >>_{\Diamond inst}}$$

WFConInst ensure that(1)all fields that the connector instance has, including those inherited from its superclasses, are well-formed ;(2)component instances connected by the connector instance are well-formed;(3)the type of the provided interface at connected port of each component instance is subtype of the type of role that component instance plays.

We then map the conditions for well-formed instances and local variables over the heap σ, the connection relationship heap ρ, and the locals map λ:

(WFHeap)

$$\frac{\forall l \in dom(\sigma) \Rightarrow P, \sigma \vdash \sigma(l)_{\Diamond inst}}{P \vdash \sigma_{\Diamond heap}}$$

(WFLocals)

$$\frac{\forall x \in dom(\Gamma) \Rightarrow P, \sigma, \Gamma \vdash \lambda(x) : \Gamma(x)}{P, \sigma, \Gamma \vdash \lambda_{\Diamond locals}}$$

(WFConHeap)

$$\frac{\forall (l_1, z_1, l_2, z_2) \in dom(\rho) \Rightarrow P, \sigma \vdash \sigma(l_1)_{\Diamond inst} \wedge P, \sigma \vdash \sigma(l_2)_{\Diamond inst} \wedge P, \sigma \vdash \sigma(\rho(l_1, z_1, l_2, z_2))_{\Diamond inst}}{P, \sigma \vdash \rho_{\Diamond conheap}}$$

We consider a program configuration $< \Gamma, \sigma, \rho, \lambda, s >$ to be well-formed when σ, ρ and λ are well-formed, and where s is type-correct.

2.2.4 Soundness

We prove type soundness using standard theorems of type preservation and progress, and the proofs for the two theorems are left for another paper due to space limitations.

Theorem 1 (Type Preservation). *In a well-typed program, P, where configuration* $<\Gamma,\sigma,\rho,\lambda,s>$ *executes to a new configuration* $< \Gamma', \sigma', \rho', \lambda', s' >$, *that configuration will be well-formed. Furthermore, all instances in* σ *retain their dynamic type in* σ'.

Theorem 2 (Progress). *For all well-typed programs, P, all well-formed configurations* $<\Gamma,\sigma,\rho,\lambda,s>$ *execute to either an error configuration* $< \Gamma', \sigma', \rho', \lambda', Error >$ *or a new statement configuration* $< \Gamma', \sigma', \rho', \lambda', s' >$.

Together, progress and type preservation imply that well-typed programs do not go wrong.

3. Related Work

In the research on software architecture, component and connector are two basic concepts. For a long time people have been focusing on component structure, component interface and some other aspects of component, the research on connector was completely ignored. Nowadays, people are becoming more and more concerned about that for a large-scale complex system, functional properties and non-functional properties of a system usually depend on the interactions among components. The research on connector has been advanced rapidly in recent years, especially the research on how to realize connector has already been a hot topic.

Spitznagel and Garlan proposed a compositional approach using connector transformation [10]. With connector transformation, basic connectors (procedure call, pipeline, for example) were transformed into new connectors. Connector was defined as a 6-tuple composed of application-level code that appeared within a component, low level infrastructure services provided by, for example, the operating system, formal specification describing the connector's proper behavior and so on. Through the modification of one or more parts of an existing connector with transformation, a new connector was derived. The main problem of the connector transformation lied in its feasibility. The codes of connector were dispersed throughout the system, while the transformation needed to modify the application-level code, even replace the service provided by low level infrastructure. What is more, there was no formalization proving on the validity of connector transformation.

Aldrich proposed an approach of offering support for user-defined connector in programming language [11]. On the basis of component-oriented programming language ArchJava, the reflection mechanism was used to fulfill connector abstraction. User-defined connectors were derived from the pre-defined connectors offered by ArchJava reflection library, while the "Invoke" function of connector that described the dynamic semantics of interactions was overridden. Furthermore, it has been argued that offering support for connector abstraction in programming language had the flexibility which could greatly improve the reuse of connector, because almost all kinds of connectors enumerated in [12] could be realized with the connector abstraction of ArchJava. The main problem of this approach lied in that it was difficult to guarantee the type soundness in the implementation of connector abstraction with reflection mechanism, and that it had not advanced how to construct composite connector.

Some other approaches of realizing connector exist [13] [14]. In contrast to these works, our approach is more formal. We believe that such a formal, mathematical approach is essential to set a solid foundation for researchers and users. At the same time, our approach is more flexible and more direct because programming language has direct influence on the implementation of software system. Compared with

Aldrich's work, we provide connector abstraction directly instead of using reflection mechanism. Furthermore, we can guarantee the type soundness in the implementation of connector abstraction.

4. Conclusion

In this paper, we argue that connector is worthy of explicit support in programming language from the perspective of both software architecture and component-oriented programming language. We have researched, designed and developed SAJ which integrates concepts of software architecture into the Java language .Connector is a first-class entity in SAJ, as well as component, which makes the connector in implementation able to conform to the connector in software architecture description and design. Thereby, the consistency among description, design and implementation of software architecture is able to be held.

Our work on SAJ is at a preliminary stage, however. Future tasks include further perfecting our component model and connector model and researching the technique of automatic connector creation. For perfecting the component model and connector model, we are particularly interested in enriching component and connector specification and offering composition methods or appropriate composition operators to allow construction of complex connector. Concerning automatic connector creation, we want to propose an approach of creating connector automatically basing on the enriched component and connector specification.

Reference

[1] Allen R., Garlan D., "A formal basis for architectural connection", ACM Trans. on Software Engineering and Methodology, 1997, 6(3): 213~249.

[2] David C. Luckham, John J. Kenney, Larry M. Augustin, James Vera, Doug Bryan, Walter Mann, "Specification and Analysis of System Architecture Using Rapide", IEEE Trans. on Software Engineering, 1995, 21(4), 336~355.

[3] Oreizy P, David S. Rosenblum, Richard N. Taylor, "On the Role of Connectors in Modeling and Implementing software Architectures", Technical Report UCI-ICS-98-04, University of California, Irvine, 1998

[4] Aldrich J., Chambers C. and Notkin D., "Architectural Reasoning in ArchJava", European Conference on Object-Oriented Programming, Málaga, Spain, June 2002.

[5] Vugranam C. Sreedhar, "Mixin' Up Components", International Conference on Software Engineering, Orlando, Florida, May 2002.

[6] ZHANG Jiachen, FENG Tie, CHEN Wei, JIN Chunzhao, "Active-Connector-Based Software Architecture and Its Description Method", Journal of Software, 2000, 11 (8): 1047~1052 (in Chinese with English abstract).

[7] Kung-Kiu Lau, Perla Velasco Elizondo, ZhengWang, "Exogenous Connectors for Software Components", Eighth International SIGSOFT Symposium on Component-based Software Engineering, 2005

[8] MEI Hong, CHEN Feng, FENG Yao-Dong, YANG Jie, "ABC: An Architecture Based, Component Oriented Approach to software development", Journal of Software, 2003, 14 (04):721~732(in Chinese with English abstract).

[9] Bierman G., Wren A., "First-class relationships in an object-oriented language", European Conference on Object-Oriented Programming, Glasgow, Scotland, 2005.

[10] Spitznagel B., Garlan D., "A Compositional Approach for Constructing Connectors", the Working IEEE/IFIP Conference on Software Architecture (WICSA'01*)*, Royal Netherlands Academy of Arts and Sciences Amsterdam, The Netherlands, 2001.

[11] Aldrich J., Sazawal V., Chambers C., Notkin D., "Language Support for Connector Abstractions", European Conference on Object-Oriented Programming, Darmstadt, Germany, 2003.

[12] Nikunj R. Mehta, Nenad Medvidovic, and Sandeep Phadke, "Towards a Taxonomy of Software Connectors", International Conference on Software Engineering, Limerick, Ireland, June 2000.

[13] Cheoljoo Jeong and Sangduk Lee. "Implementing software connectors through first-class methods", Proceedings of the IEEE conference on system, man, and cybernetics, Denver, United States, 2000.

[14] Oussalah M., Smeda A., Khammaci T., "An Explicit Definition of Connectors for Component-Based Software Architecture", IEEE International Conference on the Engineering of Computer Based Systems,Brno, Czech Republic, 2004.

TaxTOOLJ: A Tool to Catalog Java Classes

Djuradj Babich, Kayan Chiu and Peter J. Clarke
School of Computing and Information Sciences
Florida International University
Miami, FL 33199, USA
email: {dbabi001, kchiu001, clarkep}@cis.fiu.edu

Abstract

We present the design and implementation of TaxTOOLJ, Taxonomy Tool for Object-Oriented Language Java. TaxTOOLJ is a reverse engineering tool that catalogs the classes of a Java software application using the taxonomy of object oriented classes. TaxTOOLJ is entirely implemented in Java, which makes it very reliable, portable, and maintainable. It represents a scalable tool that performs efficiently during introspection of large scale Java applications and enables support for program understanding, software metrics, testing, reengineering and reuse of the systems.

1 Introduction

The complexity of modern software systems requires the use of several different metrics to aid in the understanding of the software [8, 10, 12]. Many of the OO design metrics are used to study the complexity of software that focus on single class characteristics. Clarke et al. [4, 6] presented the first class abstraction that focuses on how class characteristics are combined within a class. These class characteristics include features such as: types, accessibility, shared class features, polymorphism, dynamic binding, deferred features, exception handling, and concurrency. Crowther et al. [7] extended the taxonomy of Java classes to describe all possible combinations of the characteristics for Java classes.

In this paper we present the details of *TaxTOOLJ - a Taxonomy Tool for Object-Oriented Language Java*, a reverse engineering tool that catalogs the classes of a Java application using the taxonomy of OO classes described in [7]. TaxTOOLJ is the first tool that catalogs Java classes presenting a summary of the class characteristics for each class in a Java application. The tool also summarizes the class characteristics for the attributes and routines of each class. We demonstrate the scalability of TaxTOOLJ by presenting a summary of the results obtained when JDK 1.5.05 [11], containing 17,343 classes, is reverse engineered.

The combination of class characteristics generated by TaxTOOLJ can be used to research several issues, including:

- The relationship between the different combinations of class characteristics and the testability of Java software [2, 15].
- Further automating implementation-based testing of classes in large Java applications [4, 5].
- The identification of how the combinations of characteristics of classes and their features (attributes and routines) change in large Java applications [6].

Knowing the combinations of characteristics for a class can be used to determine if one class will be easier to test than another class [2], based on the test histories for the different class groups. Furthermore, the tool may be used to determine what patterns of change the systems may undergo during maintenance. Such information may be valuable during regression testing with selective retest techniques.

In the next section, background material for the taxonomy of Java classes is presented. Section 3 presents the algorithm to catalog classes, Section 4 the design and implementation of TaxTOOLJ and in Section 5 we present a case study. We then discuss related work in Section 6 and conclude the paper with Section 7.

2 Taxonomy

Crowther et al. [7] extended the taxonomy of OO classes by Clarke et al. [4, 6] for Java classes. The taxonomy of OO classes is used to classify a Java class C into a group based on the dependencies C has with other types in a program. The dependencies of C with

Descriptors		
Nomenclature	*Attributes*	*Routines*
(Public)	(Transient)	(Final)
(Final)	(Volatile)	(Native)
(Has-Nested)	New	(Generic)
(Has-Inner)	Recursive	New
(Interface)	Concurrent	Recursive
(Implements)	Polymorphic	Redefined
(Serializable)	Private	Concurrent
Generic	Protected	Synchronized
Concurrent	Public	Exception-R
Abstract	Constant	Exception-H
Inheritance-free	Static	Has-Polymorphic
Parent	-	Non-Virtual
External Child	-	Virtual
Internal Child	-	Deferred
-	-	Private
-	-	Protected
-	-	Public
		Static

(a)

Type Families	
NA	no type
P	primitive type
P*	reference to P
U	user-defined type
U*	reference to U
L	standard library
L*	reference to L
A	any type (generics)
A*	reference to A
$m < n >$	parameterized type
$m < n >$*	reference to parameterized type
where $m \in$ {U, L} and n is any combination of {P, P*, U, U*, L, L*, A, A*}	

(b)

Figure 1. Entities used in a cataloged entry for a Java class. (a) Descriptors, and (b) type families. Add-on descriptors peculiar to the Java language are shown in parentheses.

other types are realized through declarations and definitions of C's features and C's role in an inheritance hierarchy. The artifact generated when a Java class is cataloged using the taxonomy is a *cataloged entry*. The properties of the taxonomy include [7]: (1) *domain coverage* - provides a means of cataloging any Java class, (2) *mutual exclusion* - partitions the set of all Java classes into mutually exclusive groups (taxa), and (3) *unambiguity* - the strings used to represent groups of Java classes (attributes and routines) are specified in an unambiguous manner.

A cataloged entry is defined as a 5-tuple consisting of: (1) *Class Name* - fully qualified name of a class (2) *Nomenclature Component* - the group (or taxon) containing the class, (3) *Attributes Component* - a list of entries representing the subgroups of attributes, (4) *Routines Component* - a list of entries representing the routines, and (5) *Feature Classification Component* - a list summarizing the inherited features of the class. Each component entry consists of two parts: (1) a *modifier* - describing the properties of the class and its features (attributes and routines), and (2) the *type families* - types associated with the class.

A modifier consists of a list of core and add-on descriptors representing the class characteristics. The core descriptors represent class characteristics found in most OO languages and the add-on descriptors represent characteristics peculiar to the Java language [1]. The tables in Figure 1 show the list of descriptors and the type families. Figure 1(a) lists both the core and add-on descriptors. Add-on descriptors within the table are enclosed in parenthesis. Columns 1, 2, and 3 of the table in Figure 1(a) show the descriptors used in the modifier part of the component entries in the Nomenclature, Attributes and Routines components respectively. Type families used in the component entries are presented in the table shown in Figure 1(b). A detailed explanation of the descriptors and type families are provided in [7].

3 Algorithm to Catalog Classes

In this section, we describe the algorithms used in TaxTOOLJ to catalog the characteristics of a class. The algorithms include: (1) class characteristics extraction, and (2) cataloged entry generation.

3.1 Class Characteristics Extraction

The algorithm class_Characteristics shown in Figure 2 is used to extract the class characteristics from each class in a Java application. The algorithm extracts the class characteristics in two stages. The first stage of the algorithm, shown in lines 5 through 25 of Figure 2, collects all the class characteristics excluding information found within method implementations. This stage of the algorithm recursively descends the package structure, extracting the attribute and routine information accessible by the Java Reflection facility. In addition, class inheritance hierarchies are processed to extract information on recursive and redefined members, lines 21 through 25.

The second stage starts at line 26 and generates an abstract syntax tree (AST) for each class being analyzed, in order to extract method-implementation dependent information. The AST is generated using the ASTParser class in the Java Development Tooling (JDT) package from the Eclipse SDK[14].

```
1: allClassInfos class_Characteristics (global)
   /*Input: global - application's global package i.e.
                    base directory
     Output: allClassInfos - a list of all objects containing
                    OO features for each class. */
2: allClassInfos ← ∅
3: childHashMap ← ∅
4: packageStack.push(global)
5: while packageStack.size ≠ 0 do
6:    currentPackage ← packageStack.pop()
7:    for all class ∈ currentPackage do
8:       create classInfo /*classInfo stores all the characteristics
         for the class, its attributes and routines */
9:       if class.parent ≠ null then
10:         classInfo.parent ← class.parent
11:         childHashMap.add(class.parent, class)
12:      end if
13:      create reflectObj for class
14:      classInfo.info ← reflectObj.characteristics
         /*The reflectObj.features uses reflection to get the
         characteristics of the class, its attributes and routines*/
15:      allClassInfos.add(classInfo)
16:   end for
17:   for all package ∈ currentPackage do
18:      packageStack.push(package)
19:   end for
20: end while
21: for all pair(parent, child) ∈ childHashMap do
22:    classInfo ← allClassInfos.get(parent)
23:    classInfo.chidren.add(child)
24:    allClassInfos.add(classInfo)
25: end for
26: if fullCatalog = true then
27:    for all classInfo ∈ allClassInfos do
28:       classAST ← AST.parse(classInfo.getName)
29:       for all method ∈ classInfo do
30:          localInfo ← classAST.method.getLocalInfo()
           /* astParse.method.getLocalInfo produces
           information on the local declarations and exception
           handling constructs.*/
31:          classInfo.method.update(localInfo)
32:       end for
33:    end for
34: end if
35: return allClassInfos
36: end clouseauJ_API
```

Figure 2. The algorithm for class_Characteristics.

The running time of class_Characteristics algorithm on an application consisting of n .class files is $O(n * r * l)$ where r represents the average number of routines per .class file, and l is the average number of lines per routine.

3.2 Cataloged Entry Generation

Figure 3 shows the algorithm class_Cataloger that takes as input the list of objects containing the class characteristics (allClassInfos) generated by class_Characteristics, Figure 2, and returns a list of cataloged entries (allTaxEntries). After initializing the data structures in lines 2 and 3, the algorithm creates a dependency list (dependencyList) of the classInfo objects base on inheritance relationships, line 4. The loop, lines 5 to 17, iterates through the dependency list creating a cataloged entry taxEntry for each class in the dependency list. Lines 7 through 9 generate the component entries for the Attributes, Routines, and Nomenclature of the cataloged entry. If the class being cataloged has a parent then the component entries for the Attributes, Routines and Nomenclature are updated in lines 12 to 14. These updates include the identification of recursive attributes and routines, and redefined routines [7]. Complete cataloged entries are added to the the list of cataloged entries allTaxEntries in line 16.

```
1: allTaxEntries class_Cataloger (allClassInfos)
   /*Input: allClassInfos - a list of classInfo objects
     with the class characteristics for each class in the
     Java application.
     Output: allTaxEntries - a list of cataloged entries */
2: allTaxEntries ← ∅
3: taxEntry ← null
4: dependencyList ← order(allClassInfos)
   /* Order classInfo objects based on inheritance
     relationships i.e. a parent is always processed
     before a child */
5: for all classInfo ∈ dependencyList do
6:    create taxEntry(classInfo)
      /* CEs - Component Entries */
7:    taxEntry.generateAttributeCEs(classInfo)
8:    taxEntry.generateRoutineCEs(classInfo)
9:    taxEntry.generateNomenclatureCE(classInfo)
10:   if classInfo.hasParent() = true then
11:      parentTaxEntry ← allTaxEntries.getParent(taxEntry)
12:      taxEntry.updateAttributeCEs(parentTaxEntry)
13:      taxEntry.updateRoutineCEs(parentTaxEntry)
14:      taxEntry.updateNomenclatureCE()
15:   end if
16:   allTaxEntries.add(taxEntry)
17: end for
18: return allTaxEntries
19: end class_Cataloger
```

Figure 3. The algorithm for class_Cataloger.

The running time of this algorithm class_Cataloger is $O(n * (a + r))$, where n is the total number of .class files, and a and r are the average number of attributes and routines per .class file, respectively.

4 TaxTOOLJ

TaxTOOLJ - *a Taxonomy Tool for the Object-Oriented Language Java* reverse engineers the classes of a Java application and catalogs them using a taxonomy of OO classes [7]. Figure 4 shows a package diagram of the major components in TaxTOOLJ. The packages clouseauJ_API and tax_Cataloger implement the algorithms class_Characteristics and class_Cataloger shown in Figures 2 and 3, respectively. The package tax_Controller contains the logic that interacts with the

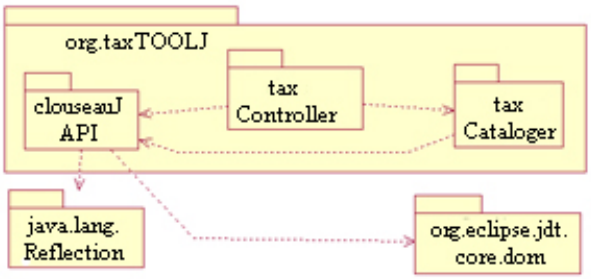


Figure 4. Class diagram for TaxTOOLJ.

user either via a GUI or command prompt. TaxTOOLJ uses two library packages java.lang. (for reflection) [11] and org.eclipse.jdt.core.dom (to build the AST) [14].

4.1 ClouseauJ_API

The clouseauJ_API provides an interface that allows the tax_Cataloger to access all of the class information required to generate a cataloged entry including: accessibility, visibility, and types of classes, routines, and attributes for the program under consideration. We use a combination of the reflection facility in Java and querying the abstract syntax tree (AST) to extract this class information.

Reflection: The Reflection facility provides access to the information of a class through a Class object, which is used to fetch information about the class in the Java application [1]. The core Reflection API is located in the package java.lang.reflect and includes classes Field, Method, and Constructor that serve as a means for extracting information about Java objects. Reflection provides a means for determining the properties, events, methods, and members of a class. However, reflection cannot provide the information about the implementation of the method that might contain local variable declarations and exception handling.

AST: In order to identify a complete set of the descriptors and type families for a each routine, we generate an AST, which is traversed to find the information about the implementation of the methods. We create the AST by using the Eclipse package org.eclipse.jdt.core.dom [14] which contains the set of classes that model the source of the Java program as a structured document.

4.2 Tax_Cataloger

The package tax_Cataloger uses the clouseauJ_API package to access information used to catalog each class in a Java application, starting with the classes

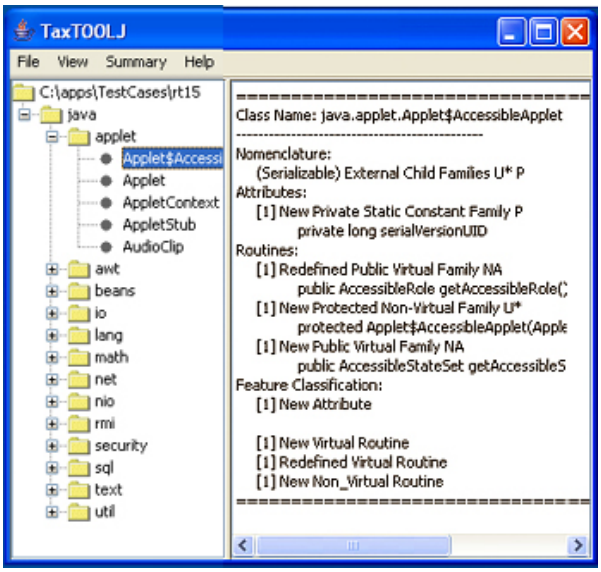


Figure 5. Snapshot of the GUI interface for TaxTOOLJ. A cataloged entry for the class java.applet.Applet$AcessibleApplet**.**

in the global package of the application followed by the class definitions in other packages. The algorithm class_Cataloger in Figure 3 provides additional details.

4.3 Example

Figure 5 shows a snapshot of TaxTOOLJ when the classes in Java Development Kit (JDK) 1.5.05 [11] are cataloged. The tool features a simple menu bar that allows the user to change the level of catalog information that is displayed. The tool's interface further consists of two panes. The left pane is an explorer window that lists the package structure of the application being cataloged, including the names of the individual classes within packages. The explorer window allows users to easily navigate through the tree structure of the application classes, and view the cataloged entry of each individual class simply by clicking on the desired class name, as illustrated in Figure 5. The cataloged entry for the class java.applet.Applet$AcessibleApplet is shown in Figure 5.

5 Case Study

In this section we present a summary of the approach used and the results obtained when TaxTOOLJ is used to catalog the classes in the JavaTM 2 Platform Standard Edition 5.0 Development Kit (JDK 1.5) update 5 (JDK 1.5.05).

5.1 Overview of the Application

Prior to cataloging all of the classes in the JDK 1.5.05 we ran a script that uncompressed all of the jar files, placing each class file in the appropriate package. After extracting the class files the JDK 1.5.05 folder contained 19,219 files in 978 folders, of which 17,343 were .class files. The experiment was performed on a Xeon 2.40 GHz PC with 3GBs of RAM. The settings for the JVM were -Xms1000m -Xmx1500m -XX:MaxPermSize=128m. These settings were required due to the large number of classes that were loaded during analysis.

5.2 Results

Table 1 shows a summary of the results obtained when JDK 1.5.05 was analyzed by TaxTOOLJ. The first column of Table 1 identifies the categories of entities and the second column the number of entities. Row 1 states that the 17,343 classes were cataloged by TaxTOOLJ and Row 2 states that the classes in JDK 1.5.05 were cataloged into 1,057 groups. Rows 4 and 6 show similar information for Attributes. Note that the number in Row 4 represents all the attributes cataloged by TaxTOOLJ including inherited (or recursive) attributes. Row 5 identifies the number of attributes that are not recursive. Row 7 though 9 can be described similarly for the routines cataloged by TaxTOOLJ.

Categories	*Total*
Classes in application	17,343
Class groups generated	1,057
Attributes cataloged	106,825
Non-recursive Attributes cataloged	66,799
Attribute groups generated	198
Routines cataloged	542,554
Non-recursive Routines cataloged	143,047
Routine groups generated	1056

Table 1. Results obtained when JDK 1.5.05 is cataloged using TaxTOOLJ.

Table 2 shows a sample of the groups generated by TaxTOOLJ for classes, attributes and routines. There are three entries shown each for classes, attributes and routines. The three entries represent the least frequent, median and most frequent group. For example, the median group of classes has entry *(Public) (Implements) (Serializable) Generic Internal Child Families U* U<A*>* P A**. This entry represents those classes that are public, implements an interface, is serializable, has generic types, and is neither a root or leaf class in an inheritance hierarchy. The entry also declares references to user-defined types (*U**), a parameterized type containing a reference to an unknown type *U<A*>**, a primitive type (*P* and a reference to an unknown type *A**. The entry described summarizes the class characteristics of two classes: java.util.concurrent.Executors$DelegatedExecutorService and java.util.LinkedHashSet. The complete set of results for the classes in JDK 1.5.05 is provided at [9]. Note that the results shown in Tables 1 and 2 are generated using TaxTOOLJ's Reflection capability only. Although the AST inspection capability of TaxTOOLJ is implemented we are facing problems with memory management for large applications.

No.	Groups
Classes:	
1	(Public) (Has-Inner) Internal Child Families U<U*>* U<A*>* U* P A*
2	(Public) (Implements) (Serializable) Generic Internal Child Families U* U<A*>* P A*
1,153	(Implements) Inheritance-free Family U*
Attributes:	
1	New Private Family U<L*, U*>*
75	New Polymorphic Protected Static Family U*
14,051	Recursive Public Constant Static Family P
Routines:	
1	(Final) New Non-Virtual Private Family A*
8	(Generic) New Virtual Public Static Families U* U<A*>*
94,112	Recursive Virtual Public Family NA

Table 2. Some of the groups generated by TaxTOOLJ for JDK 1.5.05.

Currently, the results generated by TaxTOOLJ are not necessarily conclusive but can be used to further automate the implementation-based testing process. A prototype of a similar tool was created for the C++ language which shows how the information generated by TaxTOOLJ can be used to automatically map implementation-based testing techniques to classes [5].

6 Related Work

Harrison et al. [10] overview three common OODM sets that are captured by several tools, including *Dependency Finder* [13] and *JDepend* [3]. Dependency Finder recognizes four levels of metrics: Project, Group, Class and Method. It can compute the number of classes, number of public methods, depth of inheritance, and other design metrics in different levels. TaxTOOLJ and Dependency Finder produce several metrics in common, such as number of classes, number of attributes and number of methods. Although Dependency Finder provides additional design metrics, such

as depth of inheritance and ratio of public classes, it does not show how the characteristics of a class are combined. Identifying the combinations of characteristics for a class and its members can be used in certain phases of software development, e.g., implementation-based testing and maintenance [4, 6].

JDepend [3] is another tool that computes several types of code metrics and dependency metrics. Some of those metrics, such as number of classes and number of interfaces, are closely related to the features captured by our tool. Our tool is able to gather some valuable OO features, such as class properties related to inheritance and exception handling, which are not provided by JDepend. And again, although those metrics generated by JDepend can provide valuable information, it does not show how the class characteristics are combined in order to aid the tester in the testing process.

Clarke et al. [6] presents a complete taxonomy to catalog C++ classes and constructs a tool which is able to catalog C++ classes using their taxonomy. Similar to our tool, their tool enables maintainers to abstract combined characteristics of a class. However, their implementation of the tool for C++ does not handle several features such as templates, exception handling structures and parameterized types, which are covered by our tool for Java.

Crowther et al. [7] describes the class abstraction technique to support the analysis of Java programs and provides a high-level design for a tool to catalog Java classes based on their taxonomy. In this paper we present the details of their tool by describing the concrete algorithms and methodologies used for the actual implementation of the tool. We also show how the tool is used to catalog the classes in JDK 1.5 update 5 [11].

7 Concluding Remarks

In this paper we presented the design and implementation of TaxTOOLJ - *a Taxonomy Tool for the Object Oriented Language Java*, that reverse engineers the Java classes. TaxTOOLJ takes class files and the source code of the given Java application as input, and generates cataloged entries for all the classes using the descriptors and type families of the various taxonomy components. We also presented the results of applying the tool to catalog JDK 1.5.05. Our tool is the first step to developing an implementation-based testing framework for large Java applications.

ACKNOWLEDGEMENTS

We would like to thank David Crowther and the members of the FIU Software Testing Research Group for their contribution to this work.

References

[1] K. Arnold, J. Gosling, and D. Holmes. *The Java (TM) Programming Language, Fourth Edition*. Addison Wesley Professional, Reading, Massachusetts, fourth edition, 2005.

[2] M. Bruntink and A. van Deursen. Predicting class testability using object-oriented metrics. In *Proceedings of SCAM '04*, pages 136–145. IEEE, Sept 2004.

[3] M. Clark. JDepend, 2001. `http://www.clarkware.com/software/JDepend.htm` (Mar. 2006).

[4] P. J. Clarke and B. A. Malloy. A taxonomy of oo classes to support the mapping of testing techniques to a class. *Journal of Object Technology*, 4(5):95–115, July-August 2005.

[5] P. J. Clarke, B. A. Malloy, J. Ding, and D. Babich. A tool to automatically map implementation-based testing techniques to classes. *IJSEKE (to appear December 2006)*.

[6] P. J. Clarke, B. A. Malloy, and P. Gibson. Using a taxonomy tool to identify changes in OO software. In *Proceedings of 7th European CSMR*, pages 213–222. IEEE, March 2003.

[7] D. Crowther, D. Babich, and P. J. Clarke. A class abstraction technique to support the analysis of Java programs during testing. In *Proceedings of the 3rd ACIS SERA Conference*, pages 22 – 29. IEEE, 2005.

[8] J. Daly, V. P. L. Brian, and J. Wust. Predicting fault-prone classes with design measures in object-oriented systems. In *Proceedings of the The Ninth International Symposium on Software Reliability Engineering (ISSRE '98)*, pages 334–343, Washington, DC, USA, 1998. IEEE Computer Society.

[9] FIU Software Testing Research Group. TaxTOOLJ, March 2006. `http://www.cis.fiu.edu/~tking003/strg/project.html` (Mar. 2006).

[10] R. Harrison, S. Counsell, and R. Nithi. An overview of object-oriented design metrics. In *8th International Workshop on Software Technology and Engineering Practice*, pages 230–237. IEEE, July 1997.

[11] Sun Microsystems, Inc. Core Java J2SE 5.0, February 2005. `http://java.sun.com/j2se/1.5.0/index.jsp` (Mar. 2006).

[12] M.-H. Tang, M.-H. Kao, and M.-H. Chen. An empirical study on object-oriented metrics. In *Proceedings of the Sixth International Software Metrics Symposium (METRICS'99)*, pages 242–249, Los Alamitos, CA, Nov. 1999. IEEE Computer Society Press.

[13] J. Tesser. Dependency Finder, 2002. `http://depfind.sourceforge.net` (Mar. 2006).

[14] The Eclipse Foundation. Eclipse SDK, March 2006. `http://www.eclipse.org/`.

[15] H. Younessi. Managing software defects in an object-oriented environment. *Defense Software Engineering*, pages 13–16, September 2003.

A Relationship-based Flexible Authorization Framework for Mediation Systems *

Li Yang, Joseph M. Kizza
Department of Computer Science
University of Tennessee at Chattanooga
Chattanooga, TN 37415

Raimund K. Ege†, Malek Adjouadi
Computer Information Science
Florida International University
Miami, FL 33199

Abstract

Security is a critical concern for mediator-based data integration among heterogeneous data sources. Due to the interconnected and collaborative computing environment, most mediation systems demand flexible and fine-grained authorization capability. There are three major security challenges faced by mediation systems: context-awareness, semantic heterogeneity, and multiple security policy specification. Currently no existing approach addresses all three security challenges in mediation systems. This paper provides a modeling and architectural solution to the problem of mediation security that addresses the aforementioned security challenges. This paper presents a generic, extensible modeling method for the security policies in mediation systems. A series of authorization constraints are identified based on the relationship on the different security components in the mediation systems. Moreover, we enforce the flexible access control to mediation systems while providing uniform access for heterogeneous data sources.

Keywords: Security, mediation, semantic heterogeneity, access control, relationship, context-aware

1 Introduction

Mediators[12] are intelligent middleware components that sit between information system clients and sources. They perform functions such as integrating domain-specific data from multiple sources, reducing data to an appropriate level and restructuring the results into an object-oriented structure. A mediation security system is designed to cope with security issues in a collaborative computing environment. The users' access to their authorized information should be permitted, while access to the unauthorized information must be denied. For the safety reason, the latter is more critical.

Only when the access constraints on information from different databases are assured is effective information sharing and interoperation possible. So security checking should be interposed between external accessors and data sources to be protected. Regarding to the security issues, the major requirements of mediation efforts can be summarized as: (1) users of an application should be able to access information from any or all of the sources in a uniform and consistent way. Specialized knowledge of the individual sources should not be assumed; (2) users of an application must be denied access to any unauthorized information from either global level or source level, i.e. both the global and local policies must be respected; and (3) access control must support flexible and fine-grained security policies.

Several projects have been carried out to support fine-grained access control in context-based or multiple policies environment, and this paper focus on both context-aware and semantic heterogeneity concerns in mediation systems. S. Dawson [5] presented a solution to the problem of providing interoperation while preserving autonomy and security of the local sources based on the use of wrappers and a mediator. But in the authorization they did not consider context information that frequently determines the access control: especially in a highly shared and interconnected environment. E. Damiani et al. [4] exploited XML's own capabilities, allowed the definition and enforcement of access restrictions directly on the structure and content of the documents, but only the hierarchy between objects was supported. G. Kuper et al. [9]specifies access control policies with XPath

*supported in part by Tennessee Higher Education Commission's Center of Excellence in Applied Computational Science and Engineering under grants R04-1302-005

†affiliation: Department of Computer Science, Northern Illinois University, De Kalb, IL.

expression for data-dependent access control policies and security views are computed to characterize the information accessible to authorized user. The XML representation and view computation share the same ideas with our works, but they did not handle the context-aware constraints. J. Crampton [3] restricted the constraints in the domain of separation of duty. Besides the exclusion constraints, we support the constraints on context-aware, information-flow, semantic mapping and the relationship between the subject and the object. XACL [8] integrated security features such as authorization, non-repudiation, confidentiality and an audit trail for XML document. J. Barkely [1] introduced the relationship concept in the security specification but did not provide the formal definition and modeling strategy for the authorization specification. R. Bhatti et al. proposed X-GTRBAC [2] to solve the conflict among heterogeneous access control policies of multiple domains to allow secure interoperation within the enterprise, but they leave out the solution for data sources heterogeneity. Moreover, above related works disregard the pressing need for interoperation and information sharing among databases, especially the semantic-related heterogeneous data sources.

In this paper we propose a generic, reusable and extensible modeling method for access control to secure information upon interoperation in the context of a distributed mediation system. The data sources are semantically related and the security policy specification at different data sources may differ. We present a security enforcement architecture that make data from heterogeneous data sources available to external applications in such a way that their autonomy and security are not compromised.

This paper is organized as follows: Section 2 introduces the background knowledge about mediation systems and observations on the requirements. Section 3 illustrates the components for access control specification in mediation system. Section 4 explains the access control enforcement for mediation system. Section 5 contains the concluding remarks.

2 Mediation and Security Concerns

Mediation systems are designed to provide an integrated view of information from heterogeneous sources. The goal of such systems is to permit the exploitation of several independent data sources as if they were a single source, with a single global schema. A user query is formulated in terms of the global schema; to execute the query, the system translates it into subqueries expressed in terms of the local schemas, sends the subqueries to the local data sources, retrieves the results, and combines them into the final result provided to the user. We proposed the three-layered mediator architecture [14] to resolve the semantic gap among the heterogeneous data sources. A mediation system where employs the Global As View (GAV) [11] includes three parts: a *global (mediated) schema type*, a set of *source schemas type* and a *mapping relation*. A *global schema type* is tree type whose labels are terms of a global vocabulary, different from *source schema type* that uses labels defined by local data schema. The global schema vocabulary has been chosen to unify the local vocabulary and represent a specific domain of interest. The *mapping relation* between the global schema type and source schemas type specify how to load global schema patterned data from the sources.

The authorization policies are specified at the global and source level respectively. In order to observe both global and source security policies, we need the semantic translation or mapping between the global and source data type to facilitate the policies propagation and conflict resolution.

In a mediation system, particular requirements regarding to the security occur [15]: Firstly, multiple heterogeneous data sources participate in the system and autonomy of security policy specification at each source level is allowed. That means security policy specification should cover security policy specification at different points, i.e., global level and source level. Secondly, we face an interactive and interconnected system where context information like time, location, and access history frequently determines the access control decision [10]. Therefore, in order to enforce context-aware and fine-grained access control policies, permissions and permission assignment should support context attributes from the user, from the object to be accessed, or from the relationship between user and object. Thirdly, there is semantic heterogeneity for various information sources.

3 Flexible Context-Aware Authorization Specification

In this section, we explain the components of our flexible context-aware authorization framework and give a formal definition to capture the nature of a secure mediation data system.

3.1 Components Description

The authorization framework is designed to describe the circumstance where a user tries to execute a

specific access operation on given data. Thus, the authorization framework should define the subject (user, role), operation (read, write, update) and object (data) entities. The data could be stored in the flat file system, a semi-structured or structured database. In our authorization framework we deem the data items as partially ordered. The users who attempt to access the data object are organized hierarchically to conform to the structure of the organization. In role-based access control (RBAC) [6] the user assumes roles, and roles may organized as a hierarchy too, i.e., *chair, dean*. The operations the user performs on the data objects could include *read, write or update*.

Moreover, whether the user can access a data object is determined by context information. In addition to simple context information such as time and location, we also consider arbitrary relationships on entities of concern, such as, the user, the object to be accessed, and the subject matter of the information contained with in the object [1]. We capture such kind of information with an *relationship* entity. For instance, if a patient's age is over 21, his/her sensitive medical record can be read by the relative only (1) with consent of the patient his/herself, (2) in the hospital, and (3) during working hours. An access decision for this policy includes the following factors: the age attribute of the patient; the sensitive property of the object; the relationship, i.e., between the user and the patient; the history, i.e., consent from the patient; the time attribute of the access session; the location attribute of the access session.

3.2 Formal Definitions

In general, a data system consists of subjects (users, groups, roles), objects (data) they are accessing, and the types of access modes they use. The logic behind this definition derived from the ideas in the Entity-Relationship model, which is to allow the description of the conceptual scheme of a data system.

Definition 1 (Data System) *A data system DS contains 6 components (SH, OH, OP, A, C, Rel), SH is the partial ordered subjects, OH is the partial ordered objects, OP is the operation that could be performed on the objects, $A \in \{+, -\}$ denotes authorized or denied respectively, C is the session under which the access request session is initiated and Rel is a set whose elements are called relationships. Relationships can be defined on the different elements of DS and may be unary, binary or n-ary in natural.*

In the above definition, we introduced the session concept into S. Jajodia's definition [7] for data system in order to to handle context information and thus make access control context-aware. Moreover, the relationship between entities is defined as explicitly as the entity itself. In this section, the relationship entity is clearly identified, analyzed and classified to incorporate the attributes of each entity, the relationship between entities, and the relationship between the entity and the attribute of entities. Each Entity set has properties, called attributes, which associate with each entity in the set a value from a domain of values for that attribute. The entity set of subjects s may be declared to have attributes, such as name, role, recorded in his/her credentials. The entity set of sessions c may have the attributes, such as time, location, to denote the context information under which the session is initiated. The entity set of objects o may bear the attributes, such as *owner*, *relative*, *sensitive level*.

Besides the attributes of the entities, relationship can be defined over different elements of the entities. Relationships can specify the relationship between subjects, objects, operations, subjects and objects, or the subjects and the objects' subject. Examples of relationships include: 1) relationship between subjects: subjects; $supervise(s_1, s_2)$ specifies that the user s_1 is the supervisor of the user s_2. 2) relationship between objects: $typeof(o_1, o_2)$ specifies the hierarchy of two objects; $mapping(o_1, o_2)$ specifies that the object o_1 patterned in global data type can be translated to the object o_2 patterned in local data type, thus we can propagate the authorization specification along the semantic mapping path between the global data type and the local data type. 3) relationship between operations: $exclusiveOp(op_1, op_2)$ specifies that there is no user can perform both operations. 4) relationship between subject and object: $owner(s, o)$ specifying the ownership between subjects s and objects o. 5) relationship between the subject and the subject of object: $parent(s, o)$ specifies that the subject s is the parent of the person who owns the objects o.

We especially investigated two essential features of a secure mediation systems, i. e., *relationship based context-aware access control* and *interoperation among heterogeneous databases*, which distinguishes our work from the related work.

3.3 Authorization Specification

In this section, we use subject $subj$ to denote those entities for which authorization permission can be specified, subjects are therefore users, groups, and roles. And object obj refers to entities on which authorization can be specified, e.g. objects and ob-

ject schemas. The context cxt is used to denote the context-related information, i.e. environmental information or relationship defined on the subjects and objects. The context information can be captured either by environmental sensors or by accessing data sources or user credentials.

3.4 Constraints of Authorization Specification for Mediation Systems

We now categorize and enumerate a variety of constraints that have been identified as the relationship over the entities in the mediation data system (O, S, OP, C, OA, SA). Some of these constraints have already been discussed in existing literature, but several are novel; as a set they are essential to a secure mediation system. The authorization constraints denoted by the predicates are illustrated a s follows:
(1) context-based constraints define the location and time attribute of a session. For instance, the following formula denotes that the *relative* of a patient can read his/her medical record object o within the hospital hours and inside hospital:
$cando(s, o, +read)$
$\leftarrow relative(s, o)\&workTime(sessionID)$
$\&inHospital(sessionID)$, where the relationship predicate ***relative(s, o)*** denotes that the subject s is the relative of the object o's owner. The ***workTime*** and ***inHospital*** are environmental context constraints.

(2) subject-based constraints define the relationship between subjects, i.e., hierarchical, exclusive. The security policies could be propagated along the subject hierarchy. The following formula denotes that the sub-subject s_1 inherits the right from the super-subject s_2, where the predicate $super(s_1, s_2)$ denotes that s_1 is the super-subject of s_2.
$cando(s_1, o, op)$
$\leftarrow cando(s_2, o, op)\&super(s_1, s_2)$

(3) object-based constraints define the hierarchical or exclusive relationship between two objects. The following formula denotes that the object o_1 and object o_2 can not be read simultaneously by the same subject s:
$cando(s, o_1, -read)$
$\leftarrow cando(s, o_2, read)\&exclusiveObj(o_1, o_2)$

(4) operation-based constraints define the relationship between operations.

(5) history-based constraints could preventing the user from obtaining the sensitive data by data mining. We maintain the history table to record the users' access record together with access time. The predicate ***inHistory(s, o, op)*** is used to decide (s, o, op) has been stored in the history table or not. If subject s has read o_2, he/her is not allowed to read o_1: $cando(s, o_1, -read) \leftarrow inHistory(s, o_2, read)$

(6) information-flow constraints protect data from flowing from high sensitive level to low sensitive level. The user who reads object o_1 with higher sensitive label can not write object o_2 with lower sensitive label:
$cando(s, o_2, -write)$
$\leftarrow cando(s, o_1, read)\&highLevel(o_1, o_2))$,
where predicate ***highLevel*** is used to denote that object o_1 is more sensitive than object o_2.

(7) semantic mapping constraints define the relationship between the objects in the global level and the objects in the source level. The global-level policies specify authorization policies for the global data type, but need to be enforced to source data objects that are patterned by source data types. The policies at the global level are pushed to the source level and the conflicts are resolved if necessary. So the authorization policies represented at the global level need to be translated and propagated along the semantic mapping path to the objects at the local level. The conflicts between the global and source policies are resolved in the ***decision combinator*** module of our enforcement architecture (see Figure 1). The following policy denotes that the authorization policy propagates along the semantic mapping path from global level to source level, which is more significant in the mediation systems.
$cando(s, o_2, op, source)$
$\leftarrow cando(s, o_1, op, global)\&mapping(o_1, o_2)$

4 Authorization Enforcement Architecture

Our architecture [13] assumes that access is requested by a user, possible playing a role. Access control is enforced with respect to the role if a role is active. We generically refer to the requester as subject. When a subject s requests permission to execute action *op* on object *o*, we need to check whether the authorization $(s, o, +op)$ can be derived by the access control module from the authorization rules in Figure 1. If $(s, o, -op)$ is derivable, then the access is denied.

The authorization enforcement architecture is composed of the following modules: Access control module; Policy Locator; Dynamic Attribute Managers; Constraint Service; and Decision Combinator. When the *Access Control* module receives the request for authorization decision from the user, the *Policy Locator* is activated to retrieve all the applicable rules from the authorization rules repository. For

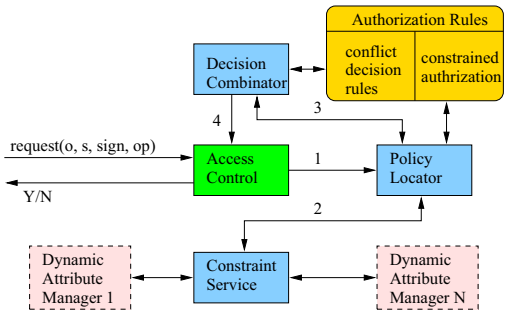


Figure 1: A Flexible Context-aware Authorization Framework

each applicable authorization rule, one or more *Dynamic Attribute Managers* are called by the *Constraint Service* module to evaluate the constraints in each authorization rule. If all the constraints in one authorization rule are evaluated to be true, that authorization is granted. And if all the applicable rules allow the requested operation on the requested object, the request is granted. If any two rules conflict with each other, the *Decision Combinator* module is activated and the conflict rules are retrieved from the authorization rules repository to resolve the conflict. The *Decision Combinator* employs the similar resolution in case no applicable rules found.

5 Conclusion

This paper has proposed a flexible context-aware authorization framework for the mediation systems. The modeling flexibility is achieved by defining the authorization constraints defined based on the *relationship* over the authorization components of the mediation systems. Such generic, extensible modeling strategy distinguishes our work from the related work. The particular feature for the mediation systems, i. e., interoperation among heterogeneous databases was investigated in the security policies specification also. Our authorization framework supports enforcement of the context-aware security policy specification.

Our future research interests include trust in mediation system, security policy observance over data integration.

References

[1] J. Barkley, K. Beznosov, and J. Uppal. Supporting relationships in access control using role based access control. In *Proceedings of the fourth ACM workshop on Role-based access control*, pages 55–65. ACM Press, 1999.

[2] R. Bhatti, B. Shafiq, E. Bertino, A. Ghafoor, and J. B. D. Joshi. X-GTRBAC admin: A decentralized administration model for enterprise-wide access control. *ACM Transaction Information System Security*, 8(4):388–423, 2005.

[3] J. Crampton. Specifying and enforcing constraints in role-based access control. In *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 43–50. ACM Press, 2003.

[4] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for XML documents. *ACM Transaction Information System Security*, 5(2):169–202, 2002.

[5] S. Dawson, S. Qian, and P. Samarati. Providing security and interoperation of heterogeneous systems. *Distributed and Parallel Databases*, 8(1):119–145, 2000.

[6] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM Transaction Information System Security*, 4(3):224–274, 2001.

[7] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transaction Database System*, 26(2):214–260, 2001.

[8] M. Kudo and S. Hada. XML document security based on provisional authorization. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 87–96. ACM Press, 2000.

[9] G. Kuper, F. Massacci, and N. Rassadko. Generalized XML security views. In *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 77–84, New York, NY, USA, 2005. ACM Press.

[10] M. Strembeck and G. Neumann. An integrated approach to engineer and enforce context constraints in rbac environments. *ACM Transaction Information System Security*, 7(3):392–427, 2004.

[11] J. D. Ullman. Information integration using logical views. *Theoretical Computer Science*, 239(2):189–210, 2000.

[12] G. Wiederhold. Mediators in architecture of future information systems. *IEEE Computer*, 25:38–49, 1992.

[13] L. Yang and R. K. Ege. Security enforced mediation systems for data integration. *Journal of Infocomp*, 2005.

[14] L. Yang, R. K. Ege, and H. Yu. Dynamic integration strategy for mediation framework. In *Softerware Engineering and Knowledge Engineering (SEKE05)*, Taiwan, China, June 2005.

[15] L. Yang, R. K. Ege, and H. Yu. Security specification and enforcement in heterogeneous databases. In *The 20th Annual ACM Symposium on Applied Computing, Computer Security Track*, Santa Fe, New Mexico, March 2005. ACM press.

Towards Secure Ambient Intelligence Scenarios*

Antonio Maña, Francisco Sanchez-Cid, Daniel Serrano, Antonio Muñoz
Computer Science Department
University of Málaga, 29071 Málaga, Spain

Abstract. **Ambient Intelligence (AmI for short) represents a promising concept based on the ubiquity and intelligence of computational devices embedded into all types of objects. The realization of this concept entails many important challenges, such as the provision of intelligent and adaptive interfaces, the dynamic assembly of components, the automated negotiation, and many others. However, one of the most important barriers to the realization of Ambient Intelligence is the lack of adequate support for security. In particular, the lack of security-aware development processes is a very negative factor influencing this situation. In this paper we present a new model for addressing the security issues in the development of distributed applications and systems for very dynamic and heterogeneous scenarios such as Ambient Intelligence. The model is based on the concept of Security and Dependability (S&D) Pattern. S&D Patterns are precise, machine- understandable and user-meaningful representations of validated security and dependability solutions and mechanisms.**

I. INTRODUCTION

Security is becoming one of the most critical concerns for the companies developing software in these days. New vulnerabilities and security bugs are published everyday in specialized web pages as Security Focus [1] or Astalavista [2], and the tendency is towards an increment on the number of reported vulnerabilities. The trend towards open environments, the crescent number of devices working online and connected to some kind of network, and the demand of mobile technology to connect from anywhere at anytime, has caused the apparition of new threats and new security concerns. Attacks based on buffer overflow, SQL injection, or DNS Spoofing have been identified and largely discussed by the security community, but those attacks still as the main breach on security for the current applications. Designers continue making the same mistakes, permitting the same attacks to be successful again and again, as similar security breaches continue unsolved.

During this time, Software Engineering techniques have advanced providing more automated means to allow us developing more *correct* software. Unfortunately, the development of correct software does not imply the development of *secure* software. It is not enough providing the desired functionality: the *intended* functionality (and nothing more) has to be provided, assuring the absence of hidden side effects. It is proved that little bugs that do not affect in a visible way the functionality of the application are seeds for the grown of a future security breach. Therefore, when the Software Engineer is designing a new application he has to keep in mind the chain effects of the possible failures of the application as well as the hidden effects of some *intended* functionality. For example, in an old version of a popular email client, an email with a very long file name attachment (e.g. more than 300 characters) caused a crash when trying to download the message from the email server, causing the same error every time it tried to download mail from this account [3]. The system was properly designed, and the *usual* attachments were managed properly, but a malicious attacker using the attachment feature was able to make the system complete *unusable*, breaking the *availability* of the system.

The Software Engineer also has to take into account the possible external security threats or those coming from the interaction with other foreseen or unforeseen elements of the system. In the new environments, the concepts of system and application as we know them nowadays will disappear, evolving from static architectures with well-defined pieces of hardware, software, communication links, limits and owners, to architectures that will be sensitive, adaptive, context-aware and responsive to users' needs and habits. These ecosystems (such as Ambient Intelligence –AmI) will offer highly distributed dynamic services in environments that will be heterogeneous, large scale and nomadic, where computing nodes will be omnipresent and communications infrastructures will be dynamically assembled. The combination of heterogeneity, mobility, dynamism, sheer number of devices, along with the growing demands for security, is going to make the provision of security for these ecosystems increasingly difficult to achieve with existing security solutions, engineering approaches and tools.

The conclusion of those facts is that the average Software Engineer has to be assisted in his lack of necessary knowledge on security to analyse the threats of the system and expertise to apply sound solutions to avoid those threats, in order to improve and secure the final design.

From the late 90's, the security research community has proposed several approaches based on *security patterns* to bridge the gap on security knowledge between the average Software Engineers and the Security Engineers. Security patterns describe a proved solution to some common security problems that arise in specific scenarios. These patterns represent a standard mechanism to design more secure systems as they embed the knowledge and the expertise of Security Engineers ready to be applied on the new application designs [4]. Nevertheless, far from being simple, the application of these well-proven generic schemes requires some experience on the application of design patterns as well as a previous analysis of the potential threats faced by the system. Furthermore, each pattern has some constraints and consequences that arise when it is applied. Therefore, it may require the simultaneous application of some other security pattern or significant changes in the design of the system.

* Work partially supported by the E.U. through project IST-027587

Furthermore, dynamic environments (such as AmI) create the necessity of building systems capable of adapting their security mechanisms to the constantly changing context. Then, the semantic description of the patterns, including their side effects when applied and their exact relationship with other patterns of the system, has to be considered and monitored for correctness using information extracted from the problem and the evolving context. The inclusion of the SERENITY framework that we propose in the software engineering process will allow designers not to *secure* systems, but to *build* secure systems.

II. RELATED WORK

Currently, the provision of appropriate security and dependability (S&D) mechanisms, and its integration into the software engineering procedures, for Ambient Intelligence ecosystems remains an unsolved issue.

Several approaches have been introduced in order to capture the specialized expertise of security engineers and make it available for automated processing providing the basis for automated synthesis and analysis of the security and dependability solutions of systems [5]. These approaches are supported by a wide range of technologies: components, frameworks, middleware, aspects and security patterns, being this last the most relevant from our work point of view.

The concept of security pattern was introduced to support the system engineer in selecting appropriate security or dependability solutions. However, most security patterns are expressed in textual form, as informal indications on how to solve some particular security problem [6-9]. Some of them do use more precise representations based on UML diagrams, but these patterns do not include sufficient semantic descriptions in order to automate their processing and to extend their use [10].

Perhaps the first and the most valuable contribution as pioneer in security patterns as we know them at present, is the work from Joseph Yoder and Jeffrey Barcalow proposing to adapt the object-oriented solutions to recurring problems of information security [9]. In their own words, seven patterns were presented to be used when dealing with application security. A natural evolution of this work is the proposal presented by Romanosky in [11]. It takes into consideration new questions that arise when securing a networked application.

Going one step down in the abstraction scale, Eduardo B. Fernandez in his work about authorization patterns [10] combines for the first time the idea of multiple architectural levels with the use of design patterns. In [12] they propose the decomposition of the system into hierarchical levels of abstraction.

The same author et al. offers in [13] a good source to study the historical approaches that have been appearing in the scientific literature as pattern systems for security. Wassermann and Cheng present in [14] a revision of most of the patterns from [9] and [12] and categorise them in terms of their abstraction level. In order to facilitate the reuse of security knowledge, variations of the design pattern template given in [15] are included in order to better suit the presentation of security specific information. In the field of web application protection, [16] is the source of some patterns for *Application Firewall* and *XML Firewall*.

The special needs of secure-ware systems and the constantly changing context in which the systems are designed nowadays arise some new problems that have an starting solution in works like the one presented by Cheng et al in [8]. The proposal consist of a template for security patterns that takes into account essential information that has not been necessary in the general design patterns but appears as mandatory in the new security context. On the basis of the security patterns described by *Gamma et al.* in [15], a new patterns library is created by the addition of new relevant information (e.g. *Behaviour* or *Supported Principles* are two new field to describe the security pattern) and altering some existing fields (e.g. *Consecuences* is altered to convey a new set of possible consequences including confidentiality, usability, integrity, etc.).

Some security patterns have also been proposed for multiagent and Web Services systems. From the very beginning, the tendency has been to use the object oriented paradigm: in [17] an object oriented access control (OOAC) was firstly introduced as a result of consequently applying the object oriented paradigm for providing access controls in object and interoperable databases. Fernandez proposes in [18] some specific solutions oriented to web services: a pattern to provide authentication and authorization using Role-based access control (the so-called *Security Assertion Coordination* pattern) and a pattern for *XML Firewalls*. The *Security Assertion Coordination* pattern takes as source the abstract security architecture defined by SAML (the main standard to provide security for Web Services). However, a minor work has been done in the field of agent-based systems. A good start point is [19].

Some other authors have proposed ways to provide formal characterizations of patterns. The idea of precisely specifying a given class using *class invariants* and *pre-* and *post-conditions* for characterizing the behaviours of individual methods of the class is well known and is the basis of the *design by contract (DbC)* [20]. Evolutions of that approach have appeared and [21, 22] are some examples of that. Eden *et al.* [21] proposes to use logic formalism with an associated graphical notation to specify rich structural properties. However, it provides only limited support for specifying behavioural properties. In this sense, [22] has as main goal to preserve the *design integrity* of a system during its maintenance and evolution phases. Also Mikkonen in [23] focus his approach on behavioural properties. Here, data classes model role objects, and guarded actions (in an action system) model the methods of the roles.

III. THE SERENITY APPROACH

Defined by the EC Information Society Technologies Advisory Group (ISTAG), in the vision of Ambient Intelligence people will be surrounded by ubiquitous computers with intelligent and intuitive interfaces embedded in everyday objects around them, making the physical environments adapt and respond to users' needs in an invisible way in order to provide anytime/anywhere access to information and services.

The most relevant features inherent to the realization of this vision are the increasing decentralization, high heterogeneity (of devices, applications, user needs, capabilities, etc.), dynamism, unpredictability, lack of predefined trust relations and context awareness.

The provision of S&D in AmI ecosystems requires the dynamic application of the expertise of security engineers in order to dynamically react to unpredictable and ever-changing contexts. Due to the high heterogeneity, dynamism and decentralization of AmI ecosystems, it is not possible, even for the most experienced and skilled security engineers, to foresee all possible situations that may arise during the life of the applications in order to create solutions that can be used in these circumstances. Moreover, due to the highly distributed nature of these ecosystems, applications will no longer belong to or be under the control of a single entity. Therefore, devices and applications must be ready to participate in dynamic collaborations with heterogeneous (in terms of capabilities, functional goals, security and dependability needs, etc.) and non-trusted external elements.

In order to cope with the specific needs of these environments, a possible approach could be to try to create an "intelligent" system able to analyze the requirements and the context in order to synthesize new solutions. Unfortunately, given the state of the art in both security engineering and intelligent systems, this approach is not a promising one in the foreseeable future. On the other hand, a more realistic approach which can take advantage of recent developments in technologies regarding security engineering, run-time monitoring, semantic description and self-configuration is based in capturing the expertise of security engineers and making it available, by means of automated tools, to the dynamic ecosystems. It is our belief that this approach, supported by the enhanced concept of *S&D Patterns* and *Integration Schemes* provide the best means to capture this expertise. While S&D Patterns describe independent security mechanisms, Integration Schemes describe solutions for complex S&D requirements achieved by the combination of various S&D mechanisms.

One additional aspect is that the overall security of a system not only depends on the security mechanisms used within the boundaries of a system but also on a variety of external factors including social context and human behaviour, IT environments, and even protection of the physical environment of systems (e.g. buildings). The actual source of security and dependability requirements lies in the real world. Consequently SERENITY aims at providing solutions to capture these requirements, to trace them and to validate solutions against these requirements. In addition to that, our proposal takes especially into account the computing context including heterogeneous communication systems, computing infrastructures and external actors and components dynamically interacting with the system.

In SERENITY, the main pillar to build solutions is the enhanced concept of S&D Pattern, which can capture security expertise in a way that is more appropriate than other related concepts. Components, frameworks, middleware, and patterns have been proposed as means to simplify the design of complex systems and to capture the specialized expertise of security engineers and to make it available for non-expert developers. However, all these approaches have important drawbacks and limitations that hamper their practical use. The approach taken in SERENITY aims at integrating the best of these approaches in order to overcome the problems that have prevented them to succeed individually. Furthermore, because secure interoperability is an essential requisite for the widespread adoption of the SERENITY model, trust mechanisms will be provided for patterns.

SERENITY's S&D patterns are precise specifications of validated security mechanisms materialised as files containing models that could be described using formal or non-formal languages (e.g. XML and logic-based language) to capture the expertise of security engineers with the objective of being used by automated means. S&D patterns include a precise behavioural description, references to the S&D properties provided, constraints on the context required for deployment, information describing how to adapt and monitor the mechanism, and trust mechanisms. S&D patterns, along with the formal characterisation of their behaviour and semantics, are the basic building blocks of S&D mechanisms that will enable the provision of S&D over a wide range of heterogeneous AmI ecosystems. SERENITY's integration schemes specify ways for systematically combining S&D patterns into systems composed of dynamically collaborating elements that operate in mobile, heterogeneous, and highly dynamic ICT infrastructures.

These new concepts can be useful in two different ways: at design/deployment time and at run time.

In the first case, we must consider that today's large applications are built by *integrating solutions from different sources and at different levels* of abstraction. These applications must face the existence of multiple and heterogeneous user interfaces and access devices and the need to respond to the presence of individuals and the context in an intelligent and invisible way. In this scenario, the abovementioned concepts can facilitate the integration of components from different sources in a controlled and secure way, because components will be described by associated S&D Patterns and Integration Schemes that will contain all necessary information in order to allow system engineers to take an informed decision. Furthermore, automated tools will help these engineers to analyse the resulting systems in order to check that the application of the components has been done in a proper way. For instance, an Integration Scheme can tell the developers that you can use a "digital signature pattern" and an "asymmetric encryption pattern" together as long as the keys used on each pattern are different.

In the second case (that is; during runtime) S&D Patterns and Integration Schemes are used in order to support automated adaptation of the S&D mechanisms to the changing context conditions. To achieve this, it is necessary to have a framework supporting the management of a pattern warehouse and the constant evolution of such patterns, taking into account the context in which they are applied. The SERENITY Framework runtime support is able to select the most appropriate S&D solution among the ones available in the S&D Library, based on the end-user requirements and the actual context, as shown in figure 1.

The global SERENITY framework provides a way to secure interconnected and heterogeneous systems. It is

composed of: (i) *a set of patterns* (for security and dependability), (ii) *integration schemes* (based on the previous set of patterns) describing how to combine patterns in order to achieve the security requirements of the application; and (iii) *Deployment & Run-time monitoring mechanism* able to: evolve according to new possible solutions and requirements (e.g. changes of legal regulations, privacy, new security breaches, new technologies), and validate in terms of dependencies, integration schemes, patterns, etc.

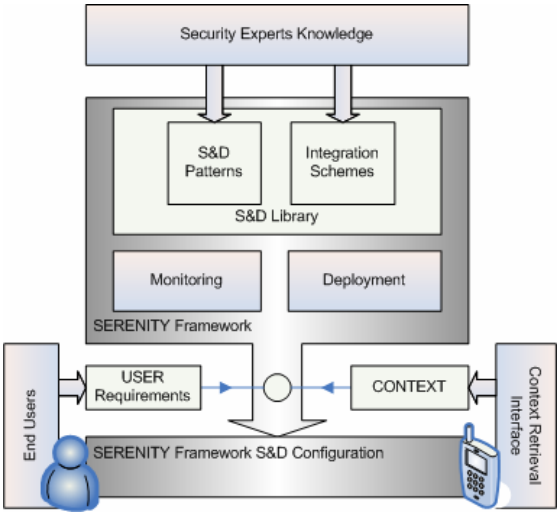


Figure 1. Global SERENITY framework

As mentioned above, the key to success in these scenarios is based on capturing security expertise in such a way that it can be supported by automated means. However, we have assumed that S&D experts will not be able to foresee all possible situations for the operation of the system. Therefore, we need to complement the static analysis performed by the security experts with mechanisms for dynamic supervision that can monitor that the conditions established by the S&D experts are fulfilled during the operation of the system. SERENITY will provide support for the dynamic supervision and adaptation of the security of systems to the changes in ever-changing ecosystems. In this way, SERENITY's integral model of S&D considers not only static aspects but also dynamic aspects by means of the *monitoring mechanism*.

IV. CONCEPTUAL MODEL OF THE SERENITY FRAMEWORK

This section is based on a UML model that describes the essential elements of the SERENITY framework. The logical view of the system is introduced in figure 2, including the different conceptual elements as well as their relations.

We can observe that S&D Patterns and Integration Schemes (*S&DPatterns*) refer to solutions (*S&DSolutions*) and contain both: (i) the semantics (*S&DSolutionSemantics*) that describe the solution and (ii) the semantics that describe the hardware or software components (*ComponentSemantics)* that take part in the solution (*Component*). The solution semantics are described in terms of the semantics (*S&DPropertySemantics*) of the particular properties (*S&DProperty*) provided by the solution. Solutions can be monitored by using certain monitoring mechanisms (*MonitoringMechanisms*).

S&D Patterns and Integrations Schemes are certified by a special type of digital certificate (*PatternCertificate*). The libraries of patterns (*S&DLibrary*) are composed of S&D Patterns and Integration Schemes that have a certificate, denominated certified patterns (*CertifiedS&DPattern*).

Finally, users will define the security and dependability requirements (*S&DConfiguration*) for their systems, which will contain a set of specific requirements (*S&DConfigurationElement*). Each specific requirement will specify a set of properties (*S&DProperty*) that must be enforced for a particular element of the system (*SystemElement*).

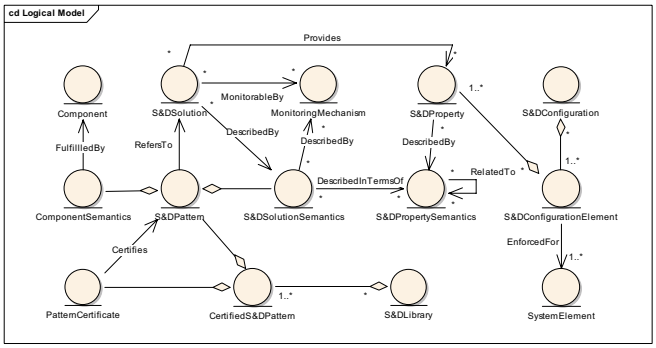


Figure 2: Logical model, conceptual elements.

SERENITY provides a series of tools for security engineers. These are tools for analysing security solutions, for creating S&D Patterns that represent those solutions, and for certifying the S&D patterns and their corresponding solutions.

The functionality offered by the system for S&D engineers are presented in figure 3. In the process of creating a new pattern, security engineers will probably use external tools (for designing, specifying, classifying, implementing ...) their solutions. SERENITY does not address these tools on purpose, in part because they are not directly related to the project goals, and in part because there are many different lines of research and we consider that all of them can provide some value in specific situations.

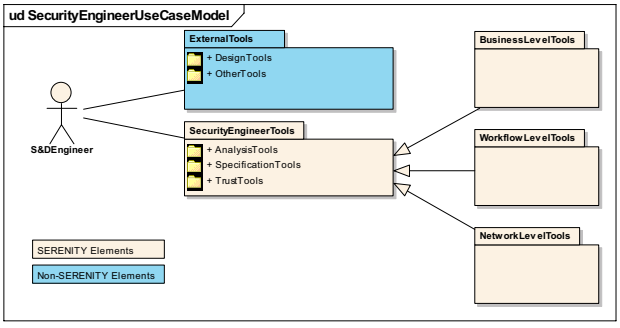


Figure 3: Main functionalities for S&DEngineers.

On the other hand, SERENITY must provide tools for analysing security solutions. There are plenty of these tools available, but in contrast to the previous case, we need to include these tools in our work plan because we want them to produce results that can be later integrated in our framework. Consequently, we are currently in the process of creation and adaptation of tools to produce results that can be represented as S&D Patterns. These tools must address issues that are specific for each abstraction level. Although, each level requires different formalisms, tools and methodologies, they all have in common that the results of the analysis is captured in the form of S&D patterns and Integration Schemes.

With the support of the specification tools security engineers will be able to capture their specific knowledge about a particular solution and to represent it in the form of S&D Patterns and Integration Schemes (see figure 4). Sometimes they will also need support for the semantic definition of security properties, which is the basis for the definition of the S&D Patterns and Integration Schemes. Finally, security engineers will use trust tools to certify their S&D Patterns and Integration Schemes.

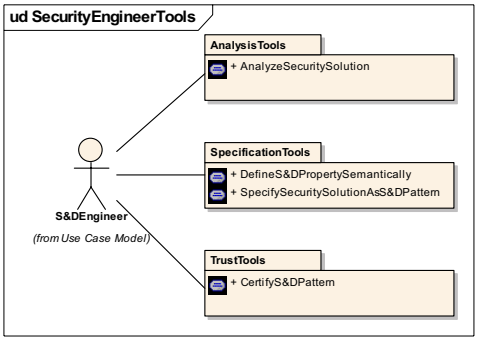


Figure 4: S&DEngineer, SERENITY Framework tools.

Apart from S&*DEngineer*, two other actors will use the framework (figure 5). On the one hand, the users of the systems where the framework is running will be able to manage their S&D patterns and Integration schemes and to configure the framework. On the other hand, external S&D controllers (representing any external element, capable of managing the S&D configuration of a node) will be able to access the monitoring services offered by one SERENITY Framework and to negotiate the configuration in order to interact with it. Of course, from the point of view of a specific instance F_i of the framework, all other instances of the SERENITY framework play the role of external S&D controllers and, reciprocally, all other instances will consider F_i as an external S&D controller. It is not assumed that external S&D controllers must implement all the functionality contained in the *SERENITYFrameworkInterface* package in order interact with SERENITY frameworks.

Additionally, the SERENITY framework will be used by applications in order to manage the selection and configuration of their security and dependability mechanisms. These applications will use the SERENITY framework in order to enforce certain S&D properties. In some cases, the enforcement of a specific property may require the monitoring of the behaviour of the solution during runtime.

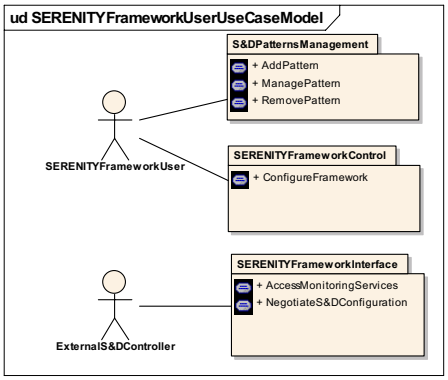


Figure 5: Use case diagram: other users.

Every instance of SERENITY framework as system will provide interfaces in order to allow interaction with other systems or other SERENITY instances, the deployment model of figure 6 represents it.

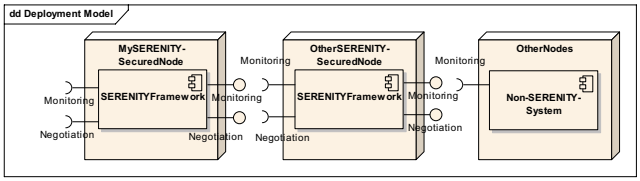


Figure 6: Deployment model, more than one framework instance.

In summary, the SERENITY framework will provide two main interfaces: On the one hand, it will provide a negotiation interface that will be used in order to establish the configuration of the interacting frameworks in a way that suits the requirements from all participants. On the other hand, SERENITY frameworks will offer a monitoring interface. External elements interacting with an instance of the SERENITY framework will be able to monitor that the behaviour of the framework is correct. Of course, the SERENITY framework will also feature the counterparts of these offered interfaces (required interfaces). It is foreseen that some external elements other than SERENITY frameworks will be able to interact with them by way of these interfaces.

Finally, figure 7 shows a simple life cycle for a pattern, including how a new S&D Solution is designed, analyzed, represented in the form of an S&D Pattern, and certified.

The S&D Pattern is then added to an S&D Library, and finally used by the SERENITY framework in order to enforce a specific set of S&D properties (that we will call S&DConfiguration) defined by the end users and their applications.

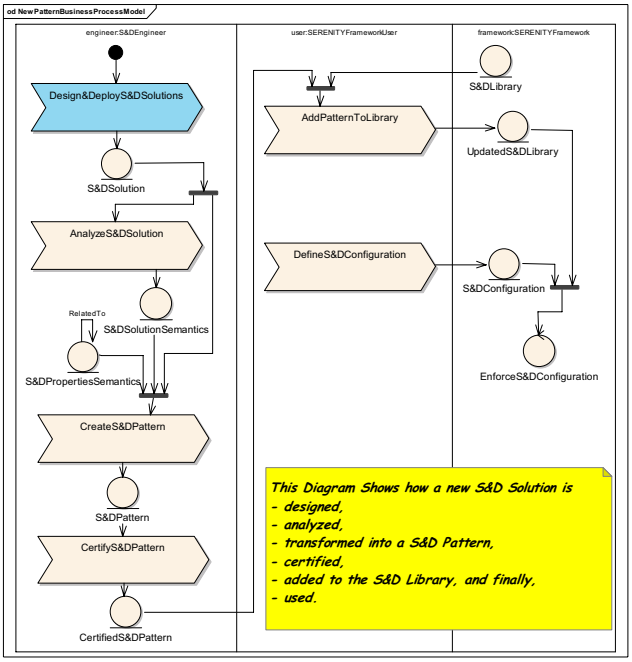


Figure 7: Business model, S&DPattern life cycle.

The conceptual elements in this model are those shown in the logical view. Security Engineer will complete the initial tasks, after of the analysis a new pattern is created, this pattern will be certify. The certifiedS&DPattern will be added to a library by a SERENITYFrameworkUser; this user will manage Framework instance patterns and will define a configuration. All this process will establish a SERENITY instance that satisfies security requirements.

V. CONCLUSIONS

The realization of the Ambient Intelligence concept entails many important challenges, but the most important barriers to this realization is the lack of adequate support for security. In this paper we have presented a new model for addressing the security issues in the development of distributed applications and systems for very dynamic and heterogeneous scenarios such as Ambient Intelligence.

Our model is based on the concept of Security and Dependability (S&D) Pattern. S&D Patterns are precise, machine-understandable and user-meaningful representations of validated security and dependability solutions and mechanisms. We have introduced the basic elements of the approach and have described the conceptual model of our solution.

We are currently in the process of implementing these concepts in the ongoing SERENITY project. Our interest focuses on the realization of the concept of S&D Pattern, the semantic definition of security and dependability properties in order to guarantee the interoperability of S&D Patterns from different sources, and the provision of a trust infrastructure for the patterns.

REFERENCES

[1] The Largest Community of Security Professionals Available Anywhere http://www.securityfocus.com/

[2] Astalavista.com, One of the world's most popular and comprehensive computer security web sites http://www.astalavista.com/

[3] James A. Whittaker, Herbert H. Thompson, Herbert Thompson. How to Break Software Security, Addison Wesley, May 9, 2003.

[4] Joshep Yoder and Jeffrey Barcalow. Architectural patterns for enabling application security. PLoP, 1997. 5, 6,7.

[5] D. C. Schmidt. Patterns, Frameworks and Middleware: Their Synergetic Relationships. Invited talk at IEEE/ACM International Conference on Software Engineering, Portland, Oregon, May 3--10, 2003.

[6] Kienzle, D.M., Elder, M.C., "Final Technical Report: Security Patterns for Web Application Development." (Available at http://www.scrypt.net/~celer/securitypatterns/final%20report.pdf).

[7] IBM's Security Strategy team. (2004). "Introduction to Business Security Patterns. An IBM White Paper". Available at http://www-3.ibm.com/security/patterns/intro.pdf. 2004.

[8] Konrad, S., B.H.C. Cheng, Campbell, Laura. A., and Wassermann R., (2003). "Using Security Patterns to Model and Analyze Security Requirements". Proc. Requirements for High Assurance Systems Workshop (RHAS '03).

[9] Yoder, J. and Barcalow, J. (2000). "Architectural Patterns for Enabling Application Security". Pattern Languages of Program Design. 4, 301-336. Reading, MA: Addison Wesley Publishing Company.

[10] Fernandez, E.B., (2000). "Metadata and authorization patterns". Technical report, Florida Atlantic University

[11] Romanosky, S., (2001). "Security Design Patterns", Part 1, 1.4.

[12] Fernandez, E.B. and Pan, Rouyi, (2001). "A pattern language for security models". PLoP'01 Conference.

[13] Torsten, P, Fernandez, E.B., Mehlau, J.I., Pernul, G. (2004). "A pattern system for access control". 18th IFIP WG 11.3 Conference on Data and Applications Security, (Sitges, Spain).

[14] Wassermann, R. and Cheng, B.H.B., (2003). "Security patterns". Technical Report MSU-CSE-03-23, Computer Science and Engineering, Michigan State University, (East Lansing, Michigan).

[15] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). "Design patterns: Elements of Reusable Object-Oriented Software". Addison-Wesley, 1994

[16] Delessy-Gassant, N., Fernandez. E.B., Rajput. S, and Larrondo-Petrie, M.M., (2004). "Patterns for Application Firewalls". PLoP'04.

[17] Essmayr, W., Pernul, G., Tjoa, A.M., (1997). "Access controls by object oriented concepts". Proceedings of 11th IFIP WG 11.3 Working Conference on Database Security.

[18] Fernandez, E. B., (2004). "Two patterns for web services security". Proc. International Symposium on Web Services and Applications (ISWS'04), (Las Vegas, NV).

[19] Mouratidis, H., Giorgini, P., Schumacher, M., (2003). "Security Patterns for Agent Systems". In Proceedings of Eighth European Conference on Pattern Languages of Programs, (Irsee, Germany).

[20] Hallstrom, J. O., Soundarajan, N., Tyler, B. (2004). "Monitoring Design Pattern Contracts". In Proc. of the FSE-12 Workshop on Specification and Verification of Component-Based Systems. pg. 87-94.

[21] Allenby, K., and Kelly, T. (2001). "Deriving Requirements Using Scenarios". In Proc. Of the 5th IEEE International Symposium on Requirements Engineering. RE'01.

[22] Hallstrom, J. O., Soundarajan, N. (2006). "Pattern-Based System Evolution: A Case-Study". In the Proc of the 18th International Conference on Software Engineering and Knowledge Engineering. San Francisco Bay, USA.

[23] Mikkonen, T. (1998). "Formalizing design patterns". In Proc. Of 20th ICSE, pages 115-124. IEEE Computer Society Press.

Ensuring Consistent Use/Misuse Case Decomposition for Secure Systems

Josh Pauli
College of Business and Information Systems
Dakota State University
Madison, SD, 57042, USA
josh.pauli@dsu.edu

Dianxiang Xu
Department of Computer Science
North Dakota State University
Fargo, ND, 57105, USA
dianxiang.xu@ndsu.edu

Abstract

Misuse case modeling is a viable option to depict the security requirements together with functional requirements. We then use decomposition to investigate the interplay between functional and security requirements, thus creating a complete set of security-centric requirements that can guide subsequent software development phases. Part of the initial decomposition is to identify relationships ("includes" and "extends") among decomposed cases for each case type (use, misuse, mitigation use). Decomposition is conducted for each case type independently and then integrated with the "threatens" and "mitigates" relationships where misuse cases "threaten" use cases and mitigation use cases "mitigate" misuse cases. Consistency must be maintained as decomposition continues and additional cases are created so that the initial relationships of each case type are preserved. Relationship specific conditions must be met for a relationship to be present among the decomposed cases. We introduce a condition-driven process to ensure consistency of decomposition past the initial identification and modeling of the relationships present among the three cases types.

Keywords: Security requirements, decomposition, use case, misuse case, consistency.

1. Introduction

Since the wide spread adoption of use case modeling for functional requirements, misuse case modeling has became a viable option to depict the security requirements of the system [1]. We use misuse cases and decomposition to create a set of requirements that is concerned with security as well as the functional requirements. More details are needed than can be expressed by a high level use, misuse, and mitigation use case model; this is why we decompose. We model functional requirements with use cases and use the STRIDE threat categories from the threat modeling approach [14] for misuse cases. The interplay between misuse cases and use cases drive the identification of mitigation use cases that preserve the goals of security. These models are then systematically decomposed to allow the details of each case to be investigated as well as the interplay among the three case types.

Part of our decomposition approach is to identify and model relationships ("includes" and "extends") in each case type as part of the initial decomposition [10]. Conditions for each relationship must be met in order for a relationship to be present. Once identified as being present, we introduce heuristics to accurately and consistently model the relationship among the decomposed cases. Actor assignments must also be made in a consistent manner for each decomposed case. This is addressed in this paper by introducing heuristics that maintain the relationships and assignments among decomposed case subsets. These heuristics are applicable at any level of abstraction after the initial identification and modeling.

To demonstrate our approach, a Payroll Information System (PIS) [9][10] is discussed throughout the paper. The PIS is a web-based system that general employees access for payroll-related information such as pay rate, deductions, and benefits. Payroll staff personnel use the PIS to complete various tasks such as entering, requesting, and retrieving employee payroll-related information. In order to label the cases accordingly we used a prefix of "U" for use, "M" for misuse, and "S" for mitigation (security) use and a numbering system that follows requirements specification standards with requirement 1 being decomposed into 1.1, 1.2, and so on. Each case also has a descriptive title that summarizes the details that are fully documented in the textual description.

Our paper is organized as follows. Section 2 reviews related work. Section 3 introduces our decomposition process while section 4 covers the maintenance of consistency during decomposition. Section 5 reveals PIS case study results and we conclude in section 6.

2. Related Work

Numerous approaches in the requirements phase have addressed software security including scenario-driven requirements analysis [2], misuse case-driven elicitation of non-functional requirements [1][11] goal-driven approaches to user requirements [6], abuse-frame oriented security requirements analysis [8], anti-goals [16], abuse frames, and reuse-based approaches to determining security requirements [12]. The anti-goal method in the goal-oriented requirements analysis can describe vulnerabilities or generate attacks that violate security goals [16]. The "actor, intention, goal" approach in the i* framework [17] models security and trust relationships as "software goals": goals depending on another actor to satisfy them. While our work is built upon the notions of misuse and mitigation use cases in the misuse case method, we exploit decomposition as a way to uncover additional details about the interplay among use cases, misuse cases, and mitigation use case at different levels of abstraction. The software security problem has also been investigated in the design phase of development with numerous other approaches including architecture-driven benefits [7], user-centered design of secure software [5], and threat modeling [14].

Smith realized that decomposition was certain if the system was of decent size, but the criteria and engineering process are important because ad-hoc efforts are not good enough [13]. Decomposing to a level of being able to solve the problem is the goal, but there is no current way to account for non-functional requirements [13]. Constantine and Lockwood stated that decomposition has several major advantages [4]. First, it leads to a smaller, simpler use case model in which each part is comparatively simple and easy to understand in itself and in relation to a small number of other use cases [4][2]. Second, the total model is simplified through reuse because common elements are factored out into separate use cases rather than repeated numerous places and variations [4]. Each piece of a problem can be solved once and the solution can be recycled wherever the corresponding use case is referred in the model [4]. Tohidi stated that each decomposed use case represents a simple, unified, unit of work that is important for the success of a usage centered design project [15]. Allenby and Kelly argue that capturing requirements is based primarily on core requirements which are discharged by decomposition and allocated across many subsystems [2].

3. An Overview of Use Case Decomposition

The interactive modeling of uses, misuses, and mitigations has had recent research applied to it [1] [3][4][5][6][9][11][13], which we use to initially model requirements. We use the misuse case approach as originally intended with additions to further strengthen the requirements such as custom-made templates for the textual descriptions of all of the use, misuse, and mitigation use cases[10]; refer to [9][10] for examples. The textual description for each case documents a level of abstraction. We use the STRIDE threat categories from the threat modeling approach for identifying threats and suggesting mitigations [14]. We use detailed threats of each of these categories, excluding those irrelevant to requirements engineering such as buffer overflow vulnerabilities. The identified misuse cases threaten one or more of the goals of security (confidentiality, integrity, and availability, authentication, and non-repudiation). The mitigation cases prevent the misuse cases, thus preserving these goals. Misuse cases "threaten" use cases, while mitigation use cases "mitigate" misuse cases. This is the interplay that we investigate for details to ensure security is a driving force of system development.

Our decomposition is driven by the textual descriptions of the cases where each step may become a new case at the more detailed level of abstraction. We decompose each case type individually to ensure proper decomposition details before integration with the other case types. Our relationship heuristics should only be used after a textual description has been written for each case [3][11].

To identify shared cases we created heuristics for the "includes" relationship so that these cases can be initially identified and modeled appropriately [10] when two or more cases have common sections and there is a need to separate this section into a separate case to be referenced. These heuristics ensure that any shared use case will be initially identified and modeled accurately [10]. We use the "extends" relationship from use case modeling to model optional behavior [3][10]. The "extended" use case is fully aware of the base use case which is not always the case in the "includes" relationship. We do not use the "generalization" relationship from use case modeling as it is concerned only with actors [3]. Conditions must be met for a relationship to be present between two decomposed cases. If any one condition is true, then a relationship exists between the cases and must be modeled accurately. When two or more cases share a pre-condition or share execution steps an "includes" relationship is present. When a case has optional executions within a step then an "extends" relationship is present.

We use the textual descriptions of each use case to identify which use cases need an actor assigned [10]. If the textual description step indicates that the user executes the use case, then that assignment is made. If the user is not present in the textual description step, then relationships with the other use cases are left unchanged.

Heuristics for the "threatens" relationship were developed in order to account for the interaction between use cases and misuse cases as decomposition occurs [10].

We must ensure each decomposed misuse case is threatening the decomposed use case accurately. Information from the use and misuse case textual description is used in this mapping. The threats documented for each use case is used to map to specific decomposed misuse cases based on the information contained in the misuse case textual descriptions. The "threatens" relationship is applicable in three situations. The first situation is when the misuse case is decomposed and the use case is not; the second situation is when both the misuse case and use case are decomposed; the third situation is when the use case is decomposed and the misuse case is not. These situations allow either the use case or misuse case to be modeled in a decomposed state or the original, higher level state. We ensure that the relationships between use cases and misuse cases at varying levels of abstraction have been consistently maintained.

"Mitigates" relationships detail exactly what security measures prevent each misuse case. The detailed information within the textual descriptions for the misuse cases and mitigation use cases are used for this relationship. The "mitigated/prevented by" column from the misuse case textual description is used to tie misuse cases to mitigation use cases. The "mitigates" relationship is also applicable for three situations where both case types may be modeled in a decomposed state or at the higher level of abstraction. The "threatens" and "mitigates" relationship heuristics create a complete model as introduced in Figure 1 where the detailed interactions among uses, misuses, and mitigations are depicted. When decomposition continues past this initial modeling, the relationships that are present within each case type must be consistently maintained so that the interactions among uses, misuses, and mitigations can be maintained as decomposition creates more detailed cases. This is the issue addressed by this paper.

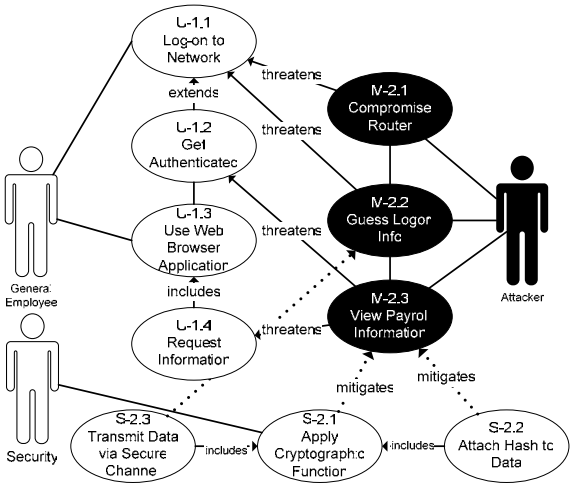


Figure 1. Complete Decomposed Model

4. Ensuring Consistent Decomposition

We must be able to maintain the relationships that were initially identified as being present in each case during the initial decomposition. The "includes" and "extends" relationship as well as actor assignments must be maintained as decomposition continues and additional cases are created. This section introduces the process needed to maintain these relationships between case subsets as decomposition continues as introduced in .

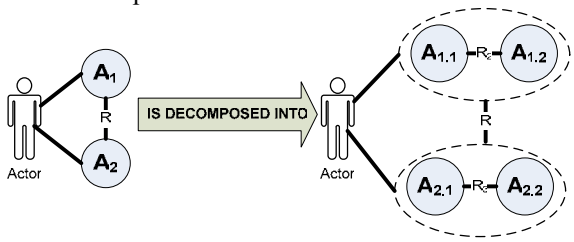


Figure 2. Relationships in Decomposed Cases

The relationship (R) in figure 2 was first identified in the initial decomposition process. When decomposition continues the 'A_1' and 'A_2' cases are decomposed into additional cases. R persists between these two cases, but the originating case in the 'A_2' case subset and the targeted case in the 'A_1' case subset are unknown. To ensure that the appropriate cases are used in the relationship we developed conditions that mandate the originating and targeted cases. These conditions are listed below.

In order to identify the targeted case:

#1: IF "R2" = "includes" THEN "R" target = "R2" originator

#2: IF "R2" = "extends" THEN "R" target = "R2" target

#3: IF "R2" = NONE THEN "R" target = the first "A1" case

In order to identify the originating cases:

#4: IF "R3" = "includes" THEN "R" originator = "R3"originator

#5: IF "R3" = "extends" THEN "R" originator = "R3" target

#6: IF "R3" = NONE THEN "R" originator = the first "A2" case

These conditions are mutually exclusive and completely exhaustive. This means that at least one of the first three conditions and one of the last three conditions must be met and that every possible scenario is covered by our conditions. We use the "M-1" decomposed misuse case as an example of using these conditions. In Figure 3 there are two "includes" relationships present among the case subsets. The "M-1.2" misuse case subset is creating traffic, the "M-1.3" misuse case subset is flooding the system server, and the "M-1.4" misuse case subset is

creating long running requests. These two relationships need guidance as to what "M-1.2" case is targeted by this relationship and which cases from the "M-1.3" and "M-1.4" case subsets originate the relationships.

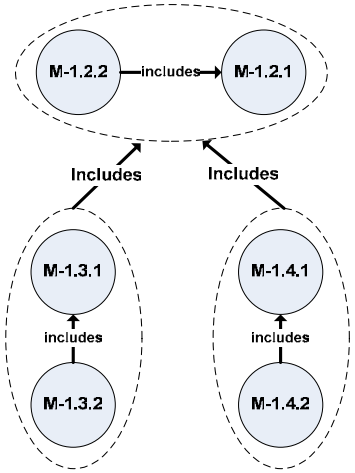


Figure 3. "includes" Relationship in Decomposed Cases

The "M-1.2" case subset is targeted by both "includes" relationships and must meet one of the three conditions that mandates the targeted case. The "includes" relationship within the "M-1.2" case subset is depicted by "R2" in our conditions. This relationship meets the first condition that mandates "R" targets the originator of "R2". Therefore both "includes" relationships target the "M-1.2.2" misuse case.

To identify the originating cases of the "includes" relationships the relationships within the "M-1.3" and "M-1.4" case subsets dictate the originating cases. The "includes" relationships within the "M-1.3" and "M-1.4" case subsets meet the fourth conditions which mandates the originating cases for "R" is the originating cases of "R3". Therefore the "M-1.3.2" and "M-1.4.2" cases originate "R". A model that reflects these conditions is introduced in Figure 4.

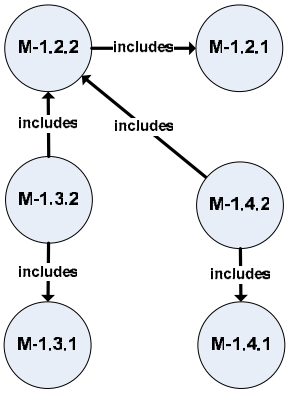


Figure 4. Identified Targets and Originators

Actors need to be assigned after the decomposed cases have been accurately identified as taking part in the relationships. We use the steps in the textual descriptions to dictate what actor executes each step (case). We inspect each step to see if a human actor executes that step. If they do, then an actor assignment is made. If a human does not execute that step, then the case is considered independent of the cases and a system actor is needed. System actors are any means of case execution that does not include humans such as a system component, system timing, or other triggers. The steps for the "M-1.2", "M-1.3", and "M-1.4" misuse cases are listed in Table 1.

Table 1. Actor Assignments for "M-1" Misuse Cases

Case ID	Case Description	Human Actor?
M-1.2.1	Attacker Coordinates Attack Machines	Yes
M-1.2.2	Attacker Sends Simultaneous Commands	Yes
M-1.3.1	Attack Consumes System Bandwidth	No
M-1.3.2	Attack Consumes System Resources	No
M-1.4.1	Attack Ties-up Web Server Resources	No
M-1.4.2	Attack Times-out Web Server	No

For each case that was executed by a human ("Yes" in the last column), an actor assignment is made in the final misuse case model. A system actor is assigned to the cases that are not executed by a human. The system actor in this misuse case example is "System Timing" because these independent cases rely upon the timing of the attack in order to execute. A final misuse case model is introduced in Figure 5.

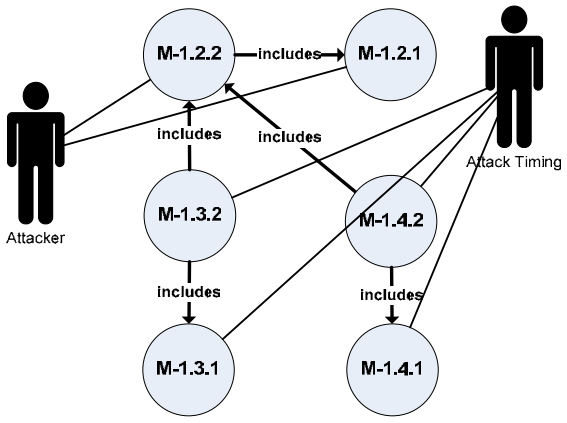


Figure 5. Misuse Case Model with Actor Assignments

We created conditions to drive the maintenance of relationships among cases as decomposition continues.

The "includes" and "extends" relationships, and actor assignments that were first made in the initial decomposition process must be maintained as more detailed models are created. Each case type is applicable for these conditions when further decomposition creates additional models. These conditions provide guidance as to which cases originate the relationships and which cases are targeted by these relationships. This ensures that the relationships modeled at a higher level of abstraction are consistently maintained in the more detailed models.

We also maintained the actor assignments which change as decomposition creates more cases. Some cases are not executed by the human actor and need a system actor to execute. The system actors for misuse and mitigation use cases differ from the system actors for use cases because there is no system to execute misuse or mitigation use cases in. Instead we use timing as the system actor for misuse and mitigation use cases because in most cases if a misuse or mitigation is not executed directly by the human actor then the case is dependent on the timing of the actual misuse or mitigation.

5. PIS Case Study

We completed a complete PIS case study to illustrate our entire decomposition approach with initial high level use cases, misuse cases, and mitigation use cases that were then decomposed independently to identify the relationships within each case type. These decomposed models were then integrated with "threatens" and "mitigates" relationships as introduced in Figure 1. In order to decompose further we created the conditions that we introduced in this paper as a guide to ensure consistency after the initial identification and modeling for each case type. While this case study was narrow in its scope, the amount of cases that were created from decomposition grew rapidly from the initial models that included two use cases, 2 misuse cases, and 2 mitigation use cases.

In order to track each of the case types, we constructed tables that list each high level use case and the cases that were created from decomposition. First we partially list the cases from the PIS in Table 2 while

Table 3 lists three cases from the "U-1: Complete Administrative Tasks" use case at varying levels of abstraction, what misuse cases threatens each, and the mitigation cases necessary to mitigate the threats.

Table 2 illustrates the interplay between uses, misuses, and mitigations at two levels of abstraction. The heuristics introduced in this paper enable the consistent decomposition from the initial decomposition (from U-1 to U-1.3, U-1.4, and so on) to the more detailed models (U-1.3.1, U-1.4.1, U-1.4.2, and so on) for each case type.

Table 2. Titles for Decomposed Cases

ID	Title
U-1.3	PIS staff uses web browser application
U-1.3.1	PIS staff selects task
U-1.4	General user requests general information
U-1.4.1	General user executes request
U-1.4.2	System retrieves general information
U-1.5	PIS staff enters information
U-1.5.1	PIS staff enters hours worked
U-1.5.2	PIS staff assigns deductions
U-1.5.3	System calculates gross pay, deductions, net pay
U-1.6	PIS staff retrieves payroll information
U-1.6.1	PIS staff submits payroll information request
U-1.6.2	System retrieves deductions, pay rate, or W-2 Info.
M-1.2	Attacker creates traffic
M-1.3	Attacker floods server
M-1.4	Attack creates long running requests
M-2.1	Attacker compromises router
M-2.2	Attacker guesses logon
S-1.2	Allow authentic requests
S-1.2.1	Allow recognized requests
S-1.2.3	Enforce resource limits
S-2.1	Apply cryptographic function
S-2.1.1	Generate key
S-2.1.2	Encrypt data

Because of space constraints we can not show every decomposed case, but the format of Table 2 was used for every level of abstraction for each use case.

Table 3. Relationships Among PIS Cases

Use	Mis	Miti		Use	Mis	Miti
U-1.3	M-2.2	S-2.1		U-1.5.1	M-1.4	S-1.2.1, S-1.2.2, S-1.2.3
U-1.3.1	M-2.2.1	S-2.1.1 S-2.1.2		U-1.5.2	M-1.4	S-1.2.1, S-1.2.2, S-1.2.3
U-1.4	M-1.2, M-1.3, M-1.4, M-2.1	S-1.2 S-2.1		U-1.5.3	M-1.4	S-1.2.1, S-1.2.2, S-1.2.3
U-1.4.1	M-1.4	S-1.2.1, S-1.2.2, S-1.2.3		U-1.6	M-1.2, M-1.3, M-1.4, M-2.1	S-1.2 S-2.1
U-1.4.2	M-1.4	S-1.2.1, S-1.2.2, S-1.2.3		U-1.6.1	M-1.4	S-1.2.1, S-1.2.2, S-1.2.3
U-1.5	M-1.2, M-1.3, M-1.4, M-2.1	S-1.2 S-2.1		U-1.6.2 U-1.6.3 U-1.6.4	M-1.4	S-1.2.1, S-1.2.2, S-1.2.3

Generally, decomposing to these levels is sufficient to uncover all of the details of the cases without prematurely making design decisions in the requirements phase. Over decomposition is a concern with our approach as we do

not want to make premature design decisions in the requirements phase.

The findings from this case study help validate our approach to decomposing for the benefit of security requirement specification in the requirement phase by identifying uses, misuses, and mitigations early in the development process.

6. Conclusions

Our decomposition process is driven by the textual descriptions of each case and includes extensive work on identifying and modeling the "includes" and "extends" relationships accurately and consistently. We created a decision making process which includes steps and conditions to identify and model these relationships as part of the initial decomposition of each case. We follow conditions that make an "includes" or "extends" relationship likely among the newly decomposed cases and apply our steps and conditions to ensure the appropriate relationships among the decomposed cases.

We ensure accurate and consistent relationships by creating a condition-driven process for relationship maintenance. After the first iteration of decomposition occurs there is a need to maintain the "includes" and "extends" relationships that were identified and modeled initially. The presence of these relationships in the case subsets has a direct impact on which case is the originator of the relationship and which case is the target of the relationship. We developed a strict set of conditions that mandates exactly which case originates and is targeted by the "includes", "extends", "threatens", and "mitigates" relationship when there are "includes" and/or "extends" relationships within the case subsets. Accurate relationships can be modeled as decomposition continues by following the conditions we developed to deal with relationship maintenance.

Integration is the main reason for decomposing the cases; we are not trying to create a functional decomposition of use cases. When the decomposed cases are integrated the detailed interactions between them are documented to aid in the specification of the security requirements. Integration allows security requirements to be investigated and specified earlier in the development lifecycle when compared to other approaches to security requirements. We are able to create a set of requirements that is based on the detailed interactions among uses, misuses, and mitigations that can be used as system development moves forward. No longer will only functional requirements be investigated in the requirements phase separate from the security requirements.

References

[1] I. Alexander. "Misuse Cases Help to Elicit Non-Functional Requirements" *Computing & Control Engineering Journal,* vol. 14, no. 1, pp. 40-45, Feb. 2003.

[2] K. Allenby and T. Kelly. "Deriving Requirements Using Scenarios". In *Proc. of the 5th IEEE International Symposium on Requirements Engineering (RE'01)*, Aug. 2001.

[3] K. Bittner and I. Spence. *Use Case Modeling,* Boston: Addison-Wesley, 2003.

[4] L. Constantine and L. Lockwood. *Structure and Style in Use Cases for User Interface Design,* 2000.

[5] D. Firesmith. "Security Use Cases". *Journal of Object Technology*, Vol. 2, No. 3, 53-64. (May-June 2003).

[6] I. Jacobson, M. Christerson, P. Jonsson, and G. Overgaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison Wesley, 1994.

[7] R. Kazman, G. Abowd, and L. Bass. "Scenario-Based Analysis of Software Architecture". *IEEE Software,* pp. 47-55, Nov. 1996.

[8] K. Khan and J. Han. "Composing Security-Aware Software". *IEEE Software,* pp. 34-41, Jan.-Feb. 2002.

[9] J. Pauli and D. Xu. "Threat-Driven Architectural Design of Secure Information Systems". In *Proc. of the 7th International Conference on Enterprise Information Systems (ICEIS'05)*, pp. 136-143. Miami, May 2005.

[10] J. Pauli and D. Xu. "Integrating Functional and Security Requirements with Use Case Decomposition". Submitted to *11th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2006)*. Palo Alto, August 2006.

[11] G. Sindre and A.L. Opdahl. "Templates for Misuse Case Description". In *Proc. of the 7th International Workshop on Requirements Engineering, Foundation for Software Quality (REFSQ'2001)*, June 2001.

[12] G. Sindre, D. Firesmith, and A. Opdahl. "A Reuse-Based Approach to Determining Security Requirements" In *Proc. 9th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'03),* pp. 127-136, 2003.

[13] J. Smith. "The Estimation of Effort Based on Use Cases". *Rational Software White Paper,* Oct. 1999.

[14] F. Swiderski and W. Snyder. *Threat Modeling*. Microsoft Press, 2004.

[15] M. Tohidi. "Task Modeling". *Directed Studies Research Honors Project,* Carleton University, Nov. 2003.

[16] A. Van Lamsweerde. "Elaborating Security Requirements by Construction of Intentional Anti-models". In *Proc. of the 26th International Conference of Software Engineering (ICSE'04)*, pp.148-157, 2004.

[17] E. Yu and L. Liu. "Modeling Trust for System Design Using the i* Strategic Actors Framework", in R. Falcone, M.P. Singh and Y.H. Tan (eds.), *Trust in Cyber-Societies, Integrating the Human and Artificial Perspectives*, pp. 175-194, 2001.

Improving Intrusion Detection Systems Using Reference Vectors

Ohm Sornil[1] and Pattree Sidthikorn[2]

[1,2]School of Applied Statistics, National Institute of Development Administration
118 Seri Thai Road Bangkapi Bangkok, Thailand 10240
[1]osornil@as.nida.ac.th, [2]pattree_s@as.nida.ac.th

Abstract

An intrusion detection system is designed as a second line of defense to detect intrusions which circumvent protection mechanisms. A number of studies have proposed techniques for this purpose. This paper presents an inclusion of a new set of attributes derived from reference vectors to improve the performance of intrusion detection systems. The performance evaluation demonstrates that models using attack-based reference vectors outperform other configurations in all situations with different number of attack types. The detection rate against 22 attack types is 96.54% with 5.37% false alarm rate.

1. Introduction

Nowadays the world is connected by computer networks. The growing number of connections comes more potentials of damages from security attacks. Attack prevention mechanisms such as firewalls and hardening system components aim to prevent attackers from causing damages. Preventive measures are likely to fail due to the difficulty of covering every possible vulnerability and attack that can happen to a system. A second line of defense is required to detect attempts to break into or misuse the system. An intrusion detection system (IDS) inspects all inbound and outbound network activities and identifies suspicious patterns that may indicate a network or system attack by someone attempting to break in or compromise the system.

At the heart of an intrusion detection system (IDS) lies the ability to distinguish normal activities from unauthorized ones. Two major approaches to the problem are anomaly detection and misuse detection. Anomaly detection identifies activities that vary from established patterns of normal users. It typically involves creation of knowledge from profiles of normal activities. However, this approach may not fully reflect the complex, dynamic nature of computer systems and users, thus can cause lots of false alarms. Misuse detection involves comparison of a user's activities with the known behaviors of attacks attempting to penetrate a system. This approach focuses on a smaller scope (attacks only), thus can attain high levels of accuracy. However, this approach is vulnerable to novel attacks whose patterns are not known to the system. Due to the complementary nature of the two approaches, many systems combine both of these techniques.

A variety of techniques have been devised to detect intrusions. Anomaly detection includes techniques like setting heuristic limits of event occurrences over a time interval [3], Markov process modeling of events, and event clustering [1]. Techniques exploited in misuse detection range from state transition analysis of attacking events [11] to genetic algorithms [9].

Neural network is one of the widely used techniques for detecting intrusions. It does not rely on preconceptions about the behaviors of the monitored system, and its learning ability allows compensation for behavior drifts. Ryan [12] applies a neural network to an off-line host-based anomaly detection system, trained by everyday's user commands. Lee [8] uses a hierarchy of neural networks to detect anomalies by polling data from both hosts and networks. The system comprises multiple levels of neural networks where a network in an upper level examines results from those in lower levels. At the lowest level, a set of small neural networks monitor signals in the network. The results indicate benefits of this type of organization. Zhang [14] proposes an agent-based architecture to detect anomalies in a network. Small IDSs act like agents stationed in various parts of the network, e.g., servers, bridges, routers, and firewalls. The results from these agents are sent to and put together at the IDS center to examine the signs of attacks.

Mukkamala [10] compares the performance of Support Vector Machine (SVM) and Multilayer Backpropagation neural networks with various types of attacks. The results show that SVM yields a higher accuracy. Jirapummin, *et al.* [5] use a combination of a Resilient Propagation Algorithm (RPROP) together with Self-Organizing Map (SOM) as an intrusion classifier.

In this paper, we propose an improvement to classification models in the context of intrusion detection through the use reference vectors. The rest of this paper is organized as follows. Section 2 presents the proposed technique. Section 3

discusses the performance evaluation against a standard test collection. Section 4 provides concluding remarks.

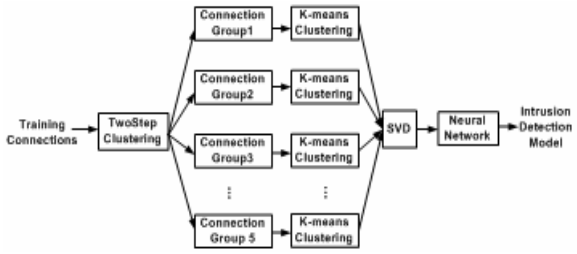


Figure 1. The proposed intrusion detection process

2. The Proposed Intrusion Detection Approach

A network connection is expressed as a vector $C_j = (f_{1,j}, f_{2,j}, \ldots, f_{s,j})$ where $f_{i,j}$ is the value of attribute i in connection j. Let s be the number of connection attributes and n be the number of connections in the training data set. The overall process is shown in Figure 1.

First the TwoStep clustering algorithm is performed to generate an initial set of clusters. An important characteristic of this algorithm lies in its abilities to handle a large data set and determine the final number of clusters automatically. The resulting clusters are then used to guide the grouping of attacks according to a criterion described below. Next K-means clustering [2] is performed for each attack group. Each attack group will then have K cluster centroids as reference vectors. For each connection, a set of Euclidean distances between the connection vector and each reference vector are calculated.

These distances are added into the list of connection attributes. The augmented connection-attribute matrix is then passed through a matrix reduction process to reduce noises and discover latent relationships among attributes. Finally, the compressed, augmented connection-attribute matrix is used to create a Backpropagation neural network model for detecting intrusions.

2.1 Connection Clustering

TwoStep clustering is employed to create an initial set of clusters. The algorithm consists of two steps: (1) pre-clustering connections into many small sub-clusters; and (2) clustering the sub-clusters resulting from the pre-cluster step automatically into a number of clusters. The TwoStep algorithm discussed here is described in [2].

The pre-clustering procedure is implemented by constructing a modified cluster feature (CF) tree [13]. A CF tree consists of levels of nodes, and each node contains a number of entries. An entry in a leaf node represents a final sub-cluster. The non-leaf nodes and their entries are used to guide a new record quickly into a correct leaf node. Each entry is characterized by its CF that consists of its number of records, mean and variance of each range field, and frequency for every category of each symbolic field.

For each successive record, starting from the root node, it is recursively guided by the closest entry in the node to find the closest child node and descends along the CF tree. Upon reaching a leaf node, it finds the closest leaf entry in the leaf node. If that record is within a threshold distance of the closest leaf entry, it is absorbed into the leaf entry, and the CF of that leaf entry is updated. Otherwise it starts its own leaf entry in the leaf node. If there is no space in the leaf node to create a new leaf entry, the leaf node is split into two. The entries in the original leaf node are divided into two groups using the farthest pair and redistributing the remaining entries based on the closeness criterion.

All records falling in the same entry can be collectively represented by the entry's CF. When a new record is added to an entry, the new CF can be computed from this new record and the old CF without knowing the individual records in the entry. These properties of CF make it possible to maintain only the CFs, rather than the sets of individual records. Hence, the CF-tree is much smaller than the original data and can be stored in memory more efficiently. The distance between clusters i and j is defined as: $d(i,j) = \xi_i + \xi_j - \xi_{(i,j)}$

where $\xi_\upsilon = -N_\upsilon \left(\sum_{k=1}^{K^A} \frac{1}{2} \log\left(\hat{\sigma}_k^2 + \hat{\sigma}_{\upsilon k}^2 \right) + \sum_{k=1}^{K^B} \hat{E}_{\upsilon k} \right)$ and

$$\hat{E}_{\upsilon k} = -\sum_{l=1}^{L_k} \frac{N_{\upsilon kl}}{N_\upsilon} \log \frac{N_{\upsilon kl}}{N_\upsilon}$$

which

K^A is the number of range type input fields,

K^B is the number of symbolic type input fields,

L_K is the number of categories for the k^{th} symbolic field,

N_υ is the number of records in cluster υ,

$N_{\upsilon kl}$ is the number of records in cluster υ which belongs to the l^{th} category of the k^{th} symbolic field,

$\hat{\sigma}_k^2$ is the estimated variance of the k^{th} continuous variable for all records,

$\hat{\sigma}_{\upsilon k}^2$ is the estimated variance of the k^{th} continuous variable for records in the υ th cluster, and

(i, j) is an index representing cluster formed by combining clusters i and j.

The cluster step takes sub-clusters resulting from the pre-cluster step as input and then groups them into a number of clusters. Since the number of sub-clusters is much less than the number of original records, traditional clustering methods can be used effectively. An agglomerative hierarchical clustering method is then carried out in two steps.

In the first stage, the Bayesian information criterion (BIC) for each number of clusters within a specified range is

calculated and used to find the initial estimate for the number of clusters. The BIC is computed as:

$$BIC(J) = -2\sum_{j=1}^{J}\xi_j + m_j \log(N)$$

where $m_j = J\{2K^A + \sum_{k=1}^{K^B}(L_K - 1)\}$.

The ratio of change in BIC at each successive merging relative to the first merging determines the initial estimate. Let *dBIC(J)* be the difference in BIC between the model with *J* clusters and that with *(J+1)* clusters, *dBIC(J) = BIC(J)-BIC(J+1)*. The change ratio for model *J* is calculated as $R_1(J) = dBIC(J) / dBIC(1)$.

If $dBIC(1) < 0$, then the number of clusters is set to 1 (and the second stage is omitted). Otherwise, the initial estimate for number of clusters *k* is the smallest number for which $R_1(J) < 0.04$.

In the second stage, the initial estimate is refined by finding the largest relative increase in distance between the two closest clusters in each hierarchical clustering stage. Starting with the model C_K indicated by the BIC criterion, take the ratio of minimum inter-cluster distance for that model and the next large model C_{k+1}, that is, the previous model in the hierarchical clustering procedure as: $R_2(k) = \frac{d_{\min}(C_k)}{d_{\min}(C_{k+1})}$ where C_k is the cluster model containing *k* clusters and $d_{\min}(C)$ is the minimum inter-cluster distance for cluster model *C*. From model C_{k-1} compute the same ratio with the following model C_k. Repeat for each subsequent model until reach ratio $R_2(2)$.

Compare the two largest R_2 ratios; if the largest is more than 1.15 times the second largest, then select the model with the largest R_2 ratio as the optimal number of clusters; otherwise, from those two models with the largest R_2 values, select the one with the larger number of clusters as the optimal model.

2.2 Reference Vector Creation

In Figure 2, we can see that some attacks have a wide variants of connection characteristics (e.g., portsweep and neptune), and connections of some attack types are very close to those of other attacks, for instance, postsweep connections appear to be mixing with all other connections. This characteristic makes it difficult to identify specific types of attacks, given a connection. In addition, a large part of normal connections are very close to portsweep and neptune connections, making it difficult to separate these attacks from normal connections. With more attack types in consideration, this situation will be aggravated.

Cluster labels have been used in various research to enrich attributes in classification problems. However, connections of the attacks with wide variants of characteristics (such as normal and neptune) are likely to scatter in different clusters. Thus, label information may not be much helpful in improving the detection.

Distances between a connection and some points in the space (referred to as reference vectors) are potentially better attributes than cluster labels since they reflect the relative distances of this connection from locations in the space. Positions of these vectors are crucial for the detection performance. Two types of reference vector positioning are studied in this research: attack-based and similarity-based reference vectors.

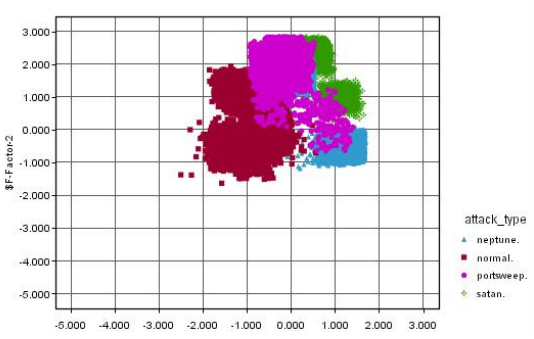


Figure 2. A scatter plot of three attack and normal connections in a two-dimensional space reduced using Singular Value Decomposition.

Attack-Based Reference Vectors (ARV)

In attack-based reference vector approach, an attack remains in its initial cluster only if a majority, say at least 70%, of that attack's connections fall into that cluster. Otherwise, all connections of the attack become a new connection group. Grouping connections in this fashion isolates and emphasizes attacks with wide variants by creating reference vectors which reflect the characteristics of those attacks more closely.

Cluster-Based Reference Vectors (CRV)

In cluster-based reference vector approach, an initial cluster is treated as a connection group.

After connection groups are formed, K-means clustering is performed for each group, generating *K* subgroups of connections whose centroids represent reference vectors. Euclidean distances between a network connection and all of reference vectors are used to augment the original set of attributes and improve the effectiveness of intrusion detection models.

2.3 Network Attack Modeling

Let A_{nm} be an augmented connection-attribute matrix with *n* rows and *m* columns whose attributes consist of connection-based features together with the new distance attributes. The organization of a connection-attribute matrix assumes that all attributes are independent which is generally

not true in practice. Also, with a large number of attributes, further calculations are computationally expensive due to the high dimensionality.

Connection-Attribute Matrix Compression

Singular Value Decomposition (SVD) is employed in this research to compress the matrix into a lower dimensional attribute space which can uncover hidden relationships among attributes and connections, and reduce the effects of noises in connection characteristics [8].

SVD decomposes matrix *A* into three components: the orthogonal matrix of singular values, where *r = min(m, n)*, and the left and right matrices (i.e., *U* and *V*, respectively), as shown in Figure 3.

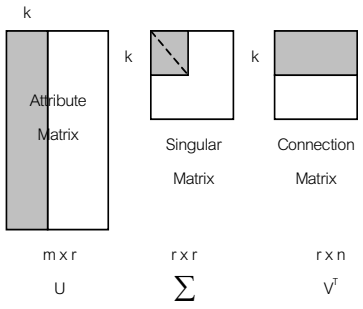


Figure 3. Singular Value Decomposition of an augmented connection-attribute matrix

By keeping $k < r$ largest singular values of the singular matrix along with their corresponding columns in *U* and *V*, the resulting matrix is a matrix of rank *k* which is closest to the original matrix *A* in the least square sense. With respect to this new space of *k* dimensions, the original attributes are no longer independent from each other.

Backpropagation Neural Network

A backpropagation neural network [6] is a structure of small processing units, called neurons, connected in a systematic way. The neural networks used are backpropagation feedforward neural networks, also known as multilayer perceptrons (MLP). The neurons are arranged in layers. Typically, there is one layer for input neurons (the input layer), one or more hidden layers for non-linear mapping, and one layer for output neurons (the output layer). Each layer is fully interconnected to the following layer.

The connections between neurons have weights associated with them, which determine the strength of influence one neuron has on the other. Information flows from the input layer through the hidden layer(s) to the output layer to generate predictions. By adjusting the connection weights during training to match predictions to target values for specific records, the network "learns" to generate better and better predictions. The computation of a backpropagation neural network consists of two passes: forward pass and backpropagation pass. The description here follows [2].

Forward Pass: Input neurons have their activations set to the values of the encoded input fields. The activation of each neuron in a hidden or output layer is calculated as $a_i = \sigma(\sum_j w_{ij} o_j)$ where a_i is the activation of neuron *i*, *j* is the set of neurons in the preceding layer, w_{ij} is the weight of the connection between neuron *i* and neuron *j*, o_j is the output of neuron *j*, and $\sigma(x)$ is the logistic transfer function: $\sigma(x) = \frac{1}{1+e^{-x}}$.

Backpropagation Pass: For each record, information flows through the network to generate a prediction, as described above. The prediction is compared to the target value found in the training data for the current record, and that difference is propagated back through the network to update the weights. To be more precise, the change value Δw for updating the weights is calculated as:

$$\Delta w_{ij}(n+1) = \eta \delta_{pj} o_{pi} + \alpha \Delta w_{ij}(n)$$

where η is the learning rate parameter, δ_{pj} is the propagated error (described below), o_{pi} is the output of neuron *i* for record *p*, α is the momentum parameter, and $\Delta w_{ij}(n)$ is the change value for w_{ij} at the previous cycle.

The value of α is fixed during training, but the value of η varies across cycles of training. η starts at the user-specified initial eta, decreases logarithmically to the value of low eta, reverts to the high eta value, and then decreases again to low eta. The value of η is calculated as:

$$\eta(t) = \eta(t-1) \cdot \exp\left(\log\left(\frac{\eta_{low}}{\eta_{high}} \right) \Big/ d \right)$$

where *d* is the user-specified number of eta decay cycles. If $\eta(t-1) < \eta_{low}$, then $\eta(t)$ is set to $\eta_{high} \cdot \eta$ continues to cycle thusly until training is complete.

The backpropagated error value δ_{pj} is calculated based on where the connection lies in the network. For connections to output neurons it is calculated as $\delta_{pj} = (t_{pj} - o_{pj}) o_{pj} (1 - o_{pj})$ where t_{pj} is the target value of output *j* for record *p*. For the weights that do not connect to the output neurons, δ_{pj} calculations take account of upstream error propagation, as $\delta_{pj} = o_{pj}(1 - o_{pj}) \sum_k \delta_{pk} w_{kj}$ where *k* is the set of neurons to which neuron *j*'s output is connected, w_{kj} is the weight between the current neuron and neuron *k*, and δ_{pk} is the propagated error for that weight for the current input record.

3. Performance Evaluation

In this section, we evaluate the proposed techniques against a standard IDS test collection. The collection used is the KDD Cup'99 data set [4] which has been used in numerous intrusion detection papers. Three scenarios used in other research are employed this study: 3 attacks and normal; 10 attacks and normal; and 22 attacks and normal. In every experiment, *K=3* is used in the K-means clustering algorithm.

Backpropagation neural networks are trained using Clementine® [2]. The structure is determined by the Prune Method [2] which attempts first to create a large network; then pruning is performed to remove connections which do not contribute to the performance. Thirty percents of the training data is held out for evaluating the stopping criterion (i.e., no performance improvements for 200 iterations in a roll) to prevent overfitting.

Three Attack Types

In this scenario, 104,510 connections are randomly selected during training while 10,620 connections are randomly selected and used for testing. Connection groups in the attack-based reference vector approach are:

Group 1 → Neptune;
Group 2 → Normal;
Group 3 → Portsweep, Satan.

Detection (%)	NN	NN with Labels	NN with CRV	NN with ARV
Neptune	81.12	81.12	100	100
Portsweep	100	96.83	100	100
Satan	97.77	96.35	99.76	98.82
Normal	99.84	94.22	93.83	100

Rate (%)	NN	NN with Labels	NN with CRV	NN with ARV
Detection Rate	94.44	91.92	98.46	99.73
False Alarm Rate	2.27	11.63	6.42	1.12
False Negative Rate	2.11	5.85	0.25	1.12
False Positive Rate	0.16	5.78	6.17	0

Ten Attack Types

In this scenario, 160,000 connections are randomly selected and used during training while 7,412 connections are used in testing. Connection groups in the attack-based reference vector approach are:

Group 1 → Neptune;
Group 2 → Buffer_overflow, Warezclient, Warezmaster;
Group 3 → Ipsweep, Smurf;
Group 4 → Rootkit;
Group 5 → Satan;
Group 6 → Normal;
Group 7 → Guess_passwd, Portsweep.

Detection (%)	NN	NN with Labels	NN with CRV	NN with ARV
Buffer_ Overflow	69.23	0	76.92	84.62
Guess_ Passwd	92.00	100	96	100
Ipsweep	84.57	87.30	99.52	99.52
Neptune	80.65	80.65	80.74	80.74
Portsweep	95.02	100	99.81	97.70
Rootkit	0	25	50	100
Satan	87.67	89.03	88.41	99.88
Smurf	99.79	99.95	99.96	99.89
Warezclient	82.03	31.55	93.12	99.62
Warezmaster	90.91	0	100	90.91
Normal	90.55	19.30	96.80	100

Rate (%)	NN	NN with Labels	NN with CRV	NN with ARV
Detection Rate	88.84	78.82	94.88	96.88
Fals Alarm Rate	3.37	15.95	10.21	0
False Negative Rate	1.15	7.21	7.01	0
False Positive Rate	2.22	8.74	3.20	0

Twenty-Two Attack Types

In this scenario, 199,572 connections are randomly selected and used during training while 86,009 connections are used in testing. Connection groups in the attack-based reference vector approach consist of:

Group 1 → Neptune, Portsweep, Satan;
Group 2 → Back;
Group 3 → Guess_passwd, Ipsweep, Land, Nmap, Pod, Teardrop;
Group 4 → Smurf;
Group 5 → Buffer_overflow, Imap, Multihop, Warezmaster;
Group 6 → Ftp_write, Loadmodule, Perl, Rootkit, Spy;
Group 7 → Warezclient;
Group 8 → Phf;
Group 9 → Normal.

Detection (%)	NN	NN with Labels	NN with CRV	NN with ARV
Back	0	65.27	30.77	2.38
Buffer_ Overflow	0	100	100	40
Ftp_write	0	0	0	100
Guess_ Passwd	100	100	100	88.89
Imap	75	0	33.33	50
Ipsweep	33.65	1.59	81.73	89.42
Land	100	100	100	100
Loadmodule	50	0	0	33.24
Multihop	40	50	20	20
Neptune	80.75	99.98	99.98	99.64
Nmap	0	69.77	21.62	21.62
Perl	100	0	0	100
Phf	0	100	100	100
Pod	91.43	17.95	65.71	94.29
Portsweep	0	9.20	30.06	67.44
Rootkit	0	0	0	0
Satan	83.51	0	0	53.47
Smurf	99.94	99.86	99.94	99.93
Spy	50	50	50	50
Teardrop	100	100	90.69	100
Warezclient	0	0	0	62.43
Warezmaster	9.36	0	0	70.84
Normal	81.78	84.73	91.83	95.34

Rate (%)	NN	NN with Labels	NN with CRV	NN with ARV
Detection Rate	92.94	95.13	94.13	96.54
False Alarm Rate	7.71	10.06	10.42	5.37
False Negative Rate	2.39	0.94	1.60	0.48
False Positive Rate	5.32	9.12	8.82	4.89

4. Conclusion

This paper presents an augmentation to the original set of attributes by features derived from reference vectors to improve the performance of Backpropagation neural network models for network-based intrusion detection. Two approaches for creating reference vectors are studied. The cluster-based reference vector (CRV) approach regards the initial clusters as connection groups. The attack-based reference vector (ARV) approach considers the results of the initial clustering and separates connections of attacks whose majority do not fall into the same initial clusters with other attacks into new connection groups. Each connection group is then clustered by K-means clustering to produce a set of K reference vectors for each group. Distances between a connection and all reference vectors are added to the original set of attributes. The performance evaluation shows that models with ARV outperform models with only original attributes, models with the original set of attributes augmented with cluster labels, and models with CRV in all situations, each with different numbers of attack types.

References

[1] R. G. Bace. Intrusion Detection, ISBN 1-5780-185-6, 2001.

[2] Clementine® 8.0 Algorithms Guide, 2003.

[3] D. E. Denning. An Intrusion-Detection Model. IEEE Trans. On Software Engineering, Vol. SE-14, no.2, 222-232, 1987.

[4] KDD Cup 99 Data. http://kdd.ics.uci.edu/databases/kddcup99.html.

[5] Jirapummin, C., et al. 2002. Hybrid Neural Networks for Intrusion Detection System. The 2002 International Technical Conference on Circuits / Systems, Computers and Communications (ITC-CSCC 2002), Thailand, 2002.

[6] S. Kumar. Neural Networks: A Classroom Approach. McGraw-Hill, 2005.

[7] D.C. Lay, "Linear algebra and its applications," 2nd ed. Reading, MA: Addison-Wesley, 1996.

[8] S. C. Lee. Training a Neural-Network Based Intrusion Detector to Recognize Novel Attacks. John Hopkins, 2000.

[9] L. Me. GASSATA: A Genetic Algorithm As An Alternative Tool for Security Audit Trails Analysis. RAID'98, 1998.

[10] S. Mukkamala. Intrusion Detection Using Neural Network and Support Vector Machines. Dept. of Comp. Sci., New Mexico, 2001.

[11] P. Porras. STAT-A State Transition Analysis Tool for Intrusion Detection. TRCS93-25. Comp. Sci. Dept., UC Santa Barbara, 1993.

[12] J. Ryan. Intrusion Detection with Neural Networks. Comp. Sci., Taxas at Austin, 1998.

[13] T. Zhang, R. Ramakrishnon, and M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. Proceedings of the ACM SIGMOD Conference, pp.103-114, Montreal, Canada, 1996.

[14] Z. Zhang. HIDE: A Hierarchical Network Intrusion Detection System Using Statistical Preprocessing and Neural Network Classification. ECE, NJIT, 2001.

Using the Dynamic Proxy Approach to Introduce Role-Based Security to Java Data Objects

Matthias Merz
Department of Information Systems III
University of Mannheim
L 5,5, D-68131 Mannheim, Germany

E-mail: merz@uni-mannheim.de

Abstract

In this paper we present JDOSecure[1], a novel security approach for the Java Data Objects specification. The objective of JDOSecure is to prevent unauthorized access to the data store while using the JDO API. Based on the dynamic proxy approach, JDOSecure is able to collaborate with any JDO implementation without source code modification or recompilation. It introduces a fine grained access control mechanism to the JDO persistence layer and allows the definition of role-based permissions. Moreover, since JDOSecure uses the Java Authentication and Authorization Service, it allows the pluggable integration of further authentication services.

1. Introduction and Research Overview

This article presents the novel security architecture JDOSecure as add-on to the Java Data Objects (JDO) specification. JDOSecure introduces a role-based permission system to the JDO persistence layer. Based on a fine grained access control mechanism, JDOSecure prevents unauthorized access to the data store while using the JDO API.

The outline of the paper is organized as follows: Section one introduces the current JDO specification and emphasizes some shortcomings as well as the current development status of the JDO architecture. Section two outlines the security deficits of the JDO specification and recalls the design of the Java 2 Security Architecture. Section three presents the JDOSecure system architecture. The basic authentication and authorization concepts are introduced and the integration of JDOSecure with a JDO implementation is covered subsequently. The last section gives a critical review and addresses areas for future research.

[1]More information about JDOSecure can be found at `projekt-jdo.uni-mannheim.de/JDOSecure`

2. The Java Data Objects-Specification

The current JDO specification is an industry standard for object persistence developed by an initiative of Sun Microsystems under the auspices of the Java Community Process [5]. The latest 1.0.1 JDO version was introduced in May 2003 and is intended for use within the Java 2 Standard (J2SE) and Enterprise Edition (J2EE). It enables application developers to deal with persistent objects in a transparent fashion. Thus, JDO as a data store independent abstraction layer enables the mapping of domain object architectures to any type of data store.

The JDO specification defines two packages: The JDO Application Programming Interface (API) allows application developers to access and manage persistent objects. The classes and interfaces of the Service Providers Interface (SPI) are intended to be used exclusively by a JDO implementation and are located in the package `javax.jdo.spi`.

The interfaces and classes of the JDO API are located in the package `javax.jdo` [5]. Three important interfaces of the JDO API, namely `PersistenceManager`, `Transaction` and `Query` are summarized in the following: The `PersistenceManager` serves as primary application interface and provides methods to control the life cycle of persistent objects. The `Transaction` interface provides methods for initiation and management of transactions under user control. The `Query` interface allows to obtain persistent instances from the data store. Therefore, the JDO specification provides a Java oriented query language *JDO Query Language* (JDOQL).

Every instance that should be managed by a JDO implementation has to implement the `PersistenceCapable` interface. As part of the JDO SPI package, the `PersistenceCapable` interface has not to be implemented explicitly by an application developer. Instead, the JDO specification prefers a post-processor

tool (*JDO-Enhancer*) that automatically implements the `PersistenceCapable` interface. It transforms regular Java classes into persistent classes by adding the code to handle persistence. A XML-based *persistence descriptor* has to be configured previously. The JDO-Enhancer evaluates this information and modifies the Java bytecode of these classes adequately. The JDO specification assures the compatibility of the generated bytecode for the use within different JDO implementations. The `StateManager` interface as part of the JDO SPI provides the management of persistent fields and controls the object lifecycle of persistent instances.

Although JDO provides a standardized, transparent and data store independent persistence solution including tremendous benefits to Java application developers, the JDO specification has been discussed critically in the Java community. Beside technical details like the JDO enhancement process [10], the conceptual design as a lightweight persistence approach has been often criticized[2]. Some experts suggest, to shift JDO to a more comprehensive approach including distributed access functions to the persistent objects and multi-address-space communication [9]. As a result of it's lightweight nature, JDO does not provide a role-based security architecture, e.g. to restrict the access of individual users to the data store. Consequently, every user is able to query the entire data store as well as to delete any persistent object without further restriction.

In order to assist in the understanding of the JDOSecure security concept, the next section outlines the security deficits of the JDO specification and recalls the design of the Java 2 Security Architecture.

3. JDO Security Deficits

As already outlined in section 2, the JDO architecture is designed as a lightweight persistence approach without role-based security. Consequently, the JDO persistence layer does not provide any methods for user authentication or authorization. Every user has full access privileges to store, query, update and delete persistent objects without further restriction. For example using the `getObjectById()` method allows to receive any persistent object whereas the `deletePersistent()` method enables a user to delete objects from the data store.

A `PersistenceManagerFactory` instance will usually be constructed by calling the static `JDOHelper` method `getPersistenceManagerFactory(Properties props)`. Using this method enables an easy replacement of the currently preferred JDO implementation without source code modifications. Therefore information about the currently used JDO implementation and data store specific parameters has to be passed to this method by a `Properties` object. Even more, the user identification and password to access the underlying data store are also part of the `Properties` object. In order to guard against misunderstandings the JDO persistence approach does not distinguish between different user identifications or individual permissions. With the construction of a `PersistenceManager` instance the connection to the data store will be established and users are able to access the resource without further restriction.

At first glance, a slight improvement could be achieved by setting up individual user identifications at the level of the data store. This would allow the construction of different and user dependent `PersistenceManagerFactory` instances. If, however, all users should have access to a common database, individual user identifications and appropriate permissions have to be defined inside the data store. However, configuring user permissions to restrict the access to certain objects is quite complex. For example, when using a relational database management system, the permissions would have to be configured based on the object-relational mapping scheme and the structure of the database tables. Thus, it leads to the disadvantage of causing a strong dependency between the user application and the specific data store. In addition, a later replacement of the currently preferred data store leads to a time consuming and expensive migration. It is obvious that the strong binding of security permissions to a specific data store contradicts the intention of JDO, which is to provide application programmers a data store independent persistence abstraction layer.

JDOSecure considers an alternative approach and introduces data store independent and user specific permissions. Thus, it allows to restrict the access to store, query, update and delete persistent objects. Furthermore, it focuses on the compatibility to the JDO specification and allows the collaboration with any JDO implementation. As JDOSecure is based on the Java 2 Security Architecture the following section gives a brief overview to this architecture.

4. The System Architecture of JDOSecure

This section outlines the system architecture of JDOSecure. First, the design of the Java 2 Security Architecture as a basis for JDOSecure is recalled. The basic authentication and authorization concepts are introduced and subsequently, the integration of JDOSecure with a JDO implementation is covered.

[2] the substantial overlaps between the Enterprise JavaBeans specification [4] and JDO has been discussed in [6].

4.1. The Java 2 Security Architecture as a Basis for JDOSecure

The Java security architecture is based on three components: Bytecode verifier, class loader and security manager (cf. [8] and [3]). The bytecode verifier checks the correctness of the bytecode and prevents stack overflows. The class loader locates and loads classes into the Java Virtual Machine (JVM) and defines a unique namespace for each class. The security manager or, more accurately, the `AccessController` instance checks the invocation of security relevant operations e. g. local file system access, the setup of system properties or the use of network sockets [7].

With introduction of the Java Authentication and Authorization Service (JAAS) in Java 1.4 it gets possible to restrict the access to resources depending on the currently logged on user (*user-centric authorization*). If the user identity is authenticated successfully, e. g. with the help of user identification and password, the system is able to validate the user permissions before granting access to security relevant resources. In Java this mechanism is implemented by the `SecurityManager` which delegates access-requests to the `AccessController`. This instance validates the permissions and allows or disallows the access to the resources.

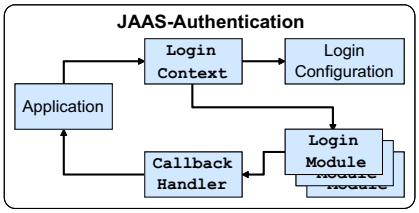


Figure 1. JAAS-Authentication

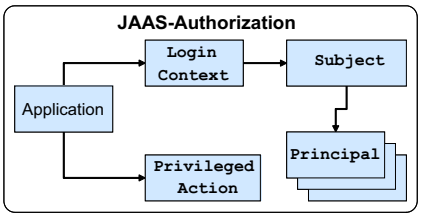


Figure 2. JAAS-Authorization

Figure 1 and 2 outlines the authentication and authorization process for JAAS. Starting with the authentication process, the application first creates a `LoginContext` and invokes the `login()` method of this instance. As defined in the login configuration file, the `LoginContext` delegates the authentication to one or several `LoginModules`. A `LoginModule` uses a `CallbackHandler` for communication with the application, the environment or the user e. g. to prompt for a password. JAAS provides the plug-in of PAM-authentication modules (Pluggable Authentication Modules) to integrate further authentication services. If an authentication attempt fails finally, a `SecurityException` will be thrown.

By calling the `getSubject()` method of the `LoginContext` it gets possible to refer to the `Subject` instance. This represents an authenticated user that is simultaneously associated with one or several `Principals` (a concrete user role). An application can perform a `PrivilegedAction` considering individual user permissions by invoking the static `Subject.doAs(subject, action)` method. Therefore the `AccessController` determines if a `Subject` is associated with at least one `Principal`, that provides the necessary permissions to perform this action. If a user or application has not the necessary permission to perform this `PrivilegedAction`, a SecurityException will be thrown. The relationship between permissions on the one hand and `Principals` on the other is defined in a separate *policy-file* (cf. section 4.4).

4.2. The Authentication Process

As described in section 3, a `PersistenceManagerFactory` instance can be invoked by calling the static `getPersistenceManagerFactory(Properties props)` method of the `JDOHelper` class. JDOSecure extends this concept in order to facilitate the collaboration between JDOSecure and any JDO implementation. Hence, JDOSecure provides a `JDOSecureHelper` class which is derived from `JDOHelper`. The `JDOSecureHelper` class overrides the `getPersistenceManagerFactory(Properties props)` method and serves as an entry-point for JDO applications.

The `Properties` object passed to the `JDOHelper` class contains amongst others user identification and password to access a JDO resource. As mentioned in section 3 the JDO architecture does not distinguish between different users. Therefore, the `JDOSecureHelper` analyzes the passed `Properties` object to authenticate a user at the level of the JDO persistence layer.

Once, a user has authenticated successfully, the `JDOSecureHelper` class constructs a new `PersistenceManagerFactory` instance. The basic idea in this context is to replace username and password in the `Properties` object, before the `JDOSecureHelper` class invokes the `getPersistenceManagerFactory(Properties props)` method of the original `JDOHelper` class.

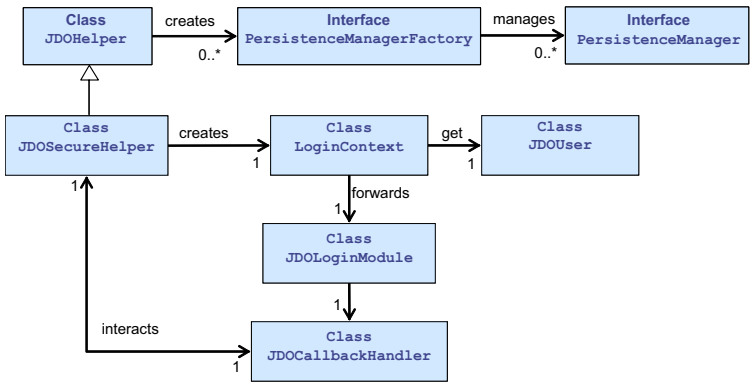


Figure 3. Context between JAAS-Authentication and JDOSecure

The intention of this replacement is to prevent a direct connection between user and JDO resource by using the `JDOHelper` class instead of the `JDOSecureHelper` class as a ”workaround”. The replaced password is unknown to the user and has to be configured by a security-administrator for the JDOSecure implementation and the JDO resource previously.

As illustrated in Figure 3, a `LoginContext` instance will be constructed by invoking the `getPersistenceManagerFactory()` method of the `JDOSecureHelper` class. The `LoginContext` instance forwards the authentication-request to the `JDOLoginModule` and `JDOCallbackHandler`. The `JDOCallbackHandler` instance validates the `ConnectionUserName` and the `ConnectionPassword` property to authenticate the user. If this process is completed without throwing a `SecurityException` the `LoginContext` instance is associated with a `JDOUser` instance and a `PersistenceManagerFactory` instance is constructed. As described in the next section, the `JDOSecureHelper` class returns a `PersistenceManagerFactory` instance (or more accurately a proxy object instead of the expected `PersistenceManagerFactory` instance, cf. section 4.3) to the user.

4.3. JDOSecure and the Dynamic Proxy Approach

There are two prerequisite conditions that could affect the acceptance of JDOSecure. First, JDOSecure should be independent from a concrete JDO implementation. And second, an overall approach should not contradict the JDO specification. In an attempt to meet these requirements, the presented security architecture implements the *dynamic proxy* pattern [1]. As it will be described, this concept enables the collaboration between JDOSecure and a standard JDO implementation, without an extensive adaptation.

A proxy instance implements the interfaces of a specific object and allows to control the access to it [2]. Generally, the creation of a proxy has to be done at compile time. Moreover, the dynamic proxy concept allows the construction of a proxy instance dynamically at runtime [1]. Dynamic proxy instances are created e.g. by using the static `newProxyInstance()` method of the `java.lang.reflect.Proxy` class and are always associated with an `InvocationHandler`. Any method invocation directed to proxy instance will be redirected to the `InvocationHandler.invoke()` method. The `invoke()` method allows to intercept method calls before they are forwarded to the original object.

The JDOSecure architecture implements the dynamic proxy concept as shown in Figure 4. The basic idea is to interpose a proxy between `PersistenceManager` and a JDO user or application. This would allow to validate specific user permissions at the `PMInvocationHandler` instance, before a method call is forwarded to the `PersistenceManager`. The following paragraph explains the architecture in more detail.

As mentioned above, the `JDOSecureHelper.getPersistenceManagerFactory()` method returns a dynamic proxy instance of the `PersistenceManagerFactory` class. Thus, the JDOSecure architecture avoids a direct interaction with the original `PersistenceManagerFactory`-instance and allows to manipulate method calls which are directed to the `PersistenceManagerFactory`. Invoking the `getPersistenceManager()` method, the `PMFInvocationHandler` returns a proxy of the `PersistenceManager` instance. JDOSecure uses the associated `InvocationHandler` (`PMInvocationHandler`) to manipulate method calls directed to the `PersistenceManager`. Thus, the

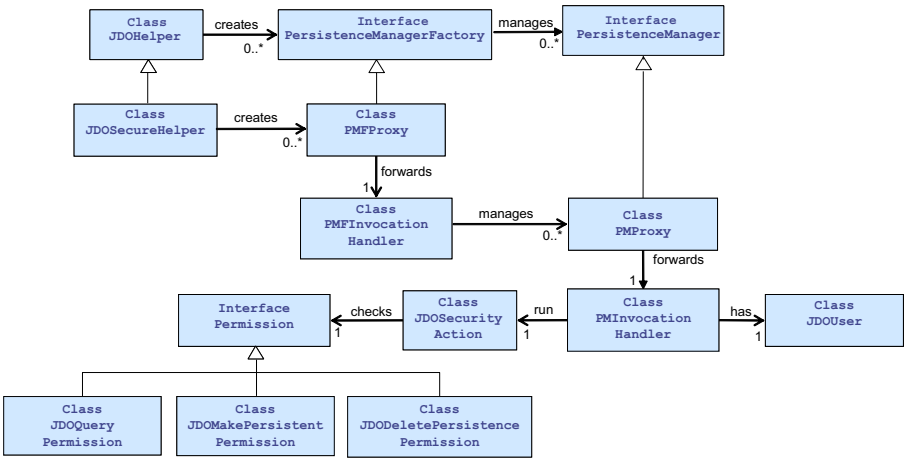


Figure 4. Context between JAAS-Authorization and JDOSecure

`PMInvocationHandler` represents the entry-point to implement the authorization function and allows to determine whether or not a user is allowed to invoke a `PersistenceManager` method.

4.4. The Authorization Process

JDOSecure enables the set-up of user specific permissions to allow or disallow the invocation of `PersistenceManager` methods. As mentioned above, a user receives a proxy of a `PersistenceManager` instance (`PMProxy`) by invoking the `getPersistenceManager()` method. Thus, JDOSecure is able to use the assigned `PMInvocationHandler` to validate, if an authenticated `JDOUser` has the permission to make a specific method invocation. The permissions are located in a separate policy-file and can be defined individually for any user. Currently, JDOSecure distinguishes between different permissions (Table 1) in order to restrict the access to the different `PersistenceManager` methods. JDOSecure enables also the limitation of user permissions to a certain package or a specific class. For example, the permission to invoke the `makePersistent()` method has to be defined for a package `org.test.sample` and a Principal "sampleuser" as follows:

```
grant Principal JDOUser "sampleuser"{
 permission JDOMakePersistentPermission
                "org.test.sample.*";
}
```

To validate if a user has the permission to invoke a specific `PersistenceManager` method, a `JDOSecurityAction` instance will be constructed and passed to the static `doAs(subject, action)` method of the `Subject` class. Consequently, the validation of a user permission is delegated to the `AccessController` as part of the Java 2 Security Architecture. If a user has the appropriate permission, the method call is forwarded to the original `PersistenceManager` instance. If not, a Java `SecurityException` is thrown.

Even this approach allows to restrict the creation, query and deletion of `PersistentCapable` instances, it is not suitable for the JDO update process. This problem is discussed in the following section.

4.5. JDOSecure and the Update of Object Attributes

JDO introduces the concept of transparent persistence and consequently JDO doesn't provide any additional methods to update object attributes or flushing instances to the data store. The security mechanism as described above, to verify user permissions when invoking methods of the JDO API, does not work in case of JDO updates.

As already mentioned, the JDO enhancer modifies regular Java classes to implement the `PersistentCapable` interface. Additionally, all setter methods are modified, that they do not change attributes directly. Instead, by invoking a setter method, an associated `StateManager` instance is notified. This `StateManager` is responsible to update the attributes in the corresponding `PersistentCapable` instance as well as to propagate these updates to the data base.

The idea in this context is to replace the `StateManager` by a proxy and to validate the user permissions in the corresponding `InvocationHandler` instance. As defined in the JDO specification, a `StateManager` instance

Methods of a `PersistenceManager`, that require specific permissions to be executed in the context of JDOSecure:	Necessary permission to invoke the according method for a specific class or package:
`makePersistent(..)` `makePersistentAll(..)`	`JDOMakePersistentPermission` $< Class >$
`deletePersistent(..)` `deletePersistentAll(..)`	`JDODeletePersistentPermission` $< Class >$
`getExtent(..)` `Query.execute(..)`	`JDOQueryPermission` $< Class >$

Table 1. JDOSecure Permissions

will be created by the JDO implementation with the invocation of the `PersistenceManager` methods `makePersistent()`, `makePersistentAll()`, `getExtent()`, `getObjectById()` as well as the `execute()` method of the `Query` instance. With the use of JDOSecure, the user does not interact with the `PersistenceManager` directly, but with the `PMInvocationHandler` instance. Before JDOSecure returns a `PersistentCapable` instance to the user, it gets possible to replace the corresponding `StateManager` by a proxy.

To implement this approach in JDOSecure, the `PMInvocationHandler` accesses the private `jdoStateManager` field by using the `java.lang.reflection` API to construct a dynamic proxy for the `StateManager`. In a second step, the `PMInvocationHandler` replaces the reference to the `StateManager` in the `PersistentCapable` instance with the proxy. The technical details like security issues when accessing private fields by using the `java.lang.reflection` API and other complications (e.g. the `jdoReplaceStateManager()` method of a `StateManager`) have been disregarded in this paper because of space limitations. However, JDOSecure enables the access control of the JDO update mechanism by introducing another proxy and a `JDOUpdatePermission`. As all other JDOSecure permissions, the `JDOUpdatePermission` could be specified individually for every user and a specific package or class.

5. Conclusion

In this article the novel security approach JDOSecure is introduced and the main advantages are highlighted. JDOSecure introduces a fine grained access control mechanism to the JDO persistence layer and allows the definition of role-based permissions. In the current version the permissions can be defined individually for every user/role with regards to certain operations (create, delete, update or query) and a specific class/package. Based on the dynamic proxy approach, JDOSecure is able to collaborate with any JDO implementation without source code modification or recompilation. Thus, JDOSecure will significantly contribute to the reduction of technical and conceptual JDO barriers and will consequently increase the security of JDO persistence architectures in the future. The JDOSecure software package and more information can be found at `projekt-jdo.uni-mannheim.de/JDOSecure`.

References

[1] J. Blosser. Explore the Dynamic Proxy API. http://java.sun.com/developer/technicalArticles/DataTypes/proxy/, 2000.

[2] 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, 1st edition, 1995.

[3] L. Gong. Java 2 Platform Security Architecture. http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc.html, 2002.

[4] Java Community Process. JSR-153: Enterprise JavaBeans 2.1, 2003.

[5] Java Community Process. JSR-012: Java Data Objects (JDO) Specification, Maintenance Draft Review, 2004.

[6] A. Korthaus and M. Merz. A Critical Analysis of JDO in the Context of J2EE. In A.-A. Ban, H. R. Arabnia, and M. Youngsong, editors, *Proceedings of the 2003 International Conference on Software Engineering Research and Practice (SERP '03)*, volume I, pages p. 34–40. CSREA Press, 2003.

[7] S. Oaks. *Java Security*. The Java Series. O'Reilly & Associates, Inc., Sebastopol, CA, USA, second edition, 2001.

[8] Sun Microsystems. *The Java Language Specification*. Addison-Wesley Professional, 3rd edition, 2005.

[9] TheServerSide.COM. Craig Russell Responds to Roger Sessions' Critique of JDO. http://www.theserverside.com/articles/article.tss?l=RusselvsSessions, 2001.

[10] TheServerSide.COM. A Criticism of Java Data Objects (JDO). http://www.theserverside.com/news/thread.tss?thread_id=8571, 2003.

A Novel Fairness Property of Electronic Commerce Protocols and Its Game-based Formalization

Ling Zhang Jianping Yin Mengjun Li Jieren Cheng
School of Computer, National University of Defense Technology
cs_zhangling@hotmail.com

Abstract

A game-based model of electronic commerce systems is built. It uses strategic game to model channels in various qualities and participants' dishonest behavior enables us to analyze cooperative and adversarial behaviors. In the game model a novel fairness of electronic commerce has been presented. Of the novel definition Balance Payoff and Abortive Fairness help us to avoid the confusion that a system satisfies fairness definition but is not fair to some participant factually; a 6-level hierarchy of fairness is proposed based on the conflict structure of the transitions, which is strong and refinement-robust.

1. Introduction

Electronic commerce prevails as a definitive alternative to traditional physical commerce, nothing should happen to undermine customers' confidence in the new technology. Thus, both electronic transactions and entities should satisfy a minimal set of properties, expected by users of electronic commerce systems. Fairness is the primary requirement of participants in commerce transactions. Unfortunately open networks and dishonest parties make non-face-to-face interactions hard to succeed. For this purpose a fair electronic commerce protocol is designed. If a party cannot get the expected item in the exchange phase, fairness must be re-established in one of the subsequent steps.

Modeling an electronic commerce system and specifying fairness in the model are crux to verify formally fairness property of a protocol. To our knowledge, Asokan[1] introduced strong and weak fairness in electronic commerce. And then there're many informal definitions of fairness following Asokan. They considered only participants' income but not expense, and in their definitions only an honest party may lose fairness if he quits voluntarily or terminates the session abnormally in response to system failures. Such resulted in that a system satisfies fairness definition but is not fair to some participant factually. In extension to definitions of Asokan, Pagina[2] proposed 6-level hierarchy of fairness, it described how to implement fairness but not specify what is fairness. Gärtner[3] et al. made the first attempt to approach a formal definition of fairness by means of a state transition model of a parallel distributed system. However, an electronic commerce transaction is not simply of a parallel distributed system. It is also regarded as a game.

In this paper, we propose a novel fairness in the game model. Of the novel fairness Balance Payoff considers both participants' income and expense, which is applicable to zero-sum transaction systems while the previous ones aren't; Abortive Fairness is introduced because honest participant may quit voluntarily or abort the protocol in response to system failures. These two help us to avoid the confusion that a system satisfies fairness definition but is not fair to some participants factually; a 6-level hierarchy of fairness is proposed based on the conflict structure of the transitions, which is strong and refinement-robust.

The remainder of this article is structured as follows. In Section 2 we propose a dynamic game-based model of an electronic commerce system. In Section 3, we introduce a novel fairness definition in the game model and its application. We conclude our paper in Section 4.

2. Modeling an Electronic Commerce System as a Dynamic Game

The object of study in game theory is the game, which is a formal model of an interactive situation. The formal definition lays out the players, their preferences, their information, strategic actions available to them, and how these influence the outcome. When players interact by playing a similar stage game numerous times, the game is called a dynamic game.

Unlike simultaneous games, players have at least some information about the strategies chosen on others and thus may contingent their play on past moves. In this section we wish to consider the representation of dynamic games and the predication of behavior in dynamic games. The extensive form is a complete description of how the game is played over time. In this subsection, we represent a dynamic in extensive form. A dynamic game is a tuple $<N,Q,p,\{ \mathcal{I}_i \}_{i\in N}, \succcurlyeq_i>$[5,6].

- N is a set of natural numbers, and represents a set of players.
- Q is a set of action sequences that satisfies the following properties:
  1) The empty sequence $\varnothing$ is a member of Q.
  2) If $(a_k)_{k=1}^{w} \in Q$ and $0<v<w$ (w is a natural number), then $(a_k)_{k=1}^{v} \in Q$.
  3) If an infinite action sequence $(a_k)_{k=1}^{\infty}$ satisfies $(a_k)_{k=1}^{v} \in Q$ for every positive integer v, then $(a_k)_{k=1}^{\infty} \in Q$.

  An action sequence $q\in Q$ is terminal if it is infinite or if there is not an available action a such that $q.a\in Q$. The set of terminal action sequences is denoted by Z.
- p is a function that assigns a player in N to every action sequence in Q\Z.
- $\mathcal{I}_i$ ($i\in N$) is an information partition of player $i\in N$.
- $\succcurlyeq_i$($i\in N$) is a preference relation of player $i\in N$ on Z.

$I_i\in \mathcal{I}_i$ is an information set obtained from an action sequence by i. $A(I_i)$ denotes the set of available actions of player i according to his current information set I_i. $A_i = \cup_{I_i\in\mathcal{I}_i} A(I_i)$ denotes all the available actions of i. A strategy of player i is a function s_i that assigns an action in A_i to every information set in $\mathcal{I}_i$. We denote the set of all strategies of i by S_i. A strategy profile is a vector $(s_i)_{i\in N}$ of strategies, where each s_i is a member of S_i.

2.1. Players

We model each protocol participant (i.e., the two main parties $Node_i$, $Node_j$ and the trusted third party TTP if there is any in Fig.2) as a player. In addition, we model the communication network named NET as a player too. Therefore, the player set N of the game is defined as N={1,2,3,4}, where 1 and 2 represents the two main parties of the protocol, 3 stands for the trusted third party, and 4 denotes the network. We denote the set N\{4} by N'.

TTP always behaves correctly, so we restrict the player that represents TTP to follow a particular strategy, whereas we allow the players that represent the potentially misbehaving main parties to choose among several strategies. We consider three kinds of channels. They are unreliable, resilient, and operational channels. So the player represents NET is also allowed to choose several strategies.

2.2. Information Set

Each player $i\in N$ has a local state $\sum_i(q)$ that represents all the information that i has obtained after the action sequence q. $\sum_i(q)$ is defined as a tuple $<Act_i(q), H_i(q), MS_i(q), R_i(q)>$, where

$Act_i(q)$: is 1 iff player i is still active(i.e., she has not quitted the protocol)after action sequence q;

$H_i(q)$: is player i's local history after action sequence q, which contains the events that are generated for i together with the round number of their generation;

$MS_i(q)$: is player i's message sets consisted of three parts: the initial knowledge, messages received in the protocol, and the new messages generated by i after action sequence q;

$R_i(q)$: is a positive integer that represents the round number for play i after action sequence q.

2.3. Available Actions

A good game can model all the possible ways in which the protocol participants can misbehave within the context of the protocol. Letting the protocol participants misbehave in any way they would lead to a game that would allow interactions that has nothing to do with the protocol being studied. Clearly, such a game would not be good because it would be far too rich, and we suspect that it would be difficult to analyze, and thus to draw interesting conclusions about the protocol. Therefore, we require the protocol participants to receive messages that are compatible with the protocol.

Communication between participants is generally achieved by sending and receiving messages. The action of sending messages is modeled by a trans event. The event trans.i.j.m means that participant i sends the message m to agent j via a CSP channel[7] named trans.i. Similarly, the action of receiving messages is modeled by the event rec.j.i.m. It means j receives the message m from via the channel named rec.j. Participants also have an evidence channel which they use to present evidence to an arbiter. This is illustrated in Fig.2. All participants send messages to NET and receive messages from NET. A trans event occurrence represents simultaneous occurrence of a pair of events: an active event and a respondent event. For example, the trans.i.j.m event occurs iff the trans.i.j.m event of i

and the trans.i.j.m event of NET occur simultaneously, where the former is an active event and the latter is a respondent event. All events except for trans and rec are active events.

Let us consider now how the main player $i \in N'$ to move after an action sequence. A special event named quit.i is always available for i at any time; a trans.i.j.m event means i can only send messages he can construct; a participant is always ready to receive messages as long as he is active, which is represented by a rec event; a message has been received does not mean it is able to be accepted, insteadly a received message is accepted only if it is verified correctly, which is represented by an acc event; a participant's process may terminate abnormally, i.e., in a way not prescribed by the protocol. Abnormal termination can be caused by local or remote factors, which are represented by faillocal and failremote events; a protocol consists of several subprotocols (e.g. a main and a recovery protocol), making branching possible, although they are intended to be executed in a given order by a honest participant, a misbehaved participant may change the order of execution.

The set $A_i(\Sigma_i(q))$ of available actions of player i(i=1,2) after an action sequence q is defined as Table 1. Every available action consists of two parts: guarded conditions and updated states. An event is available only if its guarded conditions are met. On the other hand the state is updated by the selected event.

Table 1 $A_i(\Sigma_i(q))$ (i=1,2,j∈N, i≠j)

$a \in A_i(\Sigma_i(q))$	Guarded Conditions		Updated States	
trans.i.j.m	$MS_i(q) \vdash m$; $Act_i(q)=1$	i send only messages he can construct	$H_i(q.a) := H_i(q).trans.i.j.m$	i records the trans event
evidence.i.m	$MS_i(q) \vdash m$; $Act_i(q)=1$	i can construct the evidence m	$H_i(q.a) := H_i(q).evidence.i.m$	i records the evidence event
rec.i.j.m	$Act_i(q)= 1$	i is active	$H_i(q.a) = H_i(q).rec.i.j.m$	i records the rec event
acc.i.j.m	$last(H_i(q)) = rec.i.j.m$ $Act_i(q)= 1$, $valid_i(m,q,cond_m)=1$	i has received m i has verified m	$H_i(q.a) = H_i(q).acc.i.j.m$ $MS_i(q.a) = MS_i(q) \cup \{m\}$	i records the acc event and adds m to its message set
quit.i	$Act_i(q)= 1$	i is active	$Act_i(q.a)= 0$	i is inactive
faillocal.i	$Act_i(q)= 1$	i is active	no	no
failremote.i	$Act_i(q)= 1$	i is active	no	no

For any action sequence q, if $MS_{NET}(q)$ is empty then NET can do nothing but accept messages, and it is always ready to accept any message; if it is not empty then it can accept and deliver messages; The delete event is used to model a unreliable channel where messages may be lost; NET can buffer messages and deliver them after a finite, but unknown amount of time, thus it models a resilient channel.

TTP must be impartial, and then it may not help one or the other player. To make sure that TTP does not have a strategy to help one of the executions of the protocol to cheat, we model TTP such that it is deterministic: at each stage of the execution of the protocol, TTP executes the action requested by the protocol.

2.4. Action Sequences and Player Function

The game is played in repeated rounds, where each round consists of the following two phases: (1) each active player in N' moves, one after the other, in order;

(2) the network moves. The game is not finished until every player in N' becomes inactive.

In order to make this precise, let us denote the set of players that are still active after the action sequence q and have an index larger than v by $P'(q,v)$. Formally, $P'(q,v)=\{ k \mid k \in N', Act_k=1, k>v\}$. Furthermore, let $k_{min}(q,v)$ denote the smallest index in $P'(q,v)$, which is defined as $k_{min}(q,v)=min_{k \in P'(q,v)}k$. Sending Preference of a message in $MS_{NET}(q)$ is a real vector (r,t) means the tth message NET received in the rth round, i.e., the tth trans event generated in the r run. We use $MQ'_{NET}(q)$ to specify a result message sequence of sorting messages from $MS_{NET}(q)$ in descending order by Sending Preference. In such a way for any $m_i \in MS_{NET}(q)$ and r_i , $t_i \in N(i=1,2)$,

$$'m_1 \succsim m_2' \text{ iff } 1) r_1=r_2 \text{ and } t_1<t_2 \text{ or } 2) r_1<r_2.$$

We define the set Q of action sequences and the player function p of the game together in an inductive manner:

a: the empty sequence $\varnothing \in Q$, $P(\varnothing)=1$.
b: let $q \in Q$, $p(q)=v(v \in N')$, then - $q.a \in Q$ for every $a \in A_v(\Sigma_v(q))$; - if $P'(q,v) \neq \varnothing$, then $p(q.a)=k_{min}(q.a,v)$; else $p(q.a) = NET$.

c: let q∈Q, p(q) = NET, the initial value of variable m is set to the first message of $MQ'_{NET}(q)$, then

c.1 : if m is empty, then $p(q) = k_{min}(q,0)$, $R_i(q) = R_i(q) + 1 (i \in N)$; else

c.2 : NET selects an active event a from $A_{NET}(\Sigma_{NET}(q))$ for the message m, then $q.a \in Q$, $q = q.a$, $p(q) = NET$, and the value of m is set to the next element of $MQ'_{NET}(q)$, and the head message of $MQ'_{NET}(q)$ is deleted; or NET does nothing for m, and the value of m is set to the next element of $MQ'_{NET}(q)$.

c.3: return c.1.

2.5. Payoffs

In this game, payoffs are numbers which represent the profits of players. Let us denote the items exchanged in the protocol by γ_i and γ_j. Furthermore, let us denote the value that γ_j is worth to i($i,j=1,2$; $i \neq j$) by $r_i(\gamma_j)$. We require only that $r_1(\gamma_2) > r_1(\gamma_1) > 0$ and $r_2(\gamma_1) > r_2(\gamma_2) > 0$ hold. The payoff $u_i(q)$ for i assigned to the terminal action sequence q is defined as Def.4. We call $u_i^+(q)$ the income and $u_i^-(q)$ the expense of i, and define them as follows:

Definition 4 Payoff

$$u_i(q) = u_i^+(q) - u_i^-(q)$$

$$u_i^+(q) = \begin{cases} r_i(\gamma_j); & \phi_i^+(q) = \text{true} \\ 0; & \phi_i^+(q) = \text{false} \end{cases}$$

$$u_i^-(q) = \begin{cases} r_i(\gamma_i); & \phi_i^-(q) = \text{true} \\ 0; & \phi_i^-(q) = \text{false} \end{cases}$$

$\phi_i^+(q)$ and $\phi_i^-(q)$ are logical formulae. The exact form of $\phi_i^+(q)$ and $\phi_i^-(q)$ depends on the particular exchange protocol being modeled, but the idea is that $\phi_i^+(q)$=true iff i gains access to γ_j, and $\phi_i^-(q)$=true iff i loses control over γ_i in q.

Note that according to the above definition, the payoff $u_i(q)$ of player i can take three kinds of possible values: =0, <0, and >0. $u_i(q)$ =0 means i' income is equal to his expense; $u_i(q)>0$ means i gains; $u_i(q)<0$ means i loses. As TTP and NET don't expect any profit, $u_{TTP}(q)=u_{NET}(q)=0$ holds for any terminal action sequence q.

3. A Novel Definition of Fairness and its Formalization

In this section we at first introduce generatability and revocability that can help the third party to resolve conflicts. Based on these two item properties we define a 6-level fairness. Then we define three kinds of fairness: Compliant Fairness, Abort Fairness and Deceptive Fairness. A protocol achieves fairness if and only if it achieves them respectively. Let us consider an electronic commerce protocol Π and its game model is G(Π). Furthermore, let s_i^* ($i \in N'$) denote honest strategy of the player i and s_4^* of the player NET represents that communication channels are reliable.

3.1. Generatability and Revocability

Certain item properties refer to the item descriptions which participating parties must give before engaging in electronic commerce protocol. In this section, we describe mainly generatability and revocability in detail that can help the third party to resolve conflicts.

A generatable item is an item which can be generated by TTP in case the receiving party can prove that if he has behaved correctly. Depending on the effectiveness of generatability we distinguish strong from weak generatability.

Definition 5 (Strong Generatability) It is guaranteed if TTP will be able to generate such an item.

Definition 6 (Weak Generatability) TTP can try to generate such an item, but he may fail in generating it, if a participating party has misbehaved. If this is the case, TTP will always detect such misbehavior and he will be able to provably determine the cheating party.

An item is called revocable if TTP can revoke it in case it has sufficient evidence to do so. Thus, revocability can be used to undo an exchange that cannot be completed. We distinguish strong and weak revocability.

Definition 7 (Strong Revocability) It is guaranteed if TTP will be able to revoke such an item.

Definition 8 (Weak Revocability) TTP can try to revoke such an item, but he may fail in revoking it. However, such a failure proves that the weakly revocable item really has been delivered to the party that desired this item.

If the desired item γ_j provides strong generatability, TTP will succeed in generating the item. Then the advantage of this is that conflicts are now automatically processed within the system, thus increasing the degree of the fairness. If γ_j is only weakly generable, TTP may either succeed or fail. In the first case the protocol terminates at this point, but in the second case additional steps are necessary: An external dispute can be started. If the item provides strong revocability, TTP will revoke the item and thus establish fairness. The same result is achieved if TTP succeeds in revoking a weakly revocable item. If the revocation of a weakly revocable item fails, then an external dispute can be started. In detail we set $r_i(\gamma_j)$ values according to property of item γ_j.

- $r_i(\gamma_j)$ =6: i can get γ_j in the system without further communication with the other party.

- $r_i(\gamma_j)$ =5: i can get γ_j in the system with eventual cooperation of the other party.
- $r_i(\gamma_j)$ =4: i can be provided a compensation for a suffered disadvantage instead of getting γ_j in the system.
- $r_i(\gamma_j)$ =3: i can only get γ_j outside of the system by an external dispute without further cooperation of the other party.
- $r_i(\gamma_j)$ =2: i can only get γ_j outside of the system by an external dispute with eventual cooperation of the other party.
- $r_i(\gamma_j)$ =1: i can be provided a compensation for a suffered disadvantage by an external dispute instead of getting γ_j in the system.
- $r_i(\gamma_j)$ =0: i gets nothing about γ_j.

3.2. A Novel Definition of Fairness and its Formalization

Asokan's fairness is only applicable to zero-sum transactions. Zero-sum transactions mean that the gain one party is the loss of the other. An electronic cash payment protocol in Example 1 of section 2.4 is a non-zero sum transaction for the loss of the customer is not the merchant gains. In this case the protocol satisfies Asokan's fairness definition but is actually unfair to a customer. The source of this confusion is that only income of a player is considered in the definition. Fortunately the new definition we propose considers both income and expense. Let q be a terminal action sequence, γ_i and γ_i,are exchanging items(i,j=1,2). As a preliminary Balance Payoff is defined as follows:

Definition 9 (Balance Payoff) i keeps Balance Payoff on q if and only if

(1) $u_j^+(q) = r_j(\gamma_i) \Rightarrow u_i^+(q) = r_i(\gamma_j)$ and

(2) $u_i^-(q) = r_i(\gamma_i) \Rightarrow u_i^+(q) = r_i(\gamma_j)$

An electronic commerce protocol first of all must be effective: if both main parties behave according to the protocol, and do not want to abandon the protocol, when the protocol has completed, they both reach a success termination state. This is called Compliant Fairness. Secondly, an electronic commerce system in essence is a parallel and distributed system. Any process and communication failures affect the whole system. For the purpose of this work, processes can fail in fail-stop. If a process failstops, it terminates abnormally. Communication failures contain message lost and message delay. On the other hand, a party which behaves according to the protocol can be sure that the protocol will be completed at a certain point in time. At completion, the state of the exchange as of that point is either final or any changes to the state will not degrade the level of fairness achieved by him so far. So we introduce Abortive Fairness. Thirdly, an electronic commerce system is not only a parallel and distributed system but also a game. The participants may misbehave, and may not follow the protocol faithfully. In such a situation, each participant has choices at various stages during the interaction with the others; the decisions that the participants make determine the outcome of their interaction; in order to achieve the most preferable outcome, a participant may follow a plan that does not coincide with the faithful execution of the protocol.

Definition 10 (Compliant Fairness) A protocol achieves Compliant Fairness if and only if i(i=1,2) can keep Balance Payoff on q where q= $o(s_1^*, s_2^*, s_3^*, s_4)$

Def.10 says either honest party will eventually get his expect item in spite of channel in quantities.

Definition 11 (Abortive Fairness) A protocol achieves Abortive Fairness to i if and only if i can keep Balance Payoff on q for any $q \in o(s_i^a, s_j, s_3^*, s_4)$ $_{i,j\in\{1,2\},i\neq j}$

In Def.11 s_i^a is an abortive strategy of s_i^* iff for any non-terminal action sequence q that i enforces when he follows s_i^*, if p(q)=i then $s_i^a(q) \in$ {quit.i, faillocal.i, failremote.i}. Def.11 says the honest party i can be sure that the protocol will be completed at a certain point in time. At completion, the state of the exchange as of that point is either final or any changes to the state will not degrade the level of fairness achieved by i so far.

Definition 12 (Deceptive Fairness)A protocol achieves Deceptive Fairness to i if and only if i can keep Balance Payoff on q for any $q \in o(s_i^*, s_j, s_3^*, s_4)$ $_{i,j\in\{1,2\},i\neq j}$

Def. 12 says that i can keep Balance Payoff if he behaves as prescribed by the protocol in spite of the other's misbehavior with the 'help' of a resilient channel.

Definition 13 (k-Fairness) A protocol achieves k-Fairness to i if and only if it achieves Compliant, Abortive, and Deceptive Fairness to i as well as k= $r_i(\gamma_j)$ $_{i,j\in\{1,2\},i\neq j}$.

The bigger k is, the stronger fairness is. Let k=0 if a protocol doesn't achieve any fairness of above three and we say it doesn't achieve fairness.

3.3. Application to fairness analysis

In this section we will give two examples of problems of historical definitions. The historical definition of fairness based on Asokan's, however, it is not subtle enough. In particular, one party's interest can be hurt even if no one else has gained any advantage. This situation can happen as Example 1.

Example 1- In sale transactions where customers pay by electronic cash if the payment sent by the customer gets lost while in transit, and never reaches the merchant, and the merchant does not send the goods, then we are in a scenario where neither the customer nor the merchant receives their shares in the exchange, even though the customer has effectively lost money.

Under this scenario, no one has gained any advantage, but the customer's interests have certainly been hurt: she did not receive anything in return for the money she has now lost. Users of electronic commerce protocols need to be protected against such losses.

The usual notion of fairness is also too constraint. In particular, one honest party may lose fairness in fact if he quits or terminates the session abnormally in response to system failures. But the protocol maintains fairness to this honest party following Asokan's definition. This situation can happen as Example 2.

Example 2- Let us take Zhou and Gollmann's fair non-repudiation protocol as an example [4]. Suppose step2 of the protocol is completed and party B is not offered a deciphering key before timeouts occurs. At the moment B may exit the protocol and consequently delete all messages in this session. Although B will get the key in the future, he cannot yet decipher the message with it for he has deleted the encrypted message. Then the result is that the protocol is not fair to B but achieves fairness as Asokan defined.

Our novel definition solves these problems. Of Example 1 the customer's Balance Payoff is broken and then Compliant Fairness is not achieved because it doesn't satisfy condition 2 of Def.9. We conclude the protocol achieves 0-Fairness for the customer. In Example 2 Abortive Fairness is not achieved, so we conclude the protocol achieves 0-Fairness for party B.

4. Conclusions

In this paper we propose a novel definition of fairness of electronic commerce protocols and suggest its formalization in a game model. The improvements of the new fairness are (1) Balance Payoff considers both participants' income and expense, which is applicable to zero-sum transaction systems while the previous ones aren't; (2) Abortive Fairness is introduced because honest participant may quit voluntarily or abort the protocol in response to system failures. These two help us to avoid the confusion that a system satisfies fairness definition but is not fair to some participants factually; (3) a 6-level hierarchy of fairness is proposed based on the conflict structure of the transitions, which is strong and refinement-robust.

5. Acknowledge

This research is supported by the National Science Foundation of China under Grant No.90104026, No. 60073001, NO.60473057, NO.90604007, NO. 60373023, and the National High Technology 863 Program of China under Grant No.2002AA144040.

6. References

[1] Asokan N. Fairness in electronic commerce[D]. University of Waterloo. 1998.

[2] H. Pagnia, H. Vogt, and F.C. Gärtner. Fair Exchange[J]. The Computer Journal, 2003,46(1):55-75.

[3] F.C. Gärtner, H. Pagnia,and H. Vogt. Approaching a formal definition of fairness in electronic commerce[C]. Proceedings of the International SRDS Workshop on Electronic Commerce (WELCOM'99). IEEE Computer Society Press. 1999: 354-359.

[4] K. Kim, S. Park, and J. Baek. Improving Fairness and Privacy of Zhou-Gollmann's Fair Non-Repudiation Protocol[C]. ICPP '99: Proceedings of the 1999 International Workshops on Parallel Processing. IEEE Computer Society Press. 1999:140-145.

[5] L. Buttyan and J.-P. Hubaux. Rational exchange -A Formal Model Based on GameTheory[C]. In Proceedings of the 2nd International Workshop on Electronic Commerce (WELCOM). LNCS,2001:114-126.

[6] M.Osborne and A.Rubinstein. A course in Game Theory[M]. MIT Press, 1994.

[7] C.A.R.Hoare. Communicating Sequential Processes[M]. Prentice Hall, 2004.

Towards a contextualized access to the cultural heritage world using 360 Panoramic Images

P. Mazzoleni[1], S. Valtolina[1], S. Franzoni[1], P. Mussio[1], E. Bertino[2]
[1]CS Department - University of Milan, Italy
[2]CERIAS and CS Department - Purdue University, USA

Abstract

In this paper we present a system to include knowledge in the process of organizing and disseminating heritage using 360 panoramic images. With our solution, data in heterogeneous digital archives are integrated according to a semantic model and are enriched by the contribution of domain experts which specify stories, called narrations, reflecting their knowledge of the heritage. Data in digital archives and narrations are then used in combination to automatically propose guided tours of the cultural content in the panoramic images.

1 Introduction

The spread of high bandwidth network, the possibility to buy high quality digital cameras with few hundreds dollars, and the availability of open-source programs to manipulate and compose digital pictures, are making 360 panoramic imaging a technology used in an increasing number of web sites. In general, we can say that a panoramic image (also called panorama in the following) represents a high aspect ratio or wide screen image, especially suitable for landscapes, where a lot of sceneries can be taken in a glance.

Panoramas are used in many domains such as cultural heritage, entertainment, e-commerce and e-science, to show events, estates, furniture, cars, jewellery, astronomic images, and so forth. To give you an idea of the phenomenon, consider that there are portals such as http://www.fullscreenqtvr.com/, http://www.vrway.com/, and http://www.arounder.com/ dedicated to such technology from which users can access thousands of panoramic images all around the world.

Although such technology is very appealing, panoramic imaging is still not broadly adopted as a technology for information dissemination. We believe such limitation can be attributed to two main aspects. The first aspect is that panoramas are not integrated with the rest of the information available at a web site. Panoramas are often only an eye-caching web component to represent a space (a monument, a landscape) usually described by short text. Where interaction is supported, it is limited to some active regions, called hotspots which connect another panorama or link another page with some additional information of the region. This is not enough. In our vision, panoramic imaging technology should be used to create environments that, by hiding the complexity of the underlying DBs and archives, are easy to create and more natural for users to interact with the all the information of the heritage represented in the panoramas. To address such integration, in this paper we introduce the concept of *semantic hotspot* through which regions of the panorama are semantically linked to information stored in one (or several) data sources. The second aspect that in our opinion limits the use of panoramas is the fact that it is not possible to promote a contextualize access to elements in the panoramas according to their historical, artistic, or anthropological meanings. In particular, domain experts should be able to enhance the information presented in a panorama by guiding the attention of the end-users to the elements which are more relevant to a specific theme. This is what happens in a real visit: the expert guide does not wait for the visitor to ask additional information about an area (e.g., a fountain or a church), but points out to the visitor the elements which she/he things are more relevant to a specific context or better suited to a specific target of visitors. To address this issue, we provide a methodology, based on the concept of *narration* (which is conceived as description of a story, an interpretation of the information presented) through which domain experts can contextualize and connect different semantic hotspots specified in a panoramic image (or in different panoramas) in order to generate customized presentations of the contents. The process followed to build the semantic model, as well as the process used by domain experts to specify a narration, is out of the scope of this paper. However, for sake of completeness we will briefly overview such task, developed in the framework of the EU project called T.Arc.H.N.A. (Towards Archaeological Heritage New Accessibility).

2 A Brief History of Panoramic Images and Related Work

The use of panoramic images for visualizing landscapes and cities goes back to many years. The term "panorama" was coined in the 18th century by Robert Barker to describe the panoramic painting of Edinburgh that he depicted and arranged on a cylindrical surface in order to offer an immersive visualization of the city. Image-based virtual tour systems, like those using panoramic photos, have widely been investigated, and several tools have been developed [1]. EasyTour [2], a virtual tour system built on panorama, and TIP (Tour Into the Picture) [3] techniques are examples of such tools. With EasyTour, the user can create panoramas and edit hot areas. The hot area in the scene map corresponds to a panorama sight and the hot areas in each panorama correspond to TIP sub-sights. Through modeling the sight navigation into global model and local model, the user can virtually tour the scenes. The use of panorama as virtual set for immersive story is also discussed in [4]. Virtual characters can walk and talk in the environment by using a locomotion network, which establishes a correspondence between 2D panorama image and 3D environment. Unfortunately, as we said in the introduction, the way panoramas are supported suffers of the limitation of being simple views of a landscape. In our opinion, panoramas could become a new interaction medium. The idea is to follow the approach of those web sites (Flickr [5], Bubbleshare [6]) offering to the user the possibility to store and share their own photos, eventually enriched with other multimedia content. For example Bubbleshare allows its users to publish photographic albums integrated with textual captions, audio comments, and notes of other visitors. By doing this, the user is able to narrate the story behind each image, sharing not only a simple piece of data, but also his experiences and ideas.

3 System's Architecture

The architecture of the system we built is presented in Figure 1. As it can be seen from the figure, the system is composed by three main components: the Tarchna Engine, the Semantic Hotspot Mapping, and the Panorama Contextualized Access.

3.1 The Tarchna Engine

In brief, the Tarchna Engine is the component in charge of maintaining the link between the data in the databases and the narrations domain experts specify on such data. A narration is a description of a story, a tale, an interpretation of one (or multiple) artefact(s) that contextualizes it (them) within the society it (they) belong(s). The Tarchna system

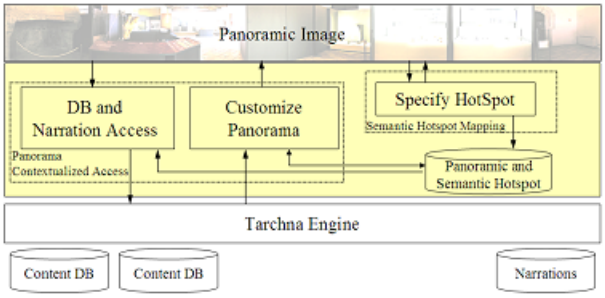


Figure 1. System's Architecture

is again organized in three levels: Source, Knowledge, and Appearance. *Source:* The system's infrastructure supports the integration of digital archives owned by the cultural institutions without having to replicate their data in a centralized repository. The integration is carried out by means of a semantic layer wrapping the different data sources (typically DB schemas) and offering a uniform interface to the stored information. *Knowledge:* The domain experts are provided with tools to interact with the data sources, hidden by the semantic layer, and edit narrations. A narration can be related to an artefact or a set of artefacts stored in the archives (e.g. an artefact exhibited in a museum, a tomb, a historical personage) or it can be related to a more generic topic (e.g. agriculture or contact with the Greeks civilization). Narrations are stored in the ontology, with all the references to the related objects. *Appearance:* Narrations can be disseminated to the public through multimedia applications called Virtual Wings (VWs). VWs are conceived as visual interactive interfaces through which users can specify their preferences and build their own "virtual" tours. In this context, the contextualized panoramic image presented in this paper can be viewed as a wing the technology of which can be used independently from the cultural heritage to be disseminated.

3.2 The Semantic Hotspot Mapping

Semantic Hotspots are a new generation of hotspots for panoramic images. Like traditional hotspots they can be connected to one or multiple regions of a panorama. However, while traditional hotspots are associated with actions (to be executed when the user clicks on the region), semantic hotspots are associated to concepts within the cultural model describing the information in the panorama. In our approach, we combine the process of specifying hotspots with the semantic domain model of the heritage in the panorama. In this way, domain experts do not have to know in advance which is the additional information available in the database (and how to retrieve it) but they only need to know which elements are important in a panoramic image and which is the semantic information describing such el-

ements. The Tarchna Engine will take care of the rest. To better understand the system, consider as an example the process of defining the semantic hotspots for a panorama of an Etruscan Tomb and moreover assume Figure 2 to be a part of the ontology created for the archaeological domain of Tarchna

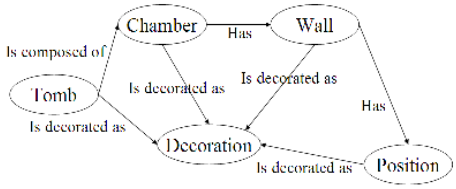
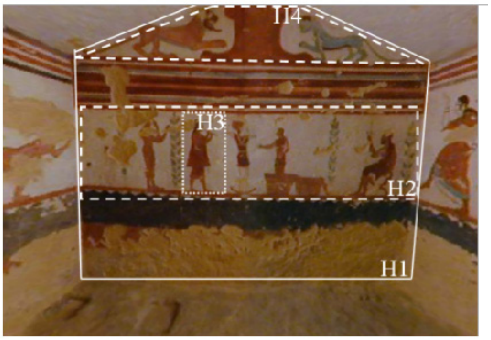


Figure 2. Ontology Sketch

As a first task, the domain expert specifies the panorama representing a tomb. By selecting the corresponding concept on the semantic model, the Tarchna Engine automatically presents a preview of the additional information of all the tombs available in the data sources. Among those data, the expert can select the name of the tomb (e.g. "Tomba delle Bighe") to complete the specification of the hotspot. At this point, if the domain expert specifies a region within the panorama, the system will suggest the semantic concepts directly related to the concept Tomb in the ontology. According to the ontology in Figure 3, the hotspot will be either associated with a Chamber or with a decoration of the "Tomba delle Bighe". According to the system architecture in Figure 3, the information linking hotspots and semantic concepts is stored in a database along with the panoramas. The database reflects the domain model of the heritage as in the Tarchna Engine with the addition of two classes, called "panorama" and "hotspot", to respectively store the Image_Id of the panoramas (unique for all images) and the Hotspot_Id of the hotspot (unique for each single panorama). The relation between a panoramic and its hotspots is maintained by the relation "is_mapped_to" through which both image_id and hotspot_id can be linked to any concepts in the domain model. Figure 3 shows an example of semantic hotspots created for the Tomb. In the figure, ABC12 is the Id of the decoration as extracted from the DB during the process of specifying semantic hotspot for a panorama

3.3 Panorama Contextualized Access

In this section, we describe the details of the process to promote a contextualized access to panoramic images. In the following we assume that panoramas include semantic hotspots and the Tarchna Engine collecting both link to database information as well as to narrations. When the visitor clicks on a region associated with a semantic hotspot, she/he can access additional information about the area. For instance in the Figure 4 the final panorama application will



Image_id	Hotstpot_id
IMG1	H1…H4

Relation between panoramas and Hotspots

Hotstpot_id	Semantic Instance	Semantic Concept
IMG1	Tomba delle bighe	Tomb
H1	Back_Wall	Wall
H2	Main_Decoration	Position
H3	ABC12	Decoration
H4	Slopes	Position

Relation between images and hotspots to the semantic concepts

Figure 3. Semantic Hotspot of a panoramic image of the Tomb

present to the user the possibility, by selecting the hotspot H3, to retrieve both all narrations directly connected to the decoration and those regarding the chamber (H1) or the tomb (IMG1). The text of the relevant narrations as well as the results of the query are then returned to the visitor. In addition to dynamically extract information associated with a semantic hotspot, the system can contextualize the information in the panorama according to a theme selected by the visitor. When the user selects a narration among the ones prepared by domain expert, the system extracts and highlights the semantic hotspots in the panorama which are related to selected theme. If a narration is associated with hotspots belonging to multiple panoramas, the user has the opportunity to preview them and find out which are the artifacts related to the narration she/he selected.

4 Implementation and Evaluation

Figure 4 shows an example of our final application. In this screenshot it is possible to view a panorama regarding an area of the National Museum of Tarquinia. The scenario is the following: a user browsing the panorama has clicked on the finding representing the winged horses. As a consequence, below the panorama, the user can view as much tabs as the narrations connected to the selected finding. Selecting the narration titled "L'Ara della Regina", on the right,

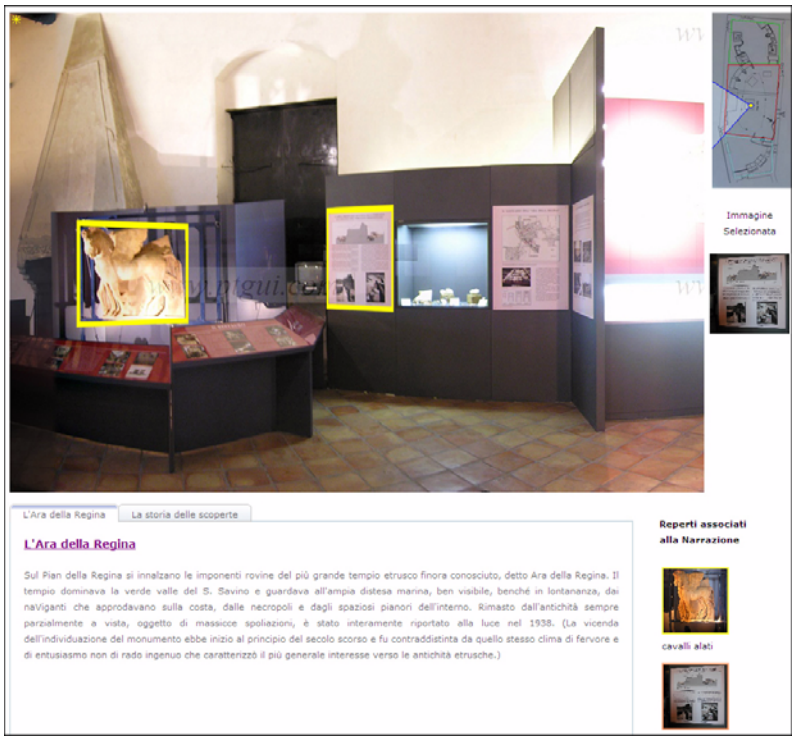


Figure 4. Screenshot of web application of the National Museum of Tarquinia

it is possible to view the two findings connected to the narration "L'Ara della Regina". Note that the two findings are also automatically highlighted in the panorama to catch the attention of the final user.

5 Conclusion and Future Work

In this paper we presented a system to include knowledge in the process of organizing and disseminating heritage using 360 panoramic images. With our solution, digital archives are integrated with the panoramas and domain experts can actively participate, specifying narrations, in the process of dissemination to the public. Future works are oriented in two main directions. The first one aims at giving the final user the possibility to annotate the panoramas in order to share comments with others. The second one aims at applying image extraction algorithms to semi-automatically discover the regions to which associate the hotspots.

6 ACKNOWLEDGMENTS

This work has been partially founded by the European Community under the T.Arc.H.N.A. project.

References

[1] J. Y. Zheng, M. Shi, and M. Kato, "Route panoramas for city navigation," *Proceedings of the eleventh ACM international conference on Multimedia. Berkelet, CA, USA*, pp. 108–109, November 2003.

[2] Z. Pan, X. Fang, J. Shi, and D. Xu, "Easy tour: a new image-based virtual tour system," *Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry. Singapore*, pp. 467–471, June 2004.

[3] Y. Horry, K. Anjyo, and K. Arai, "Tour into the picture: Using a spidery mesh interface to make animation from a single image," *In Proceedings of SIGGRAPH'97. Los Angeles*, pp. 225–232, August 1997.

[4] A. Nakano, C. Rai, and J. Hoshino, "Panorama-based immersive story environment," *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology ACE '04. Singapore*, pp. 354–354, September 2004.

[5] [Online]. Available: http://www.flickr.com

[6] [Online]. Available: http://www.bubbleshare.com/upload

Object and Knowledge Modelling for Impact Fusion

Catherine Howard
Electronic Warfare and Radar Division
Defence Science and Technology Organisation
PO Box 1500, Edinburgh, South Australia, 5111
catherine.howard@dsto.defence.gov.au

Markus Stumptner
Advanced Computing Research Centre
University of South Australia
Adelaide, South Australia, 5095
mst@cs.unisa.edu.au

Abstract. This paper discusses the use of Object Oriented Probabilistic Relational Models to model the tactical military domain for intent recognition, combining classic OO modelling instruments with a probabilistic knowledge formalism. Using OPRMs we model both observations of adversary behaviour and prior knowledge such as doctrine and terrain in order to infer adversary intent. We present a generic ontology which we developed for modelling the domain. Using this ontology and OPRMs, we construct a tactical level model of a hostile armored vehicle company, compare it to prior approaches, and present an algorithm for the automatic construction of situation specific models.

Keywords: Probabilistic Relational Models, Information and Impact Fusion, OO Knowledge representation

1. Introduction

This paper discusses knowledge representation and reasoning techniques for intent (or plan) recognition of tactical military commanders, via a combination of classical modular software engineering and probabilistic knowledge engineering approaches. Intent recognition involves fusing observations of an adversary's actions with prior knowledge such as adversary organizational structure, doctrinal and environmental information in order to determine an adversary's intent. In this paper, we first outline OO Probabilistic Relational Models (OPRMs) [1,2], the language which we use to model the battlespace. We then develop an OPRM model to predict the intent of a hostile armored vehicle company. We then compare our approach to a previously publish approach [3-6] which combines UML with Bayesian Networks (hereafter referred to as UML+BN). Finally, we discuss algorithms for automatic construction of situation specific models of the battlespace from OPRM classes.

The first step toward determining an adversary's intent from observations of their actions and capabilities is to create an ontology to model the battle space. The ontology shown in Fig. 1 was developed by extensively extending the Situational Awareness (SAW) ontology developed by [7] as part of a formal approach to design and development of information fusion systems. It provides a conceptual framework for the concepts modeled during intent recognition and provides a basis for defining classes in the OPRM language. This core ontology enables the development of generic representation and reasoning techniques that can be applied to a variety of new situations through using or extending this ontology.

2. OO Probabilistic Relational Models

Reasoning about intent requires reasoning about objects and the relationships between them, which requires at least some of the expressive power of a first order logical language. Our approach to intent recognition uses Object Oriented Probabilistic Relational Models (OPRMs) [1,2] a language which addresses some of the above listed requirements. OPRMs were developed to overcome some of the limitations of traditional techniques, such as Bayesian Networks, for modelling complex domains. OPRMs have a relational component, describing how the classes in the domain are related, and a probabilistic component, detailing the probabilistic dependencies between attributes in the domain.

Definition 1: An *Object-Oriented Probabilistic Relational Model (OPRM)* is a pair (*RC,PC*) of a *relational component RC* and a *probabilistic component PC*.

An OPRM's relational component *RC* comprises:

- A set of classes, $\mathbf{C} = \{C_1, C_2, \ldots C_n\}$, and a partial ordering over **C** which defines the class hierarchy.
- A set of named instances, **I**. Each instance is an instance of only one class.
- A set of descriptive attribute names $\mathbf{A_C}$ for each class C in **C**. Attribute A of class C_1 is denoted by $C_1.A$. For an instance $i \in \mathbf{I}$ s.t. A is defined for the class of i, i.A denotes the value of attribute A for i.
- A set of reference attribute names $\mathbf{R_C}$ for each class C in **C.** A reference attribute ρ of class C_1 (i.e., $\rho \in \mathbf{R}_{C1}$) is denoted by $C_1.\rho$. Reference attributes express functional relationships between instances

An OPRM's probabilistic component *PC* consists of a set of conditional probability models $P(A|Pa[A])$ for the descriptive attributes, where $Pa[A]$ is the set of parents of *A*. These models may be attached to particular instances or inherited from classes.

3. Modelling a Hostile Tank Company

Recognizing adversary intent from observed behaviours can be difficult in that the observed behaviors will be strongly affected by doctrine, terrain and environment. For example: it is difficult for tanks to traverse slopes steeper than 30° [3]. When confronted with terrain of this type, a tank platoon will need to employ different doctrines and patterns of movement to achieve their goals. Without knowledge of the terrain, this change in observed behaviour could result in incorrect hypotheses about the adversary's intent. However, combining information about the terrain with hypotheses of adversary's capabilities can lead to inferences about possible avenues

of approach and areas which provide the adversary with cover and concealment. Thus prior knowledge can play an important role in the intent recognition process by:

- Pruning the hypothesis space generated by the observations
- Recognizing plausible types of behaviours and connect them to prediction of adversary state

In order to use (often complex) prior knowledge in the intent recognition process, it is necessary to represent it in a generic, structured, flexible and compact fashion. A generic, structured representation will take advantage of the inherent structure in the domain (e.g. military organizational structure) and maximize the potential for reuse. The representation needs to be flexible in order to cope with the uncertainty in the domain. The military domain is uncertain for a number of reasons. The data from sensing systems, and indeed prior knowledge, may be incomplete, incorrect, contradictory or uncertain. Sensing systems data provide only a partial picture of the battle space. It may have various degrees of latency and may be affected by the environment or by adversary deception or confusion, which creates false or misleading data.

In order to hypothesize about the intent of the hostile tank company, we need to represent and reason about the location, deployment, movements and status of the tanks in the company as well as the relationships that exist between them, their environment and their presumptive targets. Table 1 provides details, in terms of our ontology, of the factors we have considered important in producing a static model for the hostile company. Because Procedural Doctrine is highly prescriptive and usually not open to interpretation, we will focus on modelling the tactical level procedural doctrine for our armored vehicle domain. We will assume doctrine to be static, as changing doctrine requires time. We will now discuss how these important factors are modeled using OPRM classes.

Modelling of the battlespace begins with a set of classes (a subset of the classes specified in our ontology) which represents the potential objects and relationships in the battlespace as a set of *class frames*. These class frames form a probabilistic ontology from which situation specific models can be constructed. A class frame inherits all slots and facets from its superclass, including the probability models. The ontology presented in Fig. 1 defines our class inheritance hierarchy. The classes and their reference and descriptive attributes used in our model of the tank company are shown in Table 2.

When a particular entity is being modelled, an instance of the class frame is created. This instance inherits all the slots and facets of the class frame, including the probability distributions, but the contents of the slots may be more fully specified by including specific knowledge about the instance. If the exact value of the attribute is not known, the slot takes on the possible values specified in the class frame. If an observation is made, a value is assigned to the slot, conditioning the probability distribution of the instance.

Key Factors	**Comments**
Organizational structure	Physical structure of the company: consists of 3 platoons and each platoon has 3 tanks.
Terrain	The maneuverability of armored vehicles is affected by - Vegetation (modelled as cover) - Soil types (modelled as carrying) - Gradients - Physical barriers (e.g. anti tank ditches, minefields, trenches, impassable water bodies)
Weather	Visibility (i.e. a hostile company is more likely to discover own forces if the visibility is good)
Objects	- Movement capabilities •Direction of movement •Maneuverability in certain types of terrain
Relationships	- Physical - Spatial: Distance to presumptive target - Temporal: - Perceptual: Has discovered own forces - Functional
Platoon Doctrine	- Formation - Direction of guns

Table 1. Key factors in our model.

Frame	**Slot**
Time Slice	Company, Terrain, Weather
Company	Platoon, Weather, Has Discovered Own Forces, Distance to Target, Intent
Platoon	Company, Tank, Terrain, Maneuverability, Distance to Target, Direction of Movement, Intent, Doctrine
Doctrine	Platoon, Terrain, Formation, Direction of Guns
Tank	Platoon, Terrain, Maneuverability, Distance to Target, Direction of Movement
Is Discovered	Company, Weather, Exists
Terrain	Gradient, On Road, Physical Barrier, Carrying, Cover
Weather	Visibility

Table 2. The classes, reference and descriptive attributes.

Fig. 2 shows the OPRM class model implemented for the hostile tank company. The model is the simplest form of OPRM, where the complete relational structure is known. This means that we are assuming the set of objects and the set of potential relationships between them are known and there is no uncertainty about the structure of the model. We chose to model the domain in this fashion, even though OPRMs are capable of handling structural uncertainty because we wanted to be able to directly compare our model with the model produced using the UML+BN approach in [3, 4, 6]. Given the relational structure, the OPRM specifies a probability

distribution over the attributes of the instances of the model. The unique names assumption is employed which means that each object in the knowledge base is assumed to have a unique identifier (i.e. there is no uncertainty about the identity of the objects).

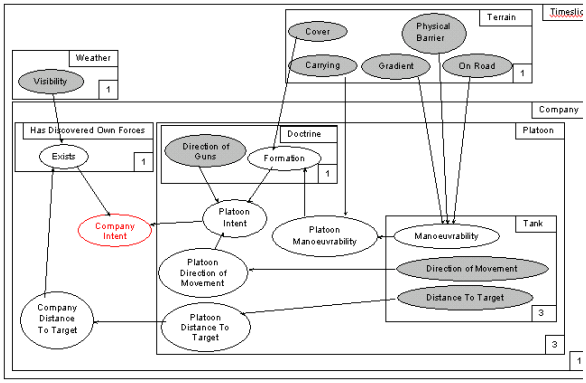


Fig. 2. The OPRM class diagram for the tank company. The query (double outline) variable is Company Intent. The observed variables (shaded) are: the Tank's *Direction of Movement* and *Distance to Target*, the Terrain's *Cover, Carrying, Gradient, On Road, Physical Barrier* and the Weather's *Visibility.*

4. Comparison of the OPRM and UML+BN Approaches to Intent Recognition

UML is a notation for modelling systems using object oriented concepts. In the UML+BN approach, UML is used to represent the class structure and the relationships between the classes in the domain. These UML classes then provide a visual structural context for the *manual* construction of the BN model to reason about the domain. For example, the Platoon and Company UML diagrams provide clues about how the Platoon and Company BNs models interact once created. In [3-6], it is argued that having the UML structure in the 'background' makes it easier to implement the large and complex BNs required for intent analysis, resulting in more reliable BNs and decreased modelling time. Suzic's hostile tank company's intent model, which was produced using UML+BN, is reproduced in Fig. 3 to enable easy comparison with the ORPM presented in Fig 2.

The UML+BN approach to intent recognition has several limitations. Firstly, BNs have been shown to have some limitations when modelling large, complex environments such as the military domain. The military domain is highly dynamic and involves an unknown number of objects interacting with each other in ways that evolve over time. When a model of the battlespace is designed, we do not know apriori which objects will be present, their attributes or the relationships in which the objects will participate. BNs are rigid: they model the domain with a predefined set of random variables and a fixed topology which applies to all problem instances. BNs cannot represent uncertainty about the existence, number or configuration of objects in the battlespace (i.e. BNs can not accommodate uncertainty about the structure of the model). This limitation is a direct consequence of BNs lacking the concept of first-class objects. OPRMs are flexible, allowing uncertainty about the existence, number and configuration of objects to be represented.

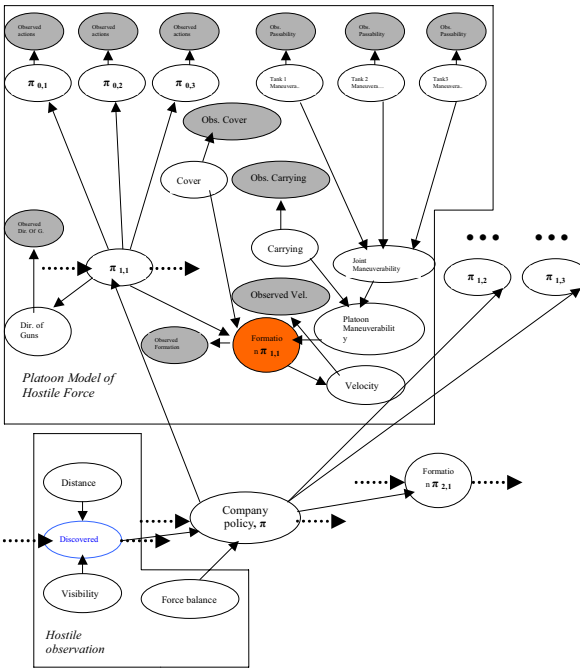


Fig. 3. The BN representing a hostile company produced using UML+BN (from [4]).

Because OPRMs distinguish classes and instances, once the classes and their probabilistic dependencies have been specified, automatic knowledge construction algorithms can be used to construct situation specific models of the battlespace as observations arrive. The UML+BN approach, however, requires the knowledge engineer to construct the UML model, and then translate the UML class diagram specifications *manually* to the equivalent BN. Which leads to another limitation of the technique. While the UML structure makes it easier to construct the large and complex BNs required for intent analysis, there is no formalism for the translation of UML class diagram specifications to the equivalent BN. The translation must be done manually by the knowledge engineer, which means that multiple distinct models could be produced from the same UML specification. So while the UML+BN technique makes *manual model construction* easier for the knowledge engineer, producing more consistent BNs and decreasing modelling time, it does not automate the intent recognition process or address the model size, complexity

and inference performance problems faced in large, complex domains.

UML+BN technique also does not allow knowledge engineers to exploit redundancies within the model. For example, when modelling the company consisting of three platoons, the OPRMs can simply instantiate 3 platoon classes. The UML+BN technique is forced to cut and paste the parts of the network modelling the platoon. This makes maintaining and modifying the model difficult; each time the reused fragment is modified, it must be updated in all the parts of the network where it is used. This approach doesn't take full advantage of the structure of the domain and limits the potential for reuse.

While UML is suitable to represent organizational structure in object-oriented fashion, is unable to represent uncertainty over the structure of the conceptual model. For example, the configuration of a tank platoon may be flexible, with the exact number of tanks and support vehicles varying from platoon to platoon. Using the UML+BN approach, there is no way of representing that Adversary A's tank platoons are more likely to have four tanks rather than three. UML also cannot represent the relationship between *attributes* of related classes (e.g. how a platoon's formation depends on the cover provided by the terrain). The UML+BN technique does not provide visual clues about how to construct these types of relationships when creating the BN model, since the UML diagrams only support the standard association, consists-of and generalization-specialization relationships. In [3,4,5], the UML specifications appear to be mainly useful for providing clues about how to fit the organizational structure of the domain together in the overall model.

In summary, OPRMs allow knowledge to be represented in a generic, structured and flexible fashion. The inheritance mechanism takes advantage of the structure of the domain and facilitates model reuse by allowing common features of a group of objects to be captured in a common superclass. The most important distinction between traditional BN and OPRMs is that OPRMs have the ability to define the dependency models at the class level. OPRMs allow the process of model construction to be automated, allowing situation specific model to be dynamically constructed from the ontology of classes. OPRMs are also flexible, allowing uncertainty about the existence, number and configuration of objects to be represented.

5. OPRM Model Construction Algorithms

5.1 Query Dependant KBMC Algorithm

The automatic model construction algorithm we have developed for OPRMs is based on an approach known as Knowledge Based Model Construction (KBMC) [8]. It uses a knowledge base to construct a standard BN tailored to the query variables. The knowledge base consists of a set of OPRM classes and instances, a set of statements assigning instances to the known values of reference attributes (e.g. Tank[1].Terrain = Terrain[1]) and a set of *inverse* statements that describe which reference attributes in instances are inverses of each other. Starting with the query variables (e.g. Company[1].CompanyIntent), the algorithm backward chains along the dependencies in the KB, extending the model to include variables related to the currently processed variable.

Algorithm KBMC(query)
begin
varsToProcess := the set of query variables
while *varsToProcess* $\neq \varnothing$
    Take first variable v $\in$ *varsToProcess*
    Generate list of parents of v
    Add link from parents to v in DAG
    If *parents* $\notin$ *varsToProcess* ***then***
        Add parents to varsToProcess
    end if
    Remove v from varsToProcess
end while
Add evidence to network
Inference with standard BN techniques
end

Algorithm 1: Query Dependant KBMC Algorithm

How the list of parents is created depends on the type of the variable. The presence of reference or number uncertainty requires the introduction of additional parent nodes to the network (see [2] for further details). We have implemented the OPRM KBMC algorithm in Matlab and used it to construct the network for the model described in Section 3. Inference was performed using the standard BN techniques implemented in Murphy's BNT [9]. A sample outcome is shown in Table 3.

The first and most important criticism of this algorithm is that while OPRMs take advantage of the structure of the domain for modelling, the KBMC algorithm does not. Nor does it facilitate reuse. All structure gained by using OPRMs to represent the observations and prior knowledge is lost as soon as this information is translated into a traditional BN. Also, KBMC is query driven. If the set of query variables changes, the entire model construction process must be rerun and a different model will result. While this characteristic is not detrimental to the current application, it can be a critical problem for other applications, e.g., for formulating situation assessments from signal data to support the situation awareness of tactical military commanders. That domain is data driven and a query driven construction process is not appropriate. We have developed the OPRM Jtree Construction algorithm described below as a solution to how to guide the construction of models for such applications.

5.2 The Junction Tree Construction Algorithm

Because the dependencies of each variable in each OPRM class are fully specified once the class is created (i.e. at design time), the OPRM JC Algorithm is able to take advantage of Bangsø's techniques [10, 11] to translate the OPRM class into a BN and then into a 'class' junction tree. That is, at design time, each OPRM class is 'precompiled' into a junction tree, once the class has been specified. Then, at run time, whenever an instance of this class is created, the junction tree for that class is plugged into the model utilizing Bangsø's plug and play techniques. Evidence is then applied to the model and inference is performed using standard BN techniques.

This algorithm provides three main benefits. Firstly, the original structure of the OPRM is maintained, secondly, reuse is facilitated, and thirdly, efficiency is gained during any modifications required to the junction tree of a class by Incremental Compilation techniques [12].

We have implemented the JC algorithm in Matlab and have applied it to models from various problem domains. An analysis of the performance of the JC and query dependant KBMC algorithms appears in [1].

We are currently examining the extension of OPRMs to incorporate identity uncertainty and their application to dynamic problems.

6. Conclusions

In this paper we have discussed the application of OPRMs to intent recognition. The OPRM approach enables both observations and a prior knowledge such as doctrine and organizational structure to be represented in a generic, structured and flexible fashion. The inheritance mechanism takes advantage of the structure of the domain and facilitates model reuse. By defining the dependency models at the class level, OPRMs allow the process of model construction to be automated, allowing situation specific model to be dynamically constructed from an ontology of classes. The implemented knowledge based model construction algorithm, while an improvement over the manual construction of models does not take full advantage of the structure of the domain or reuse. We have developed the OPRM Jtree Construction algorithm to dynamically construct models in data driven environments where the speed of model construction algorithms is important.

Languages such as OPRMs and the associated automatic model construction techniques are important in the wider context of information fusion, because while significant progress has been made developing and automating techniques applicable to Object Fusion such as object identification and tracking, substantial challenges remain in the development, automation and formalization of Situation and Impact Fusion techniques.

7. References

1. Howard, C. and M. Stumptner. *Model construction algorithms for Relational Probabilistic Relational Models.* In *Proc. FLAIRS'06, Orlando, FL, 2006. In print*.
2. Howard, C. and M. Stumptner. *Situation Assess-ments Using Object Oriented Probabilistic Relational Models*. in *Proc. 8th Int'l Conf. on Information Fusion*. Philadelphia, 2005.
3. Suzic, R., *Knowledge Representation and Stochastic Multi-Agent Plan Recognition*. Stockholm, 2005.
4. Suzic, R. *Generic Representation of Military Organisation and Military Behaviour: UML and Bayesian Networks*. in *Proc. NATO RTO Symp. on C3I and M&S Interop*. 2003.
5. Suzic, R. *Knowledge Representation, Modelling of Doctrines and Information Fusion*. in *Proc. CIMI Conference*. 2003. Enköping, Sweden.
6. Suzic, R. *Representation and Recognition of Uncertain Enemy Policies Using Statistical Models*. In *Proc. NATO RTO Symp. on Military Data and Inf. Fusion*. Prague 2003.
7. Matheus, C.J., M.M. Kokar, and K. Baclawski. *A Core Ontology for Situation Awareness*. in *Proc. 6th Int'l Conf. on Information Fusion*. 2003. Cairns, Queensland.
8. Wellman, M.P., J.S. Breese, and R.P. Goldman, *From Knowledge Bases to Decision Models.* Knowledge Eng. Review, 1992. **7**(1): p. 35-53.
9. Murphy, K., *The Bayes Net Toolbox for Matlab.* Comp Sci and Stats, 2001. **33**.
10. Bangso, O., M.J. Flores, and F.V. Jensen, *Plug and Play OO Bayesian Networks*. Proc. 10th Conf. of the Spanish Assocation for AI, LNAI 3040. 2004.
11. Bangso, O., *Object Oriented Bayesian Networks*, PhD Thesis, Aalborg U., 2004.
12. Flores, M.J., J.A. Gamez, and K.G. Olsen. *Incremental Compilation of a Bayesian Network*. in *Proc. of the 19th Conf. on Uncertainty in AI*. 2003.

On Road	Gradient	Physical Barrier	Cover	Carrying	Visibility	Direction of Platoon Guns	Tank Direction of Movement	Tank Distance to Target	P(Company Intent = Attack)
True	Slight	False	Good	Good	Good	Toward	Toward	Near	0.85
True	Slight	False	Good	Good	False	Toward	Toward	Near	0.65
False	Steep	True	Good	Good	Good	Toward	Toward	Near	0.2
True	Slight	False	Good	Neutral	Good	Away	Away	Far	0.05

Table 3. Example results from the tank company model.

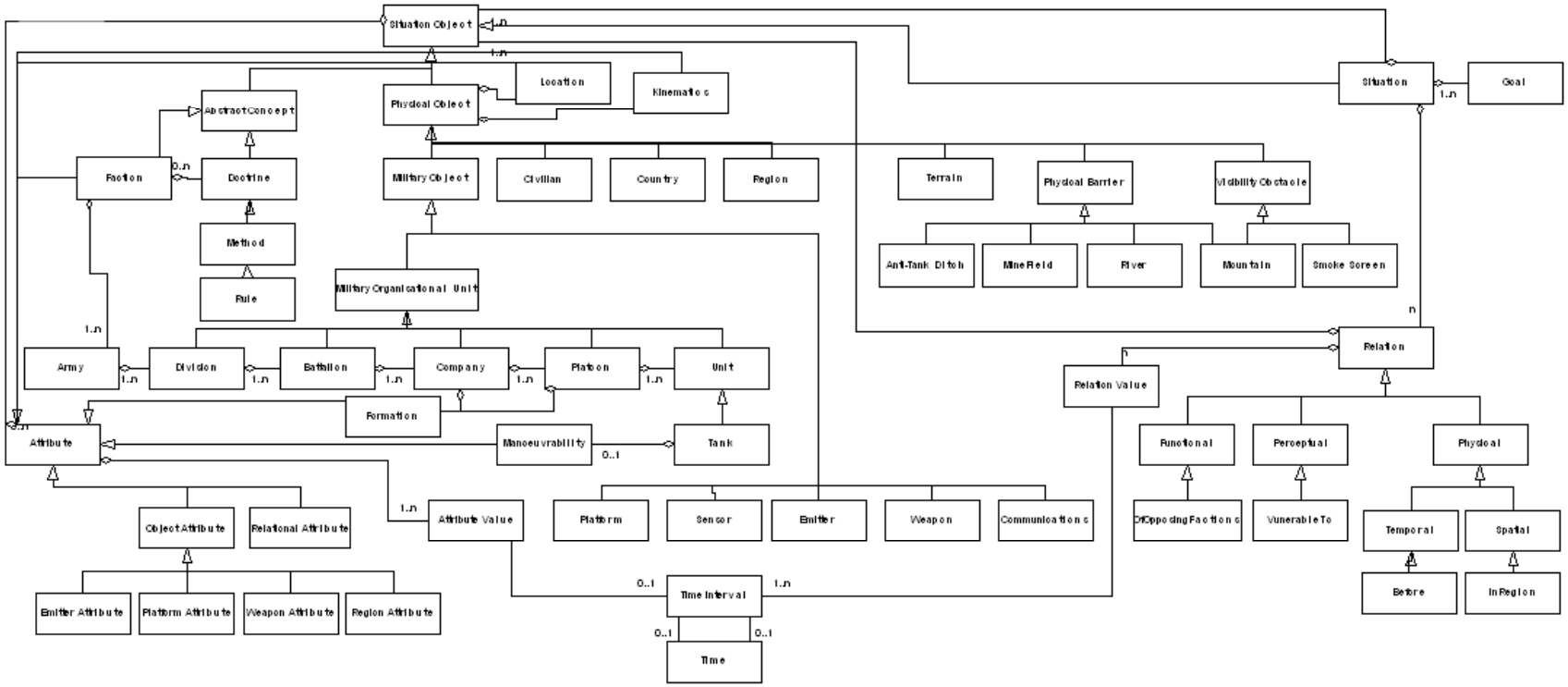


Fig. 1. The tactical domain ontology.

A Framework for Fusing Consistent Knowledge Bases Automatically

Èric Grégoire
CRIL & IRCICA
Université d'Artois
F-62307 Lens
France
gregoire@cril.univ-artois.fr

Du Zhang
Department of Computer Science
California State University
Sacramento, CA 95819-6021
USA
zhangd@ecs.csus.edu

Abstract

Knowledge-based systems play a pivotal role in the information economies. Knowledge base inconsistencies can adversely affect a system's correctness and performance. In this paper, we propose a framework for fusing several knowledge bases together to produce a consistent one. The framework is based on two cornerstones: detecting inconsistencies using algorithmic techniques to compute minimally unsatisfiable formulas, and adopting a logic-based weakening approach to restore consistency for the fused knowledge base. We study the dynamics in the framework in terms of both the model-theoretic and the fixpoint semantics for the fused knowledge base.

Keywords: knowledge base fusion, minimally unsatisfiable formulas, inconsistency detection and resolution.

1. Introduction

Knowledge-based systems play a pivotal role in today's information economies. Knowledge-based technology has found its way into so many problem domains [5] and has been the cornerstone to numerous successful applications [29, 16]. A crucial component of a knowledge-based system is its knowledge base (*KB*) that contains knowledge about a problem domain [8, 20]. Knowledge base development involves domain analysis, context space definition, ontological specification, and knowledge acquisition, codification and verification (Figure 1).

It is important to recognize the context under which domain-specific knowledge is formulated and reasoned about when developing a *KB* for an application. A context is a region in some *n*-dimensional space [19]. In the common sense *KB*, Cyc [5], there are thousands of contexts. Some of its top dimensions of the context space include time, space, logic, relations, paths, physical objects, artifacts, movement, and so forth. In a *KB* development process, domain analysis should result in identification of the region of interest in the context space. Specifying a context entails defining or locating a point or region along each of those *n* dimensions.

Clearly specifying the context space within which a *KB* is developed has a number of benefits [19]. It suppresses the bulk of irrelevant knowledge so as to concentrate on the relevant one to the problem at hand, which results in better inference and search performance. It allows for terse and sloppy communications among users and developers, and accommodates contradictory information through separating inconsistent information into different contexts. Similar or relevant assertions can be entered in the same or nearby context in terse and simpler form.

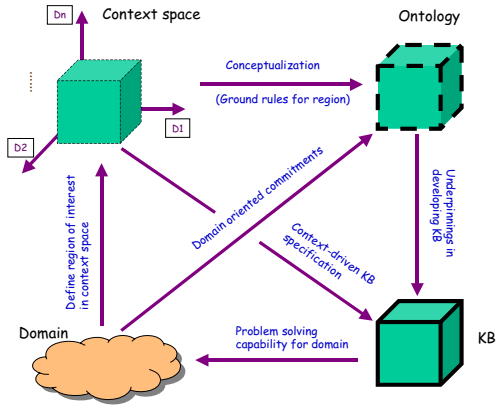


Figure 1. *KB* development process.

To categorize knowledge into contexts, we need to avoid the following granularity pitfalls: too coarse-grained is likely having no contexts at all, and too fine-grained makes it hard to find the right context for a new piece of knowledge [19].

After determining the context (or contexts) for a problem domain, ontological development is in order. An ontology is a formal, explicit specification of a shared conceptualization [3, 10, 24]. Conceptualization refers to an abstract model consisting of relevant concepts about a problem domain. Explicitly defined are the following: concepts and concept taxonomies, relationships between

and properties of concepts, and the constraints on concept use. The "shared" implies that an ontology captures consensual conceptualization accepted by groups of people. Having an ontology in place before developing a *KB* promotes reuse and sharing. There are many contemporary ontologies, methodologies and languages for building ontologies, and ontology tools [10].

After the ontology is in place, knowledge acquisition, codification and verification can be carried out to build the *KB* for a given application. Several *KB*s may need to be fused to produce an amalgamated one. Inevitably, there will be anomalies in a *KB* as a result of existing practices in its development process. Anomalies such as inconsistency can affect the correctness and performance of an intelligent system, though some systems are robust enough to perform rationally in the presence of the anomalies.

In this paper, we consider the issue of how to detect *KB* inconsistency during *KB* fusion process and how to restore consistency in the fused *KB*. We propose a framework for fusing several *KB*s together to produce a consistent one. The framework is based on two cornerstones: isolating inconsistencies using algorithmic techniques to compute *minimally unsatisfiable formulas* (MUSes), and adopting a logic-based weakening approach to restore consistency for the fused *KB*.

The paper is organized as follows. In the next section, we provide some necessary background information about Boolean forward chaining *KB*s, their semantics and inconsistencies. In Section 3, we discuss how inconsistencies can be identified through detecting MUSes in a given *KB*. Section 4 describes a technique that can resolve the detected inconsistencies in the fused *KB*. The main properties of the dynamics of this technique are then presented in Section 5. Finally in Section 6, we conclude the paper with remarks on possible future work.

2. Technical background

The focus in this paper is on forward-chaining rule-based *KB*s. A *KB* has a set of facts that is stored in a working memory (WM) and a set of rules stored in a rule base (RB). Rules represent general knowledge about an application domain. Facts in a WM provide specific information about the problem at hand and may be elicited either dynamically from the user during each problem-solving session, or statically from the domain expert during knowledge acquisition, or derived through rule deduction.

Rules and facts are formulas belonging to a Boolean language **L** of formulas over a finite alphabet **P** of Boolean variables, also called *atoms* or *propositions*. The $\wedge$, $\vee$, $\neg$ and $\Rightarrow$ symbols represent the standard conjunctive, disjunctive, negation and implication connectives, respectively. A *literal* is an atom or its negation. $\top$ [1] represents inconsistency. From a syntactical point of view, a *KB* will be a multi-set of formulas of **L** (we shall often abuse words and mention set for multi-set). In Section 4, we will consider a multi-set of rule bases $E = \{KB_1,\ldots,KB_n\}$ to be fused ($n > 1$).

We assume that rules in a *KB* have the following format: $P_1 \wedge \ldots \wedge P_n \Rightarrow R$, where P_is are the conditions (collectively, the left-hand side, LHS, of a rule), R is the conclusion (or right-hand side, RHS, of a rule). The P_is and R are literals. If the conditions of a rule are satisfied by facts in WM, then the rule is enabled and its firing deposits its conclusion into WM. A fact is represented as a atom. It specifies a proposition in a given problem domain. WM contains a collection of positive facts which are deposited through either assertion (initial or dynamically), or rule deduction using forward chaining.

A negated condition $\neg$P in the LHS of a rule is satisfied if P is not in WM. A negated conclusion $\neg$R in the RHS of a rule results in the removal of R from WM when R is in WM. Rules and negated literals can be utilized by the inference system, but never deposited into WM.

Let WM_0 denote the initial state for WM. We use WM_i (i=1,2,3…) to represent subsequent states of WM as a result of firing all enabled rules under the state of WM_{i-1}.

Actually, a language used in developing a *KB* is frequently enriched with other valuable epistemological concepts. For instance, mutual exclusive literals that cannot be true at the same time (such as "animal" and "vegetable") and incompatible ones, which are complementary pairs of synonymous literals (such as "expensive" and "$\neg$high_price"), are often explicitly or implicitly asserted [35]. In the following, the aforementioned concepts are accommodated in our proposed framework.

The semantics of *KB* can be described using several techniques. The first one is the *fixpoint semantics* [34]. For a given *KB*, we can define a transformation T_{KB} in the following way. A single step of T_{KB} to *KB* amounts to generating a set of ground literals, denoted as $\vdash_{WMi}$ RB, which is obtained by firing all enabled rules in RB under WM_i. It can be shown that T_{KB} is monotonic and has a least fixpoint $lfp(T_{KB})$ with regard to the partial order $\leq_k$ (knowledge ordering) [9]. $lfp(T_{KB})$ contains all the derivable conclusions and is denoted as $\vdash_{lfp}$. Let g be a literal, $KB \vdash_{lfp} g$ when $g \in lfp(T_{KB})$. When both g and $\neg$g (or a pair of contradicting literals) are in $lfp(T_{KB})$, the *KB*

[1] Here, we adopt the usual convention from many four-valued logics that makes use of $\top$ to denote *overdefined* or *contradiction* in the truth value set of {*true*, *false*, $\bot$, $\top$}. Let us stress that the bottom symbol $\bot$ is often used to represent contradiction in standard binary logic.

is said to be (*lfp*-)inconsistent or contains (*lfp*-)inconsistent knowledge. This is denoted as $KB \vdash_{lfp} \top$. *KB* is (*lfp*-)consistent *iff* $KB \nvdash_{lfp} \top$.

If we treat a *KB* as a set of logic formulas, we can then apply the model-theoretic approach to define the semantics for the *KB*. Let Ω denote the set of all *interpretations* of **L**, which are functions assigning either *true* or *false* to every atom. A *model* ξ of *KB* is an interpretation of Ω that satisfies every formula of *KB*. The set of models of *KB* will be denoted $[[KB]]$. $KB \models g$ when the literal *g* is entailed from *KB*, i.e., when *g* is true in all models of *KB*. *KB* is *inconsistent* or *unsatisfiable* when $[[KB]]$ is empty, also denoted by $KB \models \top$.

Though we use the terms of rules/facts and formulas interchangeably, the two semantics of $\vdash_{lfp}$ and $\models$ do not necessarily coincide for a given *KB* in the sense that we do not have $KB \vdash_{lfp} g$ *iff* $KB \models g$. For example, let $KB = \{a, b \Rightarrow \neg a\}$. $KB \models \neg b$, but we do not have $KB \vdash_{lfp} \neg b$. Indeed, using a forward chaining mechanism, we are not able to derive $\neg b$ from *KB* although $\neg b$ is a logical consequence of *KB*.

To ascertain the presence of inconsistency in a *KB*, we utilize a concept called *minimally unsatisfiable formulas* or MUSes for a *KB*. A MUS captures a smallest possible set of formulas in a *KB* that encode some contradiction.

Definition 2.1. A MUS Γ of a *KB* is a set of formulas s.t. $(\Gamma \subseteq KB)$ and $([[\Gamma]] = \varnothing)$ and $(\forall S \subset \Gamma: [[S]] \neq \varnothing)$

Identifying MUSes for a *KB* can exhibit a high worst case computational complexity. Indeed, even in the simple Boolean clausal framework, determining whether a set of clauses is a MUS is D^P-complete [26], and checking whether a formula belongs to the set of MUSes of contradictory sets of clauses is in Σ^p_2 [7]. Moreover, the number of MUSes for a set of *n* clauses is $C_n^{n/2}$. However, this high complexity often deflates to a tractable one in real-life. New techniques to compute MUSes are now available [14, 15]. They are based on the use of a (failed) local search for consistency as an oracle to locate MUSes. More precisely, they make use of a *critical clause* concept and a heuristic stating that the formulas that are most often falsified during this failed search for consistency belong most probably to these MUSes. Extensive computational studies [14, 15] on difficult benchmarks [6, 30] show that this approach is frequently the most efficient one with respect to all current competing ones.

Though defined under $\models$, MUS can be adapted with respect to the *lfp*-semantics as well. To this end, the (un)satisfiable condition is to be replaced by the absence (presence) of contradictory facts in the least fixpoint. Throughout the remainder of the paper, we shall use $MUS_{\vdash lfp}$ and $MUS_{\models}$ to distinguish the two concepts. When there is no confusion, we use the term MUS as a generic one for both $MUS_{\models}$ and $MUS_{\vdash lfp}$.

A *KB* can contain several MUSes. The inconsistent part of *KB* is denoted $\cup$MIN-UNSAT(*KB*) and is defined as follows.

Definition 2.2. The *inconsistent part* of a *KB*, denoted $\cup$MIN-UNSAT(*KB*), is the set-theoretic union of all MUSes of *KB*.

Since MUSes can exhibit non-empty intersections, the concept of *inconsistent cover* [15] proves valuable in that it does not require all MUSes to be made explicit, but provides us with enough mutually independent inconsistency causes that need to be addressed to restore consistency. This concept can be defined with both the $MUS_{\vdash lfp}$ and $MUS_{\models}$ characterizations.

Definition 2.3. An *inconsistent cover IC* of a *KB* is a set-theoretic union of MUSes of *KB* s.t. $KB \setminus IC$ is consistent.

3. Detecting inconsistency via MUSes

As illustrated earlier, the relations $\vdash_{lfp}$ and $\models$ do not necessarily coincide. What this illuminates is the discrepancy between the inconsistency characterizations under the two. Given a $KB = \{a, b \Rightarrow \neg a, d, \neg b \Rightarrow \neg d\}$. Clearly, *KB* is logically inconsistent w.r.t. $\models$, but not under $\vdash_{lfp}$. In this specific case, although the forward chaining mechanism will not allow us to derive any contradiction, we argue that the knowledge in the *KB* is inherently inconsistent[2] and this information will be important to the knowledge engineer. Thus, inconsistency under the *lfp* semantics is a sufficient but not necessary condition to inconsistency under the model-theoretic semantics.

Property 3.1. Whenever $KB \vdash_{lfp} \top$, we also have $KB \models \top$, but not conversely. Whenever Γ is a $MUS_{\vdash lfp}$ of *KB*, we also have that Γ is a $MUS_{\models}$ of *KB*, but not conversely.

Because algorithmic techniques are already in place to determining $MUS_{\models}$ [14, 15], the aforementioned property affords us a direct way to compute MUSes under $\vdash_{lfp}$.To this end, $MUS_{\models}$ are first computed. Any $MUS_{\models}$ that exhibits a least fixpoint that contains contradictory literals is also a $MUS_{\vdash lfp}$. Let us stress that the cost of this second step is negligible from a computational point of view since $MUS_{\vdash lfp}$ can be computed in polynomial time with respect to the size of the $MUS_{\models}$, which is often small in real-life applications [14].

[2] Conceivably, *KB* can be evolved through some equivalence preserving transformations (say, contrapositive law) into $KB' = \{a, a \Rightarrow \neg b, d, d \Rightarrow b\}$ where $KB' \vdash_{lfp} \top$.

Table 1 illustrates the results of MUS detection algorithm in [14, 15] on some *difficult KBs* from the Dimacs [6] and SATLIB [30] benchmarks.

Table 1. MUS detection algorithm benchmark results.

KB	#atoms	#clauses	(#atoms,#clauses) in MUS
dp02u01	213	376	(47,51)
dp03u02	478	1007	(327,760)
23.cnf	198	474	(165,221)
42.cnf	378	904	(315,421)
fpga10_11_uns_rcr	220	1122	(110,561)(110,561)
fpga11_12_uns_rcr	264	1476	(132,738)(132,738)
ca008	130	370	(110,255)
C202_FW_UT_2814	2038	11352	(15,18)

These are very difficult *KB*s with respect to the MUSes detecting problem. For example, the fpga11_12_unsr_rcr contains two MUSes with each involving 738 different clauses, i.e., the *minimal* number of clauses that can be involved in a proof of inconsistency is 738. Although the computational time to solve these benchmarks remains tractable, it should be noted that the size of MUSes for real-life instances is often very small, making the computation even more time-efficient [15].

4. Consistency restoring

When several consistent *KB*s are fused, additional inconsistencies can occur due to the interaction of the *KB*s. Once inconsistencies are detected through MUSes, the next step is how to resolve the inconsistencies. In this section, we describe a technique that allows the consistency to be restored in an automatic way in the fused *KB* without the intervention of the knowledge engineer. In particular, our focus is on the dynamics of restoring consistency during the fusion process when several *KB*s are fused successively.

Many techniques have been proposed to fuse knowledge bases, in the logic programming context (see e.g. [1, 31]) or in other logic-based settings (see e.g. [4, 13, 17, 18, 21, 22]). In [11, 12], an original way has been introduced to fuse standard logic *KB*s in an automatic manner. This approach restores consistency in the fused *KB* if MUSes are detected during the fusion process. Its main feature lies in a generic technique to weaken formulas belonging to MUSes. This contrasts with the common practice of dropping conflicting formulas by existing logic-based techniques to fuse *KB*s [2, 13].

We adopt the technique in [11, 12] as part of our framework. The essence of this technique involves weakening the contradictory formulas in MUSes by augmenting them with additional literals. Some properties in the fusion process can be observed through reasoning with those literals. In the rest of the section, we give a brief overview of the approach, and then describe how it is incorporated into the forward chaining rule-based systems.

Finally we describe some properties in terms of the $\vdash_{lfp}$ and $\models$ semantics.

We assume here that every KB_i to be fused is given an enumeration of its m formulas. In the general case, this general enumeration has no meaning and is not intended to represent any respective importance of formulas in KB_i. An ordered list ($Flag^{i\text{-}1}$,…, $Flag^{i\text{-}m}$) of atoms, called *Flags*, is associated to this enumeration. Flags will be introduced inside *KB*s using our fusion process or in a way to enforce a preference of one *KB* over the others, only.

Now, assume that we need to fuse two consistent *KB*s, namely KB_1 and KB_2 and that $[[KB_1 \cup KB_2]] = \emptyset$. To restore consistency, we weaken *some* formulas in $\cup$MIN-UNSAT($KB_1 \cup KB_2$). Depending on the way to select those formulas, we obtain a series of weakening policies.

Definition 4.1. The set of weakened formulas of *KB*, noted *WKF(KB)*, is a subset of $\cup$MIN-UNSAT(*KB*) s.t. for every MUS of *KB*, it contains at least one formula from each *KB* taking part in that MUS.

Actually, there can often exist many possible candidate *WKF(KB)*. In the following, we assume that such a set is selected, and concentrate on the common properties of all corresponding approaches. Accordingly, a $\cup_{weaken1}$ fusion by weakening operator was defined as follows in a CNF setting [12].

Definition 4.2. When $[[KB_1 \cup KB_2]] \neq \emptyset$,
then $KB_2 \cup_{weaken1} KB_1$ is defined as $KB_1 \cup KB_2$.
When $[[KB_1 \cup KB_2]] = \emptyset$,
then $KB_2 \cup_{weaken1} KB_1$ is defined as
$(KB_1 \cup KB_2) \setminus \text{WKF}(KB_1 \cup KB_2) \cup$
$\{f \vee Flag^{1\text{-}i}$ s.t. $f \in \text{WKF}(KB_1 \cup KB_2) \cap KB_1$
and i is the rank of f in $KB_1\} \cup$
$\{f \vee Flag^{2\text{-}j}$ s.t. $f \in \text{WKF}(KB_1 \cup KB_2) \cap KB_2$
and j is the rank of f in $KB_2\}$

The $\cup_{weaken1}$ operator is easily iterated.

Definition 4.3. Let $n > 2$. We denote KB for
$KB_n \cup (KB_{n\text{-}1} \cup_{weaken1} \ldots \cup_{weaken1} KB_1)$
When $[[KB]] \neq \emptyset$, then $KB_n \cup_{weaken1} (KB_{n\text{-}1} \cup_{weaken1} \ldots$
$\cup_{weaken1} KB_1)$ is defined as KB
When $[[KB]] = \emptyset$, then $KB_n \cup_{weaken1} (KB_{n\text{-}1} \cup_{weaken1} \ldots$
$\cup_{weaken1} KB_1)$ is defined as
$KB \setminus \text{WKF}(KB)$
$\cup \{f \vee Flag^{1\text{-}i}$ s.t. $f \in \text{WKF}(KB) \cap KB_1$
and i is the rank of f in $KB_1\}$
$\cup \ldots$
$\cup \{f \vee Flag^{n\text{-}j}$ s.t. $f \in \text{WKF}(KB) \cap KB_n$
and j is the rank of f in $KB_n\}$

Now, it is easy to reformulate these definitions in the context of forward chaining rule-based systems and $\vdash_{lfp}$,

and get a $\cup_{weaken2}$ operator. For instance, we can decide to mark rules that occur in the inconsistent part of the set-theoretic union of the *KB*s by inserting their corresponding flags in the LHS. In such a way, provided that WM_0 is consistent, $\vdash_{lfp}$ consistency of the merged *KB*s is ensured. From a computational point of view, it is also sufficient to consider one inconsistency cover to regain consistency. However, in this case, a unique possible fused *KB* is guaranteed only when a unique IC exists.

Example 4.1. Assume $KB_1 = \{a, a \Rightarrow b\}$ and $KB_2 = \{c, c \Rightarrow \neg b\}$. We observe that both of the following results hold: $[[KB_1 \cup KB_2]] = \emptyset$ and $[KB_1 \cup KB_2] \vdash_{lfp} \top$. There is one unique MUS and thus one unique inconsistent cover IC in $KB_1 \cup KB_2$, namely $KB_1 \cup KB_2$ itself. Accordingly, weakening all rules in the IC, the resulting fused *KB* will be given by $KB_1 \cup_{weaken2} KB_2 = \{a, a \wedge Flag^{1\text{-}2} \Rightarrow b, c, c \wedge Flag^{2\text{-}2} \Rightarrow \neg b\}$. Clearly, for the fused *KB* to be consistent, it requires that $Flag^{1\text{-}2}$ and $Flag^{2\text{-}2}$ must not be true at the same time. Enforcing one of these two flags (e.g. $Flag^{1\text{-}2}$) to be true amounts to preferring its corresponding *KB* (KB_1) over the other one (KB_2).

It is important to note that flags are not mere markers. Since they become part of rules, they can be reasoned about under $\models$. However, using the forward chaining mechanism only, we cannot infer any piece of information pertaining to these flags, unless some flags are already in WM_0. Such a weakening rule schema can be traced to the exception-blocking refinement rule mechanism that has been introduced in the frameworks of non-monotonic logics [23] and in the model-based diagnosis [28]. However, its use to restore consistency w.r.t. both the $\vdash_{lfp}$ and $\models$ semantics, and the study of its dynamics in an iterated fusion process are, to the best of our knowledge, new.

The above operators obey some nice properties.

Properties 4.1. Assume $KB_1,\ldots,KB_n$ are *n* consistent rule bases.

1. $[[KB_n \cup_{weaken2} \ldots \cup_{weaken2} KB_1]] \neq \emptyset$
2. Let $i \in [1..n]$ we have that
   $[[KB_n \cup_{weaken2} \ldots \cup_{weaken2} KB_1 \cup \{ Flag^{i\text{-}j} \forall j\}]] \neq \emptyset$
3. $WFK(KB_n \cup KB_{n-1})$ does not have any rule containing a flag.
4. $\nexists\ Flag^{i\text{-}j}$ s.t. $KB_n \cup_{weaken2} \ldots \cup_{weaken2} KB_1 \models \neg Flag^{i\text{-}j}$
5. $\nexists\ Flag^{i\text{-}j}$ s.t. $KB_n \cup_{weaken2} \ldots \cup_{weaken2} KB_1 \vdash_{lfp} \neg Flag^{i\text{-}j}$
6. $\nexists\ Flag^{i\text{-}j}$ s.t. $KB_n \cup_{weaken2} \ldots \cup_{weaken2} KB_1 \vdash_{lfp} Flag^{i\text{-}j}$

Property 4.1.1 ensures that the resulting fused *KB* is consistent with both the $\vdash_{lfp}$ and $\models$ semantics. Property 4.1.2 shows how a preference for some initial *KB* over the other ones can be enforced, by setting all its related flags to true (e.g. by inserting them in WM_0). Property 4.1.3 ensures that whenever a formula is weakened, it cannot belong to a future MUS in any subsequent fusion step. Properties 4.1.4 and 5 are best interpreted as follows. Asserting $Flag^{i\text{-}j}$ to be true enforces a preference for KB_i, and once this preference is asserted, the fusion process cannot question it in the future. Property 4.1.6 ensures that the fusion process cannot guarantee by itself an explicit preference for a given KB_i that would modify the set of inferences according to $\vdash_{lfp}$ (let us stress that Property 4.1.6 does not hold for $\models$ [12]).

5. Dynamics in the fusion schema

Although Properties 4.1.5 and 6 ensure that no explicit preference for one KB_i can be inferred using the forward chaining mechanism, this does not force the fusion schema to be unbiased with respect to the several rule-based systems to be fused in an iterated fashion. Indeed, flags are introduced in the formulas themselves and interact with the knowledge that these latter ones represent.

Example 5.1. Assume we iterate the fusion process that began in Example 4.1. with $KB_3 = \{d, d \Rightarrow b\}$. Clearly, $KB_3 \cup_{weaken2} (KB_1 \cup_{weaken2} KB_2) = \{a, a \wedge Flag^{1\text{-}2} \Rightarrow b, c, c \wedge Flag^{2\text{-}2} \Rightarrow \neg b, d, d \Rightarrow b\}$. Indeed, $KB_3 \cup (KB_1 \cup_{weaken2} KB_2)$ is both $\models$ and $\vdash_{lfp}$ consistent.

The above example illustrates that when a conflict between two *KB*s has been neutralized using flags during a previous fusion process, nothing prevents a piece of information from a third *KB* that is consistent with the previous resulting fused knowledge from conducting one branch of the alternative of the initial conflict to be adopted.

The following theorem states the conditions for the iterated fusion process to be associative, thus entails no preference for one of the KB_i. Intuitively, it requires that all MUSes (from the set-theoretic union of the initial KB_is) that share a non-empty intersection to consist of formulas coming from the same KB_is.

Theorem 5.1. Let MUS denote either $MUS_{\models}$ or $MUS_{\vdash lfp}$. Let $\mathbf{K} = \{K$ s.t. K is a MUS of $KB_n \cup \ldots \cup KB_1\}$, $\cup_{weaken1}$ is associative w.r.t. $KB_1,\ldots,KB_n$ *iff*
$\nexists\ (K_i,K_j) \in \mathbf{K}x\mathbf{K}$ s.t.

- $K_i \cap K_j \neq \emptyset$
- and $\exists\ KB_r \in \{KB_1,\ldots,KB_n\}$ s.t. $(\exists\ f \in KB_r \cap K_i$ and $\nexists\ g \in KB_r \cap K_j)$

6. Conclusions and future work

The results presented in this paper have been obtained in the propositional logic framework. From a conceptual point of view, they could be easily extended to the first-order case, i.e. to a framework that allows the use of predicates over finite domains in rules. However, it remains to be seen how the algorithmic techniques to

extract MUSes could be extended accordingly, with the help of an instantiation schema into the Boolean case, provided that any variable is bound to a specific limited-size domain. The technical results presented in this paper could be valuable tools in the pursuit from knowledge base verification and validation perspective (see e.g. [25, 27, 32, 33]). Since logical consistency checking is only a basic tool in that respect, extending the results here into those problem domains is an exciting path for future research, especially in the context of various intelligent systems where rule bases are only a basic component.

References

1. C. Baral, S. Kraus, J. Minker and V.S. Subrahmanian, "Combining knowledge bases consisting of first-order theories", *Computational Intelligence*, vol. 8(1), pp. 45-71, 1992.
2. I. Bloch and A. Hunter (guest eds), "Fusion: general concepts and characteristics", *Int. Journ. of Intelligent Systems*, vol. 16, 2000.
3. B. Chandrasekaran, J.R. Josephson and V.R. Benjamins, "What are ontologies, and why do we need them?", *IEEE Intelligent Systems*, vol. 14(1), pp. 20-26, 1999.
4. L. Cholvy, "Reasoning about merging information", *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, vol. 3, pp. 233-263, 1998.
5. Cycorp, http://www.cyc.com/
6. Dimacs Benchmarks on SAT (ftp://dimacs.rutgers.edu/pub/challenges/satisfiability/)
7. T. Eiter and G. Gottlob, "On the complexity of propositional knowledge base revision, updates and counterfactual", *Artificial Intelligence*, vol. 57, pp. 227-270, 1992.
8. R. Fagin, J.Y. Halpern, Y. Moses and M.Y. Vardi, *Reasoning about knowledge*, The MIT Press, Cambridge Mass., 1995.
9. M. Fitting, "Fixpoint semantics for logic programming: a survey", *Theoretical Computer Science*, vol. 278(1-2), pp. 25-51, 2002.
10. A. Gomez-Perez, M. Fernandez-Lopez and O. Corcho, *Ontological Engineering*, Springer-Verlag, 2004.
11. Ė. Grégoire, "An unbiased approach to iterated fusion by weakening", *Information Fusion*, vol 7(1), pp. 35-40, 2006.
12. Ė. Grégoire, "About the dynamics of iterated knowledge fusion by weakening", *Proc. of the IEEE IRI'05 Conference*, D. Zhang et al. (eds), pp. 326-331, 2005.
13. Ė. Grégoire and S. Konieczny, "Logic-based approaches to information fusion", *Information Fusion*, vol. 7(1), pp. 4-18, 2006.
14. Ė. Grégoire, B. Mazure and C. Piette, "Extracting MUSes", *Proc. of the 17th European Conference on Artificial Intelligence (ECAI'2006)*, Trento, 2006.
15. Ė. Grégoire, B. Mazure and C. Piette, "Tracking MUSes and strict inconsistent covers", *CRIL Research Report 2006-RR-02*, 2006.
16. IBM's research project on Information Economies, http://www.research.ibm.com/infoecon/index.html
17. S. Konieczny and R. Pino Perez, "Merging with integrity constraints", *Proc. of Ecsqaru'99*, pp. 233-244, LNCS 1638, Springer, 1999.
18. S. Konieczny, "On the difference between merging knowledge bases and combining them", *Proc. of KR'2000*, pp. 135-144, 2000.
19. D. Lenat, "The dimensions of context-space", *CYCorp Report*, October 1998.
20. H.J. Levesque and G. Lakemayer, *The Logic of Knowledge Bases*, The MIT Press, Cambridge Mass., 2000.
21. J. Lin, "Integration of weighted knowledge bases", *Artificial Intelligence*, vol. 83, pp. 363-378, 1996.
22. J. Lin and A.O. Mendelson, "Merging databases under constraints", *Int. Journ. of Cooperative Information Systems*, 7(1), pp. 55-76, 1998.
23. J. McCarthy, "Applications of circumscription to formalizing common-sense knowledge", *Artificial Intelligence*, vol. 28, pp. 89-116, 1986.
24. D.E. O'Leary, "Using AI in knowledge management: knowledge bases and ontologies", *IEEE Intelligent Systems*, vol. 13(3), pp. 34-39, 1998.
25. D.E. O'Leary and A. Preece, *AAAI Workshop on Verification and Validation of Knowledge-Based Systems*, 1998.
26. C. Papadimitriou and D. Wolfe, "The complexity of facets resolved", *Journ. of Computer and System Sciences*, vol. 37(1), pp. 2-13, 1988.
27. R. Plant and G. Antoniou, *Knowledge Based Systems (Special issue on verification and validation)*, vol. 12(1-2) 1999.
28. R. Reiter, "A theory of diagnosis from first principles", *Artificial Intelligence*, vol. 32, pp. 57-95, 1987.
29. R.G. Ross, *Principles of the Business Rule Approach*, Addison-Wesley, 2003.
30. SATLIB Benchmarks on SAT (http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/benchm.html)
31. V.S. Subrahmanian, "Amalgamating knowledge bases", *ACM Transactions on Database Systems*, vol. 19(2), pp. 291-331, 1994.
32. J. Vanthienen and F. van Harmelen (eds), *Proc. of the Fourth European Symp. on the Validation and Verification of Knowledge-Based Systems EUROVAV'97*, Leuven, Belgium, 1997.
33. A. Vermesan and F. Coenen (eds), *Verification and Validation of Knowlege Based Systems: Theory, Tools and Practice*, Kluwer Academic Publishers, 1999.
34. D. Zhang, "Fixpoint semantics for rule-based anormalities", *Proc. of the Fourth IEEE Int. Conf. on Cognitive Informatics*, Irvine, pp.10-17, 2005.
35. D. Zhang and Luqi, "Approximate declarative semantics for rule base anomalies", *Knowledge-based Systems*, vol. 12(7), pp. 341-353, 1999.

User Profiling in the Chronobot/Virtual Classroom System

Xin Li and Shi-Kuo Chang
Department of Computer Science,
University of Pittsburgh, USA,
{flying, chang}@cs.pitt.edu

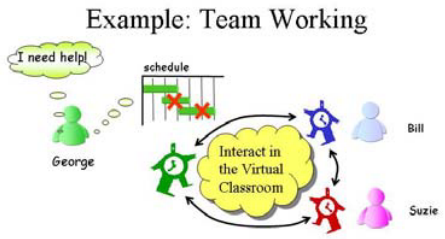
Abstract

The Chronobot/Virtual Classroom (CVC) system is a novel time knowledge exchange platform where any pair of users can exchange their time and knowledge. User profile that contains user attributes, preferences, and learning patterns serves as a primary basis to identify exchange partners and determine exchange rates. In this paper, we described the methodology to acquire knowledge about users i.e. user profile from their activities. The association between user profile and user behaviors (e.g. online reading, chatting and time/knowledge exchanging) is identified by several feedback indicators extracted from browsing history, chatting session and exchange transaction. A linear learning model is constructed to fuse multiple feedback indicators to infer user preference. The methods utilizing user profile to identify the exchange partners and determine the exchange rate are also described in detail.

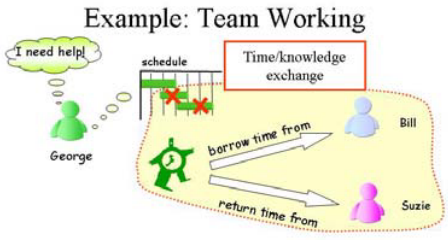
1. Introduction

Comparing with the traditional face-to-face style teaching and learning, e-Learning is indeed a revolutionary way to provide education in life long term. Nowadays more and more people have benefited from various e-Learning programs. However, present-day e-Learning systems are still too rigid and do not lend themselves to the peer-to-peer learning in which any users can exchange their knowledge with any others. Chronobot/Virtual Classroom (CVC) [3] is a novel time knowledge exchange platform where any pair of users can exchange their time and knowledge. The chronobot is a time manage tool for storing and borrowing time. Using chronobot one can borrow time from some one and return time to the same person or someone else. The virtual classroom is a versatile communication tool that combines the functions of web browser, chatting room, white board, and multimedia display. The CVC system is an integration of chronobot and virtual classroom that allows users freely to switch between these two applications and get maximum benefits from both.

For example, illustrated in figure 1, George, Bill, and Suzie are all students who are doing a group project together in a graphics design course in which they use the CVC system to collaborate with each other. The whole project is divided into several tasks, each of which is mainly assigned to one person. George meets a problem in his task and can not solve it by himself. So he interacts with Bill and Suzie in the virtual classroom, and eventually they help him out. However, in order to keep workload even among teammates, George has to put in efforts either in the past or in the future to help Bill and Suzie. The chronobot serves as a platform for them to do such time and knowledge exchanges which could have significant value for many applications. Many more interesting scenarios can be found in [3].



(a)



(b)

Figure 1. Application of the CVC system: (a) Communication in Virtual Classroom; (b) Time and knowledge exchange in Chronobot

Generally speaking, a transaction of the time/knowledge exchange in the CVC system includes the following steps:

1. Identify a slice of time or knowledge for exchange;
2. Search for exchange partner or partners;
3. Perform time or knowledge exchange through bidding and negotiation;
4. Manage the exchange slice of time or knowledge;

5. Provide feedback on the results.

In these processes, user profile that contains user attributes, preferences and learning patterns plays a vital role because:

- It provides fundamental information to identify time or knowledge for exchange and exchange partners.
- It serves as a primary basis to determine time/knowledge exchange rate.

In this paper, we describe an approach to acquire knowledge about users represented by user profile based upon user activities such as web browsing, chatting and time/knowledge exchanging in the CVC system. We believe that our work has significant value, because the acquisition of knowledge about users is a key process not only in our system but also in many other applications. For example, the recommendation systems [1, 4, 6, 9] mainly depend on user profiles in terms of similarity and differences to provide particular suggestions. The personalized web search engine [10] can construct user profiles from browsing history and consequently provide personalized results to match the information needs of individuals. Compared to these applications, an effective user profiling is much more feasible in our system because of the following two reasons:

1. Time/knowledge exchange (i.e. peer-to-peer learning) is a much more continuous process than the activities (e.g. online news reading and web searching) in many other applications.
2. Multiple data sources can be employed to assess user profile in our system. For example, in addition to browsing history, chatting session and knowledge/time exchange transaction can also serve as important input sources for the profiling process.

User preference is key information in user profile. As far as we studied, the majority of user profiling approaches mainly depends on the user feedbacks to retrieve user preference. The feedback can be assessed explicitly by rating, or implicitly by the user behaviors such as print and save. In this paper, we are not advocating either of these two approaches because both of them have significant advantage and disadvantages [10]. Instead we propose a methodology which can combine multiple feedback measures to get more complete and accurate assessment. User preference can be inferred on the basis of data from three sources, i.e. browsing history, chatting session, and knowledge/time exchange transaction. A linear learning model is constructed to fuse all the related data for the inference of user preference. Five feedback indicators – reading time, scroll number and print/save from browsing history, relational index from chatting session, and the exchange index from knowledge/time exchange transaction serve as input variables of the model. Demonstrated by the experiments in the prototype system, the proposed model can infer user preference much more accurately than any single one of these indicators. The applications of user profile – to identify the exchange partners and determine the exchange rate are also described in detail. A preliminary report about this work has been published in [7].

The rest of this paper is organized as follows: two basic concepts *user profile* and *ontology knowledge base* in our system are described in section 2 and 3 respectively. Section 4 discusses the measures to identify the association between user activities and preferences, where five implicit feedback indicators are defined. The learning model to fuse these indicators for a final assessment of user preference is described in Section 5. The application of user profile is described in Section 6. In Section 7, our prototype system and experiments in it are described. The related research is discussed in Section 8, followed by brief conclusions in Section 9.

2. User Profile

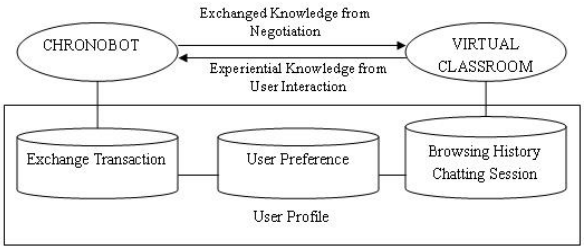


Figure 2 User Profile in the CVC System

The *user profile* is the physical realization of the *user model*, which is an abstraction of the user preferences and characteristics. Shown in figure 2, in the CVC system a user profile *upf* is organized as a 6-tuple:

upf = < *id, user-attributes, browsing-history, chatting-session, exchange-transaction, preference*>

- *id* is a unique identification number.
- *user-attributes* is a vector $A(u) = (a_1(u), a_2(u), \ldots, a_n(u)) = (x_1, x_2, \ldots, x_n)$, where $a_i(u) = x_i$ is the i^{th} attribute for the user u – it could be user's name, expertise level, schedule and so on.
- *browsing-history* is a set of the pages which user has visited. For the purpose of user profiling, the corresponding behaviors are recorded for each page, e.g. reading time, number of scrolls and print/save.
- *chatting-session* is a set of conversations which user participated in the virtual classroom. All the contents are recorded in natural language.
- *exchange-transaction* is a set of time/knowledge exchange transactions which user performed in the chronobot.
- *preference* is a vector $P(u) = (p_1(u), p_2(u), \ldots, p_n(u)) = (y_1, y_2, \ldots, y_n)$, where $p_i(u) = y_i$ is the preference of user on the i^{th} topic. The topics are defined by the *ontology knowledge base* (which will be discussed in the next section).

When a user first registers the CVC system, the user is asked to enter information such as personal data, areas of expertise, levels on these areas and so on. The *user*

profile manager provides an HTML front-end using which new users can register themselves with the system. During the registration, the *user profile manager* collects important information from the users such as the user id, name, address, credit card details, areas of experience, levels, skill set, the hourly rate, e-mail address and so on. The rationale behind having the credit card information is that if the user defaults in time/knowledge exchange transactions, then his/her credit card is billed depending upon the number of hours defaulted.

User preference is the most valuable information in user profile. However, it is usually hard to be assessed directly, because in many cases it is difficult to request users to express their interests explicitly -- it is simply too much work for them. Furthermore, users may change their preferences upon time, and hence they can not be assessed statically. For this reason, in our approach user preference is not directly input by users, but implicitly inferred from user behaviors.

3. Resource Organization

The learning resources in the CVC system are all web-based multimedia materials, which are organized by the ontology knowledge base (OKB). All the topics and their relations (i.e. mainly *subtopic* relation) are described in the OKB. For example, a small part of the OKB in our system is shown in Figure 3.

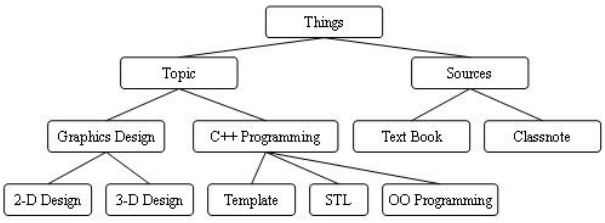


Figure 3: A small part of OKB tree

Based upon the OKB, a learning resource *lr* in the CVC system is defined as a tuple:

$$lr = <url, topic, keywords>$$

where

- *url* is the universal resource locator which is an identification string of *lr*.
- *topic* is a concept in the OKB which is the subject of *lr*.
- *keywords* is the set of key words of *lr*. They are closely related to the content of *lr*, but usually not in the OKB.

In practice, in order to build an efficient connection between the resources and user profile, two kinds of mapping are employed in our system:

- One-to-many mapping from topics to URLs.
- One-to-many mapping from topics to keywords.

Topics	Keywords	URLs
Graphics Design	Layout, Rendering, ···	http://www.cs.pitt.edu/flying/CVC/graphics/*
C++ Programming	Template, STL, ···	http://www.cs.pitt.edu/flying/CVC/cplus/*

Figure 4. An example of topics to URLs and keywords mappings

Figure 4 shows examples of these two kinds of mapping. The star "*" represents any combination of characters in URLs. The rationale behind these mappings is that they can greatly facilitate the aggregation of feedback indicators on topics.

4. Feedback Extraction

There are three user feedback sources in our system: browsing history, chatting session, and time/knowledge exchange transaction. In this section, we discuss the methods to extract the knowledge about users, i.e. user preference, from these three sources respectively.

Totally five distinct feedback indicators are defined. In fact, we can design these indicators to be as complex as is necessary for the intended application. However for practicality it is important to keep them simple and manageable.

4.1 Browsing History

There is no doubt that browsing history conveys significant information for inferring user preference. However, it is hard to determine interests of users just based on the pages they visited, because it always happens that users open a page they don't like or just by mistake. Aiming at the more accurate assessment, three feedback indicators are defined based upon browsing history:

1. Reading Time

Usually if users spend longer time on reading about a topic, it means that they have more interest on it. For this reason, we record reading time of the user u as a vector $RT(u) = (<lr_1, rt_1>, <lr_2, rt_2>, \ldots, <lr_n, rt_n>)$, lr_i is the learning resource user u has visited, rt_i is the corresponding reading time.

2. Number of Scrolls

Definitely the scroll either by mouse or PageDown/PageUp key on a page is a signal of interest. For this reason, we record the number of scrolls for user u as vector $SC(u) = (<lr_1, sc_1>, <lr_2, sc_2>, \ldots, <lr_n, sc_n>)$, lr_i is the learning resource user u has visited, sc_i is the corresponding number of scrolls.

3. Print/Save

In most cases, printing or saving a page is a strong signal of interest. For this reason, we record the set $PS(u) = (<lr_1, ps_1>, <lr_2, ps_2>, \ldots, <lr_n, ps_n>)$, lr_i is the resource which user u has visited, ps_i is 1 if it has been printed/saved, 0 otherwise.

4.2 Chatting Session

The experiences accumulated in the virtual classroom are among the most valuable assets for preference inference. In practice the experiences are the stored transcripts of the virtual classroom sessions. These transcripts are represented as XML documents. In fact we consider everything that is exchanged or recorded in the chronobot/virtual classroom system as some form of XML document.

The Relational Index *RI* is built to support easy access of the accumulated learning experiences. The session transcripts (XML documents) are stored in an experience-base. The Relational Index is then constructed. It relates learning experiences to user preferences in the user profile. For example, if x_1, ..., x_n are keywords specified in the topic keywords mapping discussed in section 3, the Relational Index can be used to find u_j, the user most closely related to the specified topics.

We can also use the Relational Index *RI* to relate users to keywords and/or users to users. In other words, *RI* is used to form an *association* in the information exchange process among users.

The *RI* is updated each time a new session transcript is created. The transcript is analyzed with respect to a set of pre-specified keywords x_1, ..., x_n in the topic keywords mapping. If a dialog of user u_j in the transcript involves a keyword x_k, we can store a new record [x_k; u_j; p] in *RI* where the frequency p is set to 1, or update p if such a record already exists. Similarly if a dialog between two users u_j and u_j in the transcript involves a keyword x_k, we can store a new record [x_k; u_j; u_j; p] in *RI* where the frequency p is set to 1, or update p if such a record already exists.

For example the transcript is as follows:

```
George: Do you think we need to add 3D graphics to
the presentation?

Suzie: No, I don't think so.  But the layout can be
improved.

George: That is good, because I still cannot find
a person to do 3D graphics.
```

The pre-specified keyword set is:

```
{layout, graphics, 3D graphics}
```

The Relational Index, after the processing of the above transcript, contains the following records as well as other previously entered records:

```
[3D graphics; George; 2]
[3D graphics; George; Suzie; 1]
[layout; Suzie; 1]
[layout; George; Suzie; 1]
```

4.3 Exchange Transaction

The transactions of time/knowledge exchange can also be a significant indicator of user preference. In our system, the exchange of time/knowledge is implemented as a bidding process: a person who needs help from others can start a bid, providing the task description, the required knowledge, and the amount of time needed. Anyone else can place a bid to offer his/her time. A successful exchange/bid transaction includes at least the following information: bid starter, bid winner, task description, keywords, and time amount. All these information is stored in XML file in practice.

The Exchange Index *EI* is built to easy access of the accumulated exchange history. Similar as Relational Index, the *EI* is used to relate user to keywords and/or user to users. It is updated every time a new transaction is created. The task description is analyzed with the respect to a set of pre-specified keywords x_1, ..., x_n in the topic keywords mapping. If a transaction of user u_j is related to a keyword x_k, we can store a new record [x_k; u_j; t] in EI where time t is set to the time amount of the transaction, or update t by adding the amount if such a record already exists. The time amount is positive if user u_j borrows the time to others, otherwise it is negative. Similarly if a transaction between two users u_j and u_j involves a keyword x_k, we can store a new record [x_k; u_j; u_j; t] in EI which means user u_j has borrowed t amount of time from u_j.

For example, George starts a bid as follows:

```
Bid Task: Help to improve the layout in a 3D graphics
design;
Time Amount: 8 hours;
```

The pre-specified keyword set is the same with the example in section 4.2.

Through a bidding process, Bill can offer 5 hours, and the rest 3 hours help can be done by Suzie. The Exchange Index, after processing these transactions, contains the following records as well as other previously entered records:

```
[3D graphics; George; -8]
[3D graphics; Bill; 5]
[3D graphics; Suzie; 3]
[3D graphics; George; Bill; 5]
[3D graphics; George; Suzie; 3]

[layout; George; -8]
[layout; George; 5]
……
```

The Exchange Index may contain records relating multiple (more than two) users or multiple (more than one) keywords as well as the Relational Index.

5. Preference Assessment

As discussed in section 2, the preference of user *u* can be expressed as a vector $P(u) = (p_1(u), p_2(u), \ldots, p_n(u)) = (y_1, y_2, \ldots, y_n)$, where $p_i(u) = y_i$ is the preference of user *u* on the i^{th} topic. The topics are described in the OKB. Give a user *u* and a topic *t*, the preference of *u* on *t* can be inferred by the feedback indicators which are described in section 4. Using the mappings from topics to URLs and keywords, the five feedback indicators can be easily collected for each topic. Single topic could have multiple learning resources, and hence it could have many feedback indicators. For this reason, the following five variables are defined for any pair of *u* and *t* by aggregation and normalization:

1. *rt* – the average reading time per 1000 words;
2. *sc* – the average number of scroll per 1000 words;
3. *ps* – the average number of print/save per view.
4. *ri* – the average relational index per 1000 words in chatting sessions.
5. *ei* – the average exchange index per 100 hours in time knowledge exchanges.

A linear model, which can predict user preference based on feedback variables, is constructed using linear regression. In this model, the five variables mentioned above serve as the input variables, the output variable -- user preference *up* can be assessed by a linear combination of the input variables:

$$up = \alpha_1 \cdot rt + \alpha_2 \cdot sc + \alpha_3 \cdot ps + \alpha_4 \cdot ri + \alpha_5 \cdot ei + \beta + e \quad (1)$$

in which:

- $\alpha_i (i = 1,2,...,5)$ and β are coefficient variables, which can be assessed through linear regression on training data.
- e is the residual with the mean zero for all the training data.

For more information about linear prediction and regression, please refer to [5].

6. Application of User Profile

In our system, user profile can be applied to identify appropriate partners and determine exchange rate in time/knowledge exchange transactions. Meanwhile, user profile provides valuable information about ontology knowledge, which can possibly help to construct and update the ontology knowledge base.

6.1 Identification of Exchange Partners

The requirements for exchange partners are described in the exchange task. However, frequently there are so many candidates who are qualified. It is very important to choose a proper one who is most likely willing to perform the exchange. Assuming that users sharing more interests on the topics have more chances to do the time knowledge exchange, a similarity function S is defined on users. Given an exchange task t, users u and v are characterized by task related preferences (x1, ..., xn), and (y1, ..., yn) from their profiles. Denote the corresponding average preferences are $\varpi_1, \varpi_2, ..., \varpi_n$, the similarity function S on u and v is defined using Pearson correlation coefficient:

$$S(v,u) = \frac{\sum_{i=1}^{n} (x_i - \varpi_i) \times (y_i - \varpi_i)}{\sqrt{\sum_{i=1}^{n} (x_i - \varpi_i)^2 \times \sum_{i=1}^{n} (y_i - \varpi_i)^2}} \quad (2)$$

A list of exchange candidates can be sorted using the similarity function *S*. The one with the highest similarity is always the first candidate to be approached for the exchange.

6.2 Determination of Exchange Rates

User profile provides the fundamental information to determine the exchange rates between two users. Give an exchange task *t*, users *u* and *v* are characterized by task related attributes $(x_1, \ldots, x_n)$, and $(y_1, \ldots, y_n)$ from their profiles. The attributes could be the preferences, expertise levels and so on. For the two corresponding attributes x_i and y_i, the information distance measure is denoted by d_i (x_i, y_i), where d_i is between 0 and 1 (a metric).

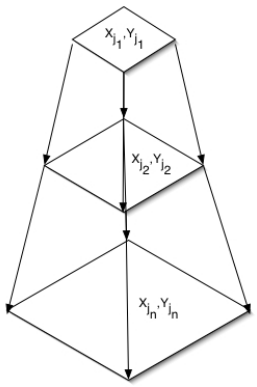


Figure 5. Determination of exchange rate.

The exchange rate between users *u* and *v*, is denoted as follows:

$$Exchange(u,v) = e^{1/\sum C_{ji} \times d(x_{ji}, y_{ji})} \quad (3)$$

where the summation is over all the terms $C_{ji} * d(x_{ji}, y_{ji})$, and C_{ji} is a scaling constant.

We now illustrate the concept by presenting an example. Let us assume that George and Bill are both media artists and the two agents' primary skill matches. Therefore C_1 $d_1(x_1, y_1) = 0$. If the primary skill does not match, C_1 d_1 (x_1, y_1) becomes a big number. For instance, C_1 is 10,000 and d_1 is between 0 and 1, in this case close to 1. Then C_1 $d_1(x_1, y_1)$ is close to 100,000 and the exchange rate is close to 1. No need to continue.

The two agents' familiarity with subject area also is comparable, so $c_2\ d_2\ (x_2, y_2) = 0$. If the familiarly does not match, then $C_2\ d_2\ (x_2, y_2)$ becomes a big number. For instance, C_2 is 1,000 and d_2 is between 0 and 1, in this case close to 1. Then $C_2\ d_2\ (x_2, y_2)$ is close to 1,000 and the exchange rate is close to 1. No need to continue either.

Finally, the two agents differ in secondary skill. Therefore $C_3\ d_3\ (x_3, y_3)$ is small and we have an exchange rate that reflects the difference in the two agents' secondary skill. Notice the index function takes care of the re-arrangement of the relative importance of the n attributes. The constants C_j are also important. They take care of the relative scaling of the various attributes. Figure 10 illustrates the determination of the exchange rate as a dynamic process of comparing different attributes to identify the ones that really matter. Once such attributes are identified, the users can negotiate to determine the exchange rate.

6.3 Information Exchange with the OKB

Ontology knowledge plays an important role in the assessment of user profile. At the same time, the user profile retrieved from user experiences is among the most valuable assets which describe the ontology. It is always possible to construct and update ontology based upon user profile. We believe that the information exchange between user profile and ontology knowledge has significant importance, because for a long time the construction of ontology knowledge remains a difficult problem [8]. Particularly, it could be extremely cumbersome for end users to construct and maintain an ontology knowledge base by themselves. However, it is much easier for them to implicitly manipulate ontology through the time/knowledge exchanges. Here we present some preliminary works along this line by the following examples; more works in this direction will be the subject of our future research.

For example, several web pages about *"Standard Template Library"* (STL), which is a subtopic of *"C++ programming"*, are newly added into our system. In order to provide a guideline for information delivery on this new topic, we can update the ontology knowledge base by adding *"Standard Template Library"* as a subtopic of *"C++ programming"*. As a result, the users who have evident preferences on the topic *"C++ programming"* probably will be suggested for it. Alternatively, supposing that there is no efficient way to manipulate ontology, this kind of update with the new topic in ontology knowledge base can be automatically performed based upon user activities. Assuming that there are a large number of users who have visited the new web pages on *"Standard Template Library"*, their most common preference can be identified from their user profiles, which is *"C++ programming"* in this case. By knowing this, the system can infer that *"Standard Template Library"* is a subtopic of *"C++ programming"* and update the ontology knowledge base automatically.

Obviously, this kind of inferences is not always correct. However, with the accumulation of user experiences, user profile retrieved from them will be more and more accurate, and hence the system will eventually be able to construct reasonable ontology knowledge through this way.

7. Prototype System and Experiments

A prototype CVC system has been implemented and used in real applications. Verified by the experiments in the prototype system, the proposed model can infer user preference much more accurately than any single one of these indicators.

7.1 Prototype System

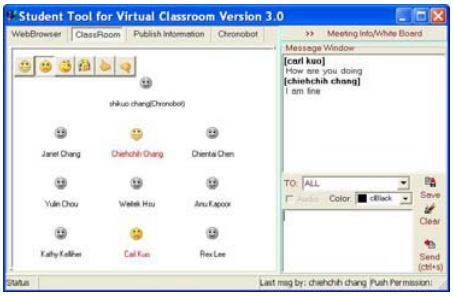
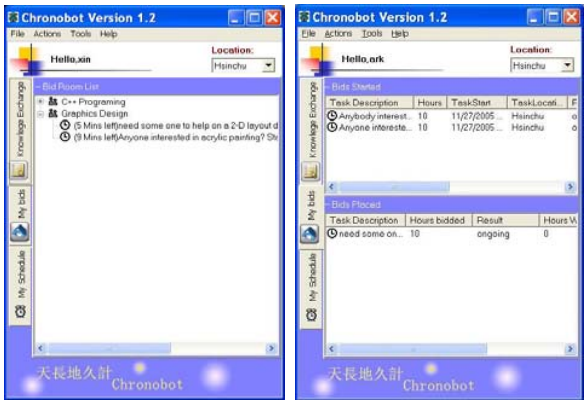


Figure 6. An example of virtual classroom

The prototype of virtual classroom system is illustrated in figure 6. Basically it is a universal communication tool. The users (students and/or instructors) represented by emotive icons can join a virtual classroom to exchange information including text messages, web pages, sketches, and audio/video clips. The system will automatically record user activities (e.g. browsing and chatting) and related information (e.g. reading time, scroll, etc.).



(a) (b)

Figure 7. An example of chronobot, (a): bidding room for time knowledge exchange; (b): transaction records.

The prototype of chronobot is illustrated in figure 7.

Basically, it is a tool to exchange time and knowledge though bidding processes. Users can start new bids in different bidding rooms (figure 7(a)). A simple example of such bids is "I need some one help me on 2-D layout design for 5 hours". Users can respond these requests by placing their own bids. For example, a user can respond by "I can contribute 2 hours on it". The details about these transactions are recorded in user profiles (figure 7(b)).

7.2 Experiments and Result Analysis

The prototype systems are used by the students from several classes in University of Pittsburgh. The following experiment has been done to obtain the training data for our linear learning model:

- Step 1: Encourage students to use the prototype system as much as they can
- Step 2: Select a set of students who used the systems for more than a reasonable time (i.e. 72 hours). Asked them to seriously indicate their interests from 0 (least) to 5 (most) on every topic in which they have involved.
- Step 3: Input the preference indicated by the students with the corresponding feedback variables into the linear model (1), assess the coefficient variables using linear regressions.
- Step 4: Separate the testing and training data sets, verify the linear model by the cross validation.

UserID	Topic ID	Preference	*rt*(sec.)	*sc*	*ps*	*ri*	*ei*
006	073	3	65	0.2	0.07	2.6	5.4
017	135	1	23	0.1	0	1.2	0
017	009	5	107	0.1	0.34	5.3	0
...	...	...	...	...	...	...	...

Figure 8. Examples of experimental results

The data of the experiment are illustrated in figure 8. For each pair of user and topic, there are five feedback variables (described in Section 5) and preference. Input by these variables, the learning model in (1) can be constructed using least-squares linear regression [5].

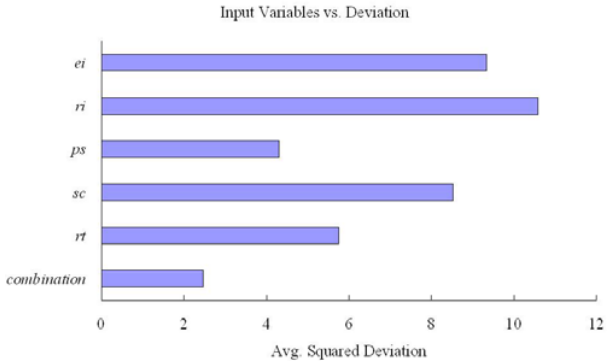


Figure 9. Input variables vs. Deviation

Using the cross validation, the model can be evaluated by the average squared deviation between the preference inferred by the linear model and the one that users explicitly indicated. Illustrated in figure 9, the combination of multiple feedbacks is compared with single feedback: the bar labeled with "combination" represents the result input by all five feedback indicators, while other bars are the results with only one indicator as the input variable i.e. *rt*, *sc*, *ps*, *ri*, and *ei* respectively. Shown in figure 9, the combination of the indicators can assess much more accurate preference than any single indicators.

8. Related Research and Discussion

Because the time and knowledge exchange is indeed a new concept, we can hardly find similar approaches with our CVC system up to the time of this paper. However, there are numerous user profiling approaches in other application areas. As the related research, recommendation system is one of the most important applications, where user profiling is the key process collecting user feedback for items in a given domain and assessing user preference in terms of similarities and differences to determine what to recommend. Depending on underline technique, recommendation systems can be divided into collaborative filtering-based [6] content-based [4] and hybrid [1, 8] approaches. Classified by means to acquire feedback, they can be categorized as explicit rating [1, 6, 8], implicit rating [6] and no rating needed [4] systems.

In fact, user's feedbacks are so important that only very few content-based recommendation systems require neither explicit rating nor implicit rating. For example, Surflen [4] is a recommendation system using data mining techniques to assess the association rules on web pages through user's browsing history without the feedbacks. However, it is hard to find exact interests of users just based on the browsing history, since it always happens that users open a page they don't like or just by mistake. This problem becomes even more severe in the situation that the system is sparsely used.

GroupLens system [6], which filters Usenet news, is a collaborative filtering system using n-nearest neighbor-based algorithm. In this algorithm, user profile is assessed based on a subset of appropriate n users similar to this user. The early version of GroupLens gathers user's feedback only by explicit rate. However, observing the extra costs of the explicit rating, in the latest version it also uses reading time as an implicit indicator.

Fab system [1] is also using the collaborative filtering model, meanwhile introducing the content analysis by a "topic" filtering. Web pages are initially ranked by the topic filter and then sent to user's personal filters. Users are required to give an explicit rate, and this feedback is used to modify both the personal filter and the original topic filter.

Compared to the approaches above, our approach can be classified as a hybrid approach. We assess the similarity

of users in the exact same way as the collaborative filter-based systems (shown in equation (2)). Meanwhile, the mappings from topics to keywords and URLs based on the OKB serve as simple and effective content-based filters. Particularly, our approach proposes a novel way to infer the user preference: multiple implicit feedback indicators are employed at the same time, and the final assessment are retrieved using a linear learning model. As a result, more accurate preference can be assessed without introducing any extra costs for users.

9. Conclusions

The CVC system is a novel time knowledge exchange platform that supports peer-to-peer learning. In this paper, we describe an approach to acquire and utilize the knowledge about users i.e. user profile in the CVC system. User preference – the most important information in user profile, is inferred from multiple data sources i.e. browsing history, chatting session, and time knowledge exchange transaction. The methods utilizing user profile to identify the exchange partners and determine the exchange rate are also described in detail.

10. Acknowledgements

This research is supported in part by the Industry Technology Research Institute (ITRI) and the Institute for Information Industry (III) of Taiwan. We would like to thank Chieh-Chih Chang and Jui-Hsin Huang for the valuable discussion and comments.

References

[1] M. Balabanovi and Y. Shoham. Fab: content-based, collaborative recommendation. Communication of the ACM, 40(3):66–72, 1997.

[2] M. Blochl, H. Rumershofer, and W. Wob. Individualized e-Learning systems enabled by a semantically determined adaptation of learning fragments. In Proceeding of the 14th international workshop on Database and Expert Systems Applications, pp. 640–645, 2003.

[3] S. K. Chang. A chronobot for time and knowledge exchange. In Proceeding of the 17th International Conference on Software Engineering and Knowledge Engineering, pp. 3–10, Taipei, Taiwan, Jul. 2005.

[4] X. Fu, J. Budzik, and K. J. Hammond. Mining navigation history for recommendation. In Proceeding of the 5th international conference on Intelligent user interfaces, pp. 106–112, 2000.

[5] T. Hastie, R. Tibshirani, and J. Friedman. The elements of statistical learning: data mining, inference, and prediction. Springer, New York, 2001.

[6] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl. Grouplens: applying collaborative filtering to usenet news. Communication of the ACM, 40(3):77–87, 1997.

[7] X. Li and S. K. Chang. A Personalized E-Learning System Based on User Profile Constructed Using Information Fusion. In Proceeding of the 11th International Conference on Distributed Multimedia Systems. pp. 109-114, Banff, Canada, Sep. 2005.

[8] Deborah L. McGuinness. Ontologies come of age. In Dieter Fensel, Jim Hendler, Henry Lieberman, and Wolfgang Wahlster, editors, Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential. MIT Press, Boston, NY, 2002.

[9] S. E. Middleton, N. R. Shadbolt, and D. C. D. Roure. Ontological user profiling in recommender systems. ACM Trans. Inf. Syst., 22(1):54–88, 2004.

[10] K. Sugiyama, K. Hatano, and M. Yoshikawa. Adaptive web search based on user profile constructed without any effort from users. In Proceedings of the 13th international conference on World Wide Web, pages 675–684, 2004.

Falsification of OTSs by Searches of Bounded Reachable State Spaces

Kazuhiro Ogata, Weiqiang Kong, Kokichi Futatsugi
School of Information Science, JAIST
{ogata, weiqiang, kokichi}@jaist.ac.jp

Abstract

We propose a way to search a bounded reachable state space of an observational transition system, or an OTS for a counterexample that the OTS satisfies an invariant property. Two case studies are also reported in which the proposed solution has been applied to an incorrect mutual exclusion protocol and the NSPK authentication protocol.

1. Introduction

In *the OTS/CafeOBJ method* [10], a system is modeled as an *observational transition system*, or an *OTS*, the OTS is specified in *CafeOBJ* [3], an algebraic specification language, and it is verified that the OTS satisfies a property using the CafeOBJ system as a proof assistant. We have conducted some case studies, among which are [9, 11], to demonstrate the effectiveness of the method and refine it.

Since the CafeOBJ system does not have any model checking facilities, there is no way to conveniently find any errors lurked in CafeOBJ specifications of OTSs. *Maude* [2], which is a sibling language of CafeOBJ, has model checking facilities such as the Maude model checker [4] and the `search` command. Although the state space of a system to be model checked by Maude does not have to be finite, its reachable state space should be finite. Note that the `search` command can be used to search an infinite state space of an OTS for a counterexample that the OTS satisfies an invariant property, but the termination is not guaranteed. It is also desirable to know that there exists no such a counterexample in a bounded reachable state space of the OTS.

The reachable state space of an OTS is generally infinite, even if the number of some entities such as processes is made finite. Therefore, a solution should be invented. The proposed solution in this paper is to search a bounded reachable state space of an OTS for a counterexample that the OTS satisfies an invariant property, which has been inspired by *Bounded Model Checking*, or *BMC* [1].

2. Observational Transition Systems (OTSs)

We suppose that there exists a universal state space denoted Υ and that each data type used in OTSs is provided. The data types include Bool for truth values. A data type is denoted D_*.

Definition 1 (OTSs) *An OTS $\mathcal{S}$ is $\langle \mathcal{O}, \mathcal{I}, \mathcal{T} \rangle$ such that*

- *$\mathcal{O}$: A finite set of observers. Each* observer *is an indexed function $o_{x_1:D_{o1},\ldots,x_m:D_{om}} : \Upsilon \rightarrow D_o$ that has m indexes $x_1, \ldots, x_m$ whose types are $D_{o1}, \ldots, D_{om}$. The equivalence relation ($\upsilon_1 =_{\mathcal{S}} \upsilon_2$) between two states $\upsilon_1, \upsilon_2 \in \Upsilon$ is defined as $\forall o_{x_1,\ldots,x_m} : \mathcal{O}. (o_{x_1,\ldots,x_m}(\upsilon_1) = o_{x_1,\ldots,x_m}(\upsilon_2))$, where $\forall o_{x_1,\ldots,x_m} : \mathcal{O}$ is the abbreviation of $\forall o_{x_1,\ldots,x_m} : \mathcal{O}. \forall x_1 : D_{o1} \ldots \forall x_m : D_{om}$.*
- *$\mathcal{I}$: The set of initial states such that $\mathcal{I} \subseteq \Upsilon$.*
- *$\mathcal{T}$: A finite set of transitions. Each* transition *is an indexed function $t_{y_1:D_{t1},\ldots,y_n:D_{tn}} : \Upsilon \rightarrow \Upsilon$ that has n indexes $y_1, \ldots, y_n$ whose types are $D_{t1}, \ldots, D_{tn}$ provided that $t_{y_1,\ldots,y_n}(\upsilon_1) =_{\mathcal{S}} t_{y_1,\ldots,y_n}(\upsilon_2)$ for each $[\upsilon] \in \Upsilon/=_{\mathcal{S}}$, each $\upsilon_1, \upsilon_2 \in [\upsilon]$ and each $y_k : D_{tk}$ for $k = 1, \ldots, n$. $t_{y_1,\ldots,y_n}(\upsilon)$ is called* the successor state *of υ wrt $\mathcal{S}$. Each transition $t_{y_1,\ldots,y_n}$ has the condition $c\text{-}t_{y_1:D_{t1},\ldots,y_n:D_{tn}} : \Upsilon \rightarrow \mathrm{Bool}$, which is called* the effective condition *of the transition. If $c\text{-}t_{y_1,\ldots,y_n}(\upsilon)$ does not hold, then $t_{y_1,\ldots,y_n}(\upsilon) =_{\mathcal{S}} \upsilon$.*

Example 1 (Incorrect ticketing lock) Each process i is supposed to execute the following program repeatedly:

Loop
l1: *ticket*[i] := *tvm*;
l2: *tvm* := *tvm* + 1;
l3: **repeat until** *ticket*[i] = *turn*;
Critical section;
cs: *turn* := *turn* + 1;

Initially each process i is at label l1, *ticket*[i] is 0 for each i, *tvm* is 0, and *turn* is 0. This program called the (incorrect) ticketing lock is modeled as an OTS $\mathcal{S}_{\mathrm{Ticket}}$.

Let Pid, Nat and Label be the types for process IDs, natural numbers, and labels (l1, l2, l3 and cs). $\mathcal{S}_{\mathrm{Ticket}}$ is $\langle \mathcal{O}_{\mathrm{Ticket}}, \mathcal{I}_{\mathrm{Ticket}}, \mathcal{T}_{\mathrm{Ticket}} \rangle$ such that

$$\begin{aligned}
\mathcal{O}_{\mathrm{Ticket}} \triangleq \{&\mathrm{pc}_{i:\mathrm{Pid}} : \Upsilon \to \mathrm{Label}, ticket_{i:\mathrm{Pid}} : \Upsilon \to \mathrm{Nat},\\
&\mathrm{tvm} : \Upsilon \to \mathrm{Nat}, \mathrm{turn} : \Upsilon \to \mathrm{Nat}\}\\
\mathcal{I}_{\mathrm{Ticket}} \triangleq \{&\upsilon_{\mathrm{init}} \in \Upsilon \mid \forall i : \mathrm{Nat}.\, (\mathrm{pc}_i(\upsilon_{\mathrm{init}}) = \mathrm{l1}) \wedge\\
&\forall i : \mathrm{Nat}.\, (\mathrm{ticket}_i(\upsilon_{\mathrm{init}}) = 0) \wedge\\
&(\mathrm{tvm}(\upsilon_{\mathrm{init}}) = 0) \wedge (\mathrm{turn}(\upsilon_{\mathrm{init}}) = 0)\}\\
\mathcal{T}_{\mathrm{Ticket}} \triangleq \{&\mathrm{get}_{j:\mathrm{Pid}} : \Upsilon \to \Upsilon, \mathrm{inc}_{j:\mathrm{Pid}} : \Upsilon \to \Upsilon,\\
&\mathrm{enter}_{j:\mathrm{Pid}} : \Upsilon \to \Upsilon, \mathrm{leave}_{j:\mathrm{Pid}} : \Upsilon \to \Upsilon\}
\end{aligned}$$

The transitions are defined as follows:

• get_j : c-$\mathrm{get}_j(\upsilon) \triangleq (\mathrm{pc}_j(\upsilon) = \mathrm{l1})$. If c-$\mathrm{get}_j(\upsilon)$, then

$\mathrm{pc}_i(\mathrm{get}_j(\upsilon)) \triangleq$ **if** $i = j$ **then** l2 **else** $\mathrm{pc}_i(\upsilon)$,
$\mathrm{ticket}_i(\mathrm{get}_j(\upsilon)) \triangleq$ **if** $i = j$ **then** $\mathrm{tvm}(\upsilon)$ **else** $\mathrm{ticket}_i(\upsilon)$,
$\mathrm{tvm}(\mathrm{get}_j(\upsilon)) \triangleq \mathrm{tvm}(\upsilon)$, and $\mathrm{turn}(\mathrm{get}_j(\upsilon)) \triangleq \mathrm{turn}(\upsilon)$.

• inc_j : c-$\mathrm{inc}_j(\upsilon) \triangleq (\mathrm{pc}_j(\upsilon) = \mathrm{l2})$. If c-$\mathrm{inc}_j(\upsilon)$, then

$\mathrm{pc}_i(\mathrm{inc}_j(\upsilon)) \triangleq$ **if** $i = j$ **then** l3 **else** $\mathrm{pc}_i(\upsilon)$,
$\mathrm{ticket}_i(\mathrm{inc}_j(\upsilon)) \triangleq \mathrm{ticket}_i(\upsilon)$,
$\mathrm{tvm}(\mathrm{inc}_j(\upsilon)) \triangleq \mathrm{tvm}(\upsilon) + 1$, and $\mathrm{turn}(\mathrm{inc}_j(\upsilon)) \triangleq \mathrm{turn}(\upsilon)$.

• enter_j : c-$\mathrm{enter}_j(\upsilon) \triangleq (\mathrm{pc}_j(\upsilon) = \mathrm{l3} \wedge \mathrm{ticket}_j(\upsilon) = \mathrm{turn}(\upsilon))$. If c-$\mathrm{enter}_j(\upsilon)$, then

$\mathrm{pc}_i(\mathrm{enter}_j(\upsilon)) \triangleq$ **if** $i = j$ **then** cs **else** $\mathrm{pc}_i(\upsilon)$,
$\mathrm{ticket}_i(\mathrm{enter}_j(\upsilon)) \triangleq \mathrm{ticket}_i(\upsilon)$,
$\mathrm{tvm}(\mathrm{enter}_j(\upsilon)) \triangleq \mathrm{tvm}(\upsilon)$, and $\mathrm{turn}(\mathrm{enter}_j(\upsilon)) \triangleq \mathrm{turn}(\upsilon)$.

• leave_j : c-$\mathrm{leave}_j(\upsilon) \triangleq (\mathrm{pc}_j(\upsilon) = \mathrm{cs})$. If c-$\mathrm{leave}_j(\upsilon)$, then

$\mathrm{pc}_i(\mathrm{leave}_j(\upsilon)) \triangleq$ **if** $i = j$ **then** l1 **else** $\mathrm{pc}_i(\upsilon)$,
$\mathrm{ticket}_i(\mathrm{leave}_j(\upsilon)) \triangleq \mathrm{ticket}_i(\upsilon)$,
$\mathrm{tvm}(\mathrm{leave}_j(\upsilon)) \triangleq \mathrm{tvm}(\upsilon)$, and
$\mathrm{turn}(\mathrm{leave}_j(\upsilon)) \triangleq \mathrm{turn}(\upsilon) + 1$.

□

Definition 2 (Reachable states) *Given an OTS $\mathcal{S}$,* reachable states *wrt $\mathcal{S}$ are inductively defined:*

- *Each $\upsilon_{\mathrm{init}} \in \mathcal{I}$ is reachable wrt $\mathcal{S}$.*
- *For each $t_{y_1,\ldots,y_n} \in \mathcal{T}$ and each $y_k : D_{tk}$ for $k = 1\ldots, n$, $t_{y_1,\ldots,y_n}(\upsilon)$ is reachable wrt $\mathcal{S}$ if υ is reachable wrt $\mathcal{S}$.*

Let $\mathcal{R}_{\mathcal{S}}$ be the set of all reachable states wrt $\mathcal{S}$. $\mathcal{R}_{\mathcal{S}}$ may be called the reachable state space wrt $\mathcal{S}$. □

Predicates whose types are $\Upsilon \to \mathrm{Bool}$ are called *state predicates*. We suppose that each state predicate p considered in this paper has the form $\forall z_1 : D_{p1} \ldots \forall z_M : D_{pM}.\, P(\upsilon, z_1, \ldots, z_M)$, where $\upsilon, z_1, \ldots, z_M$ are all variables in p and $P(\upsilon, z_1, \ldots, z_M)$ does not contain any quantifiers.

Definition 3 (Invariants) *Any state predicate $p : \Upsilon \to \mathrm{Bool}$ is called* invariant *wrt $\mathcal{S}$ if p holds in all reachable states wrt $\mathcal{S}$, i.e. $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$.* □

Both $\mathrm{tvm}(\upsilon) \geq 0$ and $\mathrm{turn}(\upsilon) \geq 0$ are invariant wrt $\mathcal{S}_{\mathrm{Ticket}}$. But, $\forall i, j : \mathrm{Pid}.\, ((\mathrm{pc}_i(\upsilon) = \mathrm{cs} \wedge \mathrm{pc}_j(\upsilon) = \mathrm{cs}) \Rightarrow (i = j))$ is not invariant wrt $\mathcal{S}_{\mathrm{Ticket}}$. The last predicate will be referenced by $\mathrm{MX}(\upsilon)$.

Definition 4 (Execution fragments) *Given an OTS $\mathcal{S}$, execution fragments wrt $\mathcal{S}$ are inductively defined:*

- *Each $\upsilon_{\mathrm{init}} \in \mathcal{I}$ is an execution fragment (to υ_{init}) wrt $\mathcal{S}$.*
- *For each $t_{y_1,\ldots,y_n} \in \mathcal{T}$ and each $y_k : D_{tk}$ for $k = 1\ldots, n$, $\upsilon_0, \ldots, \upsilon_m, t_{y_1,\ldots,y_n}(\upsilon_m)$ is also an execution fragment (to $t_{y_1,\ldots,y_n}(\upsilon_m)$) wrt $\mathcal{S}$ if $\upsilon_0, \ldots, \upsilon_m$ is an execution fragment wrt $\mathcal{S}$.*

Let $\mathcal{EF}_{\mathcal{S}}$ be the set of all execution fragments wrt $\mathcal{S}$. □

Proposition 1 (Reachable states & execution fragments) *(1) For each reachable state $\upsilon \in \mathcal{R}_{\mathcal{S}}$, there exists an execution fragment to υ wrt $\mathcal{S}$, and (2) for each execution fragment $\upsilon_0, \ldots, \upsilon_m \in \mathcal{EF}_{\mathcal{S}}$, each υ_k is reachable wrt $\mathcal{S}$ for $k = 0, \ldots, m$.*

Proof (1) By mathematical induction on υ. (2) By mathematical induction on m. □

Given an execution fragment $e \in \mathcal{EF}_{\mathcal{S}}$, let $\mathrm{depth}(e)$ denote the length of the execution fragment, e.g. $\mathrm{depth}(\upsilon_0, \ldots, \upsilon_n) = n$, and let $\mathrm{ef2set}(e)$ denote the set of the states that appear in e, e.g. $\mathrm{ef2set}(\upsilon_0, \ldots, \upsilon_n) = \{\upsilon_0, \ldots, \upsilon_n\}$. Let $\mathcal{EF}_{\mathcal{S},\leq n}$ be $\{e \in \mathcal{EF}_{\mathcal{S}} \mid \mathrm{depth}(e) \leq n\}$.

Definition 5 (Bounded reachable state space) *Let $\mathcal{R}_{\mathcal{S},\leq n}$ denote $(\bigcup_{e \in \mathcal{EF}_{\mathcal{S},\leq n}} \mathrm{ef2set}(e))$, which is called* the (*n*-)bounded reachable state space *wrt $\mathcal{S}$.* □

From Prop. 1, it is clear that every $\upsilon \in \mathcal{R}_{\mathcal{S},\leq n}$ is reachable wrt $\mathcal{S}$. For a set $\mathcal{A} \subseteq \Upsilon$ of states to be (in)finite wrt $\mathcal{S}$ means that $\mathcal{A}/{=_{\mathcal{S}}}$ consists of (in)finite elements.

Theorem 1 (Sufficient condition that $\mathcal{R}_{\mathcal{S},\leq n}$ is finite) *If $\mathcal{I}$ is finite wrt $\mathcal{S}$ and the number of transitions whose effective conditions hold in each state of $\mathcal{R}_{\mathcal{S},\leq(n-1)}$ is finite, $\mathcal{R}_{\mathcal{S},\leq n}$ is finite wrt $\mathcal{S}$.*

Proof By mathematical induction on n. □

If the number of processes involved in the (incorrect) ticketing lock is finite, say N processes, then the number of transitions whose effective conditions hold in each state

is finite. Since there are totally $4 * N$ (instances of) transitions, the number of transitions whose effective conditions hold in each state is less than or equal to $4 * N$. $\mathcal{I}/=_{\mathcal{S}_{\mathrm{Ticket}}}$ consists of one element. Therefore, when the number of processes involved in the (incorrect) ticketing lock is finite, $\mathcal{R}_{\mathcal{S}_{\mathrm{Ticket}},\leq n}$ is finite wrt $\mathcal{S}_{\mathrm{Ticket}}$, although $\mathcal{R}_{\mathcal{S}_{\mathrm{Ticket}}}$ is infinite wrt $\mathcal{S}_{\mathrm{Ticket}}$.

If $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$ does not hold, then there must exist a reachable state $\upsilon \in \mathcal{R}_{\mathcal{S}}$ such that $\neg p(\upsilon)$, and there must exit an execution fragment to υ wrt $\mathcal{S}$ from Prop. 1.

Definition 6 (Counterexamples) *Any execution fragment to $\upsilon \in \mathcal{R}_{\mathcal{S}}$ such that $\neg p(\upsilon)$ is called* a counterexample *for an invariant $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$. Let $\mathcal{CX}_{\mathcal{S},p}$ be the set of all counterexamples for $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$.* □

Any counterexample $cx \in \mathcal{CX}_{\mathcal{S},p}$ such that $\neg(\exists cx' : \mathcal{CX}_{\mathcal{S},p}.\, (\mathrm{depth}(cx') < \mathrm{depth}(cx)))$ is called a *shortest counterexample* for $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$.

3. Specification of OTSs

OTSs are defined so that they can be straightforwardly written as behavioral specifications in CafeOBJ. But, OTSs can also be written in Maude [5]. In this section, we describe how to write $\mathcal{S}$ in Maude. We suppose that there exist the functional module FM_k and the sort V_k corresponding to each data type D_k.

$\mathcal{O}$ and $\mathcal{T}$ are represented by sorts, say `OValue` and `TRule`, respectively. Therefore, each observer is represented by a term whose sort is `OValue`, and each transition is represented by a term whose sort is `TRule`. Υ is represented by a sort, say `Sys`. Each state is represented by a multiset of terms whose sorts are either `OValue` or `TRule`. `OValue` and `TRule` are then declared as subsorts of `Sys`:

```
subsorts OValue TRule < Sys .
```

Constant `none` represents the empty multiset, i.e. the empty state, and the juxtaposition operator is the data constructor of non-empty multisets, i.e. non-empty states. They are declared as follows:

```
op none : -> Sys .
op __ : Sys Sys -> Sys
                    [assoc comm id: none] .
```

The juxtaposition operator is given associativity, commutativity, and constant `none` as its identity.

Each observer $o_{x_1:D_{o1},\ldots,x_m:D_{om}} : \Upsilon \rightarrow D_o$ is represented by an operator declared as this:

```
op o[_, ..., _] :_ : V_o1 ... V_om V_o -> OValue .
```

The term `o[`a_1`,...,`a_m`] : a` appearing in a multiset that corresponds to a state υ represents the observer $o_{a_1,\ldots,a_m}$ such that $o_{a_1,\ldots,a_m}(\upsilon) = a$.

Each transition $t_{y_1:D_{t1},\ldots,y_n:D_{tn}} : \Upsilon \rightarrow \Upsilon$ is represented by an operator declared as this:

```
op t : V_j1 ... V_jn -> TRule .
```

The term `t(`b_1`,...,`b_n`)` represents the transition $t_{b_1,\ldots,b_n}$.

Transitions are defined in rewriting rules. The following is the typical rewriting rule for defining transitions:

```
crl [rule-t] :
  t(B_1,...,B_n) (o[A_1,...,A_m] : A) ...
  => t(B_1,...,B_n) (o[A_1,...,A_m] : A') ...
  if c-t(B_1,...,B_n) .
```

`crl` is the keyword to declare conditional rewriting rules, while `rl` is the keyword to declare non-conditional rewriting rules. The condition is written after the keyword `if`. `rule-t` is the label given to the rewriting rule. The left-hand side `t(`B_1`,...,`B_n`) (o[`A_1`,...,`A_m`] : A) ...` consists of one term of `TRule` and multiple terms of `OValue`, where each $A_{[k]}$ (or B_k) is a term of sort $V_{[ok]}$ (or V_{tk}). The right-hand side `t(`B_1`,...,`B_n`) (o[`A_1`,...,`A_m`] : A') ...` is equal to the left-hand side except that the values of the observers may be different in the right- and left-hand sides such that A' may be different from A. The operator `c-t` represents the effective condition of the transition represented by the operator `t`.

Basically the terms representing all (possible instances of) observers should appear in a rewriting rule for defining a transition. But, if an observer is never involved in the transition, then the term representing the observer can be omitted from the rewriting rule. If `c-t(`B_1`,...,`B_n`)` includes B_k `==` `d` as its conjunct, then `d` can be replaced with B_k in the left-hand side (and also the right-hand side if any), and B_k `==` `d` can be removed from `c-t(`B_1`,...,`B_n`)`. If any B_k appearing in `t(`B_1`,...,`B_n`)` also appears anywhere else in the left-hand side, then B_k and the corresponding argument may be removed from `t(`B_1`,...,`B_n`)` and the operator declaration of t, respectively.

An initial state of $\mathcal{S}$ can be represented by a term like this:

```
t(b_1,...,b_n) ... (o[a_1,...,a_m] : a) ...
```

The term representing an initial state of $\mathcal{S}$ consists of terms of `TRule` and terms of `OValue`.

Example 2 (Maude specification of $\mathcal{S}_{\mathrm{Ticket}}$) We suppose that there exist the functional modules `PID`, `NAT` and `LABEL`, and the sorts `Pid`, `Nat` and `Label` corresponding to the data types Pid, Nat and Label. $\mathcal{S}_{\mathrm{Ticket}}$ is specified in the module `TICKET` that imports the three functional modules. The signature of `TICKET` is as follows:

```
sorts TRule OValue Sys .
subsorts TRule OValue < Sys .
op none : -> Sys .
op __   : Sys Sys -> Sys [assoc comm id: none] .
op pc[_] :_     : Pid Label -> OValue .
op ticket[_] :_ : Pid Nat -> OValue .
op turn :_      : Nat -> OValue .
op tvm :_       : Nat -> OValue .
op get   : -> TRule .  op inc   : -> TRule .
op enter : -> TRule .  op leave : -> TRule .
```

There are four rewriting rules in TICKET, which are as follows:

```
rl [get] : get (pc[I] : l1) (ticket[I] : T) (tvm : V)
  => get(pc[I] : l2) (ticket[I] : V) (tvm : V) .
rl [inc] : inc (pc[I] : l2) (ticket[I] : T) (tvm : V)
  => inc (pc[I] : l3) (ticket[I] : T) (tvm : (T + 1)) .
rl [enter] :
  enter (pc[I] : l3) (ticket[I] : V) (turn : V)
  => enter (pc[I] : cs) (ticket[I] : V) (turn : V) .
rl [leave] : leave (pc[I] : cs) (turn : V)
  => leave (pc[I] : l1) (turn : (V + 1)) .
```

When there are two processes denoted by constants p1 and p2 of sort Pid, the initial state is represented by the following term:

```
get inc enter leave (pc[p1] : l1) (pc[p2] : l1)
(ticket[p1] : 0) (ticket[p2] : 0) (turn : 0) (tvm : 0)
```

The term will be referenced by the constant init of sort Sys. □

4. Falsification of OTSs

Maude is used to falsify $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$, i.e. to find a counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$. To this end, the Maude model checker can be used. $\mathcal{R}_{\mathcal{S}}$ should be finite wrt $\mathcal{S}$ so that $\mathcal{S}$ can be model checked by the Maude model checker. The proposed solution is to search $\mathcal{R}_{\mathcal{S},\leq n}$ for a counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$. If $\mathcal{R}_{\mathcal{S},\leq n}$ is finite wrt $\mathcal{S}$, this search can be completed within a finite time. A sufficient condition that $\mathcal{R}_{\mathcal{S},\leq n}$ is finite wrt $\mathcal{S}$ that $\mathcal{I}$ is finite wrt $\mathcal{S}$ and the number of transitions whose effective conditions hold in each state of $\mathcal{R}_{\mathcal{S},\leq(n-1)}$ is finite from Theorem 1. Since Maude is not equipped with any facilities that can be used to search only $\mathcal{R}_{\mathcal{S},\leq n}$ for a counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$, however, we need to make a small modification to $\mathcal{S}$.

Definition 7 (Bounded OTSs) *One observer* $\mathrm{steps} : \Upsilon \rightarrow \mathrm{Nat}$ *is added to* $\mathcal{S}$. *The initial value returned by* steps *is* 0, *and the inequality* $\mathrm{steps}(\upsilon) < n$ *is added to the effective condition of each transition. The value returned by* steps *is incremented whenever each transition is applied in a state where the effective condition holds. The OTS obtained by modifying* $\mathcal{S}$ *in this way is called* the (n-)bounded OTS $\mathcal{S}$ *and denoted* $\mathcal{S}^{\leq n}$. □

We present a theorem that guarantees that the search of $\mathcal{R}_{\mathcal{S}^{\leq n}}$ for a counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}^{\leq n}}.\, p(\upsilon)$ coincides with the search of $\mathcal{R}_{\mathcal{S},\leq n}$ for a counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$ if the observer steps is not used in p. Before that, a lemma is given to prove the theorem.

Lemma 1 (Execution fragments wrt $\mathcal{S}^{\leq n}$ and $\mathcal{S}$) *(1) Any execution fragment to* υ *wrt* $\mathcal{S}^{\leq n}$ *is also an execution fragment to* υ *wrt* $\mathcal{S}$, *and (2) for any execution fragment* $\upsilon_0, \ldots, \upsilon_m$ *wrt* $\mathcal{S}$ *such that* $m \leq n$, *there exists an execution fragment* $\upsilon'_0, \ldots, \upsilon'_m$ *wrt* $\mathcal{S}^{\leq n}$ *such that* $\upsilon'_k =_{\mathcal{S}} \upsilon_k$ *for* $k = 0, \ldots, m$.

Proof From the definition of $\mathcal{S}^{\leq n}$. □

Theorem 2 (Coincidence of counterexamples) *If the observer* steps *is not used in a state predicate* $p : \Upsilon \rightarrow \mathrm{Bool}$, *then (1) any counterexample for* $\forall \upsilon : \mathcal{R}_{\mathcal{S}^{\leq n}}.\, p(\upsilon)$ *is also a counterexample for* $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$, *and (2) for any counterexample* $\upsilon_0, \ldots, \upsilon_m$ *for* $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$ *such that* $m \leq n$, *there exists a counterexample* $\upsilon'_0, \ldots, \upsilon'_m$ *for* $\forall \upsilon : \mathcal{R}_{\mathcal{S}^{\leq n}}.\, p(\upsilon)$ *such that* $\upsilon'_k =_{\mathcal{S}} \upsilon_k$ *for* $k = 0, \ldots, m$.

Proof (1) We suppose that $cx \in \mathcal{CX}_{\mathcal{S}^{\leq n},p}$. Since p does not use steps at all, $cx \in \mathcal{CX}_{\mathcal{S},p}$ from Lemma 1. (2) We suppose that $\upsilon_0, \ldots, \upsilon_m \in \mathcal{CX}_{\mathcal{S},p}$ such that $m \leq n$. There must exist an execution fragment $\upsilon'_0, \ldots, \upsilon'_m$ wrt $\mathcal{S}^{\leq n}$ such that $\upsilon'_k =_{\mathcal{S}} \upsilon_k$ for $k = 0, \ldots, m$ from Lemma 1. Hence, $\upsilon'_0, \ldots, \upsilon'_m \in \mathcal{CX}_{\mathcal{S}^{\leq n},p}$ because $\upsilon'_m =_{\mathcal{S}} \upsilon_m$. □

Therefore, the Maude model checker can be used to search $\mathcal{R}_{\mathcal{S}^{\leq n}}$, instead of $\mathcal{R}_{\mathcal{S},\leq n}$, for a counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}}.\, p(\upsilon)$.

Example 3 (Falsification of $\mathcal{S}_{\mathrm{Ticket}}$) $\mathcal{S}_{\mathrm{Ticket}}$ is first modified to obtain $\mathcal{S}^{\leq n}_{\mathrm{Ticket}}$. The rewriting rules representing the transitions of $\mathcal{S}^{\leq n}_{\mathrm{Ticket}}$ are as follows:

```
crl [get] : get (pc[I] : l1) (ticket[I] : T) (tvm : V)
            (steps : C)
  => get(pc[I] : l2) (ticket[I] : V) (tvm : V)
     (steps : (C + 1)) if (C < bound) .
crl [inc] : inc (pc[I] : l2) (ticket[I] : T) (tvm : V)
            (steps : C)
  => inc (pc[I] : l3) (ticket[I] : T) (tvm : (T + 1))
     (steps : (C + 1)) if (C < bound) .
crl [enter] : enter (pc[I] : l3) (ticket[I] : V)
              (turn : V) (steps : C)
  => enter (pc[I] : cs) (ticket[I] : V) (turn : V)
     (steps : (C + 1)) if (C < bound) .
crl [leave] : leave (pc[I] : cs) (turn : V) (steps : C)
  => leave (pc[I] : l1) (turn : (V + 1))
     (steps : (C + 1)) if (C < bound) .
```

where constant bound corresponds to n.

The Maude model checker can be used to search $\mathcal{R}_{\mathcal{S}^{\leq n}_{\mathrm{Ticket}}}$ for a counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}_{\mathrm{Ticket}}}.\, \mathrm{MX}(\upsilon)$, and so can command search. In this paper, we use command search. One way to use command search is as follows:

search [1] *start* =>* *pattern* such that *condition* .

Command search performs a breadth-first search to find one state that matches *pattern* and that can be reached from *start* by applying zero or more rewriting rules.

To search $\mathcal{R}_{\mathcal{S}^{\leq n}_{\mathrm{Ticket}}}$ for a counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}_{\mathrm{Ticket}}}.\, \mathrm{MX}(\upsilon)$, all we have to do is to feed the following line to the Maude system:

```
search [1] init =>* (pc[p1] : cs) (pc[p2] : cs) S .
```

When bound is 6, command search instantaneously finds a state υ in which $\mathrm{MX}(\upsilon)$ does not hold. Command show path can be used to show the path to the state, which is a shortest counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}_{\mathrm{Ticket}}}.\, \mathrm{ME}(\upsilon)$. □

5. One More Case Study: NSPK Protocol

The NSPK authentication protocol [8] can be described as the three message exchanges:

Msg 1 $p \longrightarrow q : \mathcal{E}_q(n_p, p)$
Msg 2 $q \longrightarrow p : \mathcal{E}_p(n_p, n_q)$
Msg 3 $p \longrightarrow q : \mathcal{E}_q(n_q)$

Each principal is given a pair of keys: public and private keys. $\mathcal{E}_p(m)$ is the message m encrypted with the principal p's public key. n_p is a nonce (a random number) generated by principal p.

5.1. Modeling NSPK

One of the desired invariant properties that the protocol should have is (*Nonce*) *Secrecy Property* that any nonces cannot be leaked. The protocol is modeled as an OTS $\mathcal{S}_{\rm NSPK}$ by taking into account the intruder so as to check if the protocol has Secrecy Property. The data types used in $\mathcal{S}_{\rm NSPK}$ are: (1) Bool for truth values, (2) Prin for principals; intr denoting the intruder, (3) Rand for random numbers; seed denoting a random number available initially; next(r) denoting a random number that has never been generated so far, (4) Nonce for nonces; n(p, q, r) denoting the nonce (generated by principal p for principal q) whose uniqueness is guaranteed by random number r, (5) Cipher for ciphertexts; enc1(p, n, q) denoting $\mathcal{E}_q(n, q)$; enc2($p, n1, n2$) denoting $\mathcal{E}_q(n1, n2)$; enc3(p, n) denoting $\mathcal{E}_q(n)$, (6) SetNonce for sets of nonces; empty denoting the empty set; n , s denoting $\{n\} \cup s$; $s1$, $s2$ denoting $s1 \cup s2$, and (7) Network for multisets of ciphertexts; empty denoting the empty multiset; e , m denoting $\{\!|\, e \,|\!\} \uplus m$; $m1$, $m2$ denoting $m1 \uplus m2$.

The protocol is modeled as an OTS $\mathcal{S}_{\rm NSPK}$ that is $\langle \mathcal{O}_{\rm NSPK}, \mathcal{I}_{\rm NSPK}, \mathcal{T}_{\rm NSPK} \rangle$ such that

$$
\begin{aligned}
\mathcal{O}_{\rm NSPK} \triangleq \{ & \ {\rm rand} : \Upsilon \to {\rm Rand}, {\rm nw} : \Upsilon \to {\rm Network}, \\
& \ {\rm nonces} : \Upsilon \to {\rm SetNonce}\} \\
\mathcal{I}_{\rm NSPK} \triangleq \{ & \ \upsilon_{\rm init} \in \Upsilon \mid {\rm rand}(\upsilon_{\rm init}) = {\rm seed} \wedge \\
& \ {\rm nw}(\upsilon_{\rm init}) = {\rm empty} \wedge {\rm nonces}(\upsilon_{\rm init}) = {\rm empty}\} \\
\mathcal{T}_{\rm NSPK} \triangleq \{ & {\rm send1}_{p:{\rm Prin},q:{\rm Prin}} : \Upsilon \to \Upsilon, \\
& {\rm send2}_{p:{\rm Prin},q:{\rm Prin},n:{\rm Nonce},nw:{\rm Network}} : \Upsilon \to \Upsilon, \\
& {\rm send3}_{p:{\rm Prin},q:{\rm Prin},n1,n2:{\rm Nonce},nw:{\rm Network}} : \Upsilon \to \Upsilon, \\
& {\rm fake1}_{p:{\rm Prin},q:{\rm Prin},n:{\rm Nonce},ns:{\rm SetNonce}} : \Upsilon \to \Upsilon, \\
& {\rm fake2}_{p:{\rm Prin},n1,n2:{\rm Nonce},ns:{\rm SetNonce}} : \Upsilon \to \Upsilon, \\
& {\rm fake3}_{p:{\rm Prin},n:{\rm Nonce},ns:{\rm SetNonce}} : \Upsilon \to \Upsilon\}
\end{aligned}
$$

Given a state $\upsilon \in \Upsilon$, rand returns a random number available in υ, nw returns a multiset of ciphertexts (denoting the network) that have been sent up to υ, and nonces returns a set of nonces that have been gleaned by the intruder up to υ. The first three transitions model sending messages exactly following the protocol, while the last three transitions model the intruder's faking messages based on the gleaned nonces. The transitions are defined as follows:

• ${\rm send1}_{p,q} : {\rm send1}_{p,q}(\upsilon) \triangleq \upsilon'$ such that

${\rm rand}(\upsilon') \triangleq {\rm next}({\rm rand}(\upsilon))$,
${\rm nw}(\upsilon') \triangleq {\rm enc1}(q, n(p, q, {\rm rand}(\upsilon)), p)$, ${\rm nw}(\upsilon)$, and
${\rm nonces}(\upsilon') \triangleq$ **if** $q = {\rm intr}$ **then** $n(p, q, {\rm rand}(\upsilon))$, ${\rm nonces}(\upsilon)$ **else** ${\rm nonces}(\upsilon)$.

• ${\rm send2}_{p,q,n,nw} : c_{{\rm send2}_{p,q,n,nw}}(\upsilon) \triangleq ({\rm nw}(\upsilon) = {\rm enc1}(p, n, q)$, $nw)$. If $c_{{\rm send2}_{p,q,n,nw}}(\upsilon)$, then ${\rm send2}_{p,q,n,nw}(\upsilon) \triangleq \upsilon'$ such that

${\rm rand}(\upsilon') \triangleq {\rm next}({\rm rand}(\upsilon))$,
${\rm nw}(\upsilon') \triangleq {\rm enc2}(q, n, n(p, q, {\rm rand}(\upsilon)))$, ${\rm nw}(\upsilon)$, and
${\rm nonces}(\upsilon') \triangleq$ **if** $q = {\rm intr}$ **then** n , $n(p, q, {\rm rand}(\upsilon))$, ${\rm nonces}(\upsilon)$ **else** ${\rm nonces}(\upsilon)$.

• ${\rm send3}_{p,q,n1,n2,nw} : c_{{\rm send2}_{p,q,n1,n2,nw}}(\upsilon) \triangleq ({\rm nw}(\upsilon) = {\rm enc2}(p$, $n1, n2)$, ${\rm enc1}(q, n1, p)$, $nw)$. If $c_{{\rm send3}_{p,q,n1,n2,nw}}(\upsilon)$, then ${\rm send3}_{p,q,n1,n2,nw}(\upsilon) \triangleq \upsilon'$ such that

${\rm rand}(\upsilon') \triangleq {\rm rand}(\upsilon)$,
${\rm nw}(\upsilon') \triangleq {\rm enc3}(q, n2)$, ${\rm nw}(\upsilon)$, and
${\rm nonces}(\upsilon') \triangleq$ **if** $q = {\rm intr}$ **then** $n2$, ${\rm nonces}(\upsilon)$ **else** ${\rm nonces}(\upsilon)$.

• ${\rm fake1}_{p,q,n,ns} : c_{{\rm fake1}_{p,q,n,ns}}(\upsilon) \triangleq ({\rm nonces}(\upsilon) = n$, $ns)$. If $c_{{\rm fake1}_{p,q,n,ns}}(\upsilon)$, then ${\rm fake1}_{p,q,n,ns}(\upsilon) \triangleq \upsilon'$ such that

${\rm rand}(\upsilon') \triangleq {\rm rand}(\upsilon)$, ${\rm nw}(\upsilon') \triangleq {\rm enc1}(q, n, p)$, ${\rm nw}(\upsilon)$, and
${\rm nonces}(\upsilon') \triangleq {\rm nonces}(\upsilon)$

• ${\rm fake2}_{p,n1,n2,ns} : c_{{\rm fake2}_{p,n1,n2,ns}}(\upsilon) \triangleq ({\rm nonces}(\upsilon) = n1$, $n2$, $ns)$. If $c_{{\rm fake2}_{p,n1,n2,ns}}(\upsilon)$, then ${\rm fake2}_{p,n1,n2,ns}(\upsilon) \triangleq \upsilon'$ such that

${\rm rand}(\upsilon') \triangleq {\rm rand}(\upsilon)$, ${\rm nw}(\upsilon') \triangleq {\rm enc2}(p, n1, n2)$, ${\rm nw}(\upsilon)$, and
${\rm nonces}(\upsilon') \triangleq {\rm nonces}(\upsilon)$.

• ${\rm fake3}_{p,n,ns} : c_{{\rm fake3}_{p,n,ns}}(\upsilon) \triangleq ({\rm nonces}(\upsilon) = n$, $ns)$. If $c_{{\rm fake3}_{p,n,ns}}(\upsilon)$, then ${\rm fake3}_{p,n,ns}(\upsilon) \triangleq \upsilon'$ such that

${\rm rand}(\upsilon') \triangleq {\rm rand}(\upsilon)$, ${\rm nw}(\upsilon') \triangleq {\rm enc3}(p, n)$, ${\rm nw}(\upsilon)$, and
${\rm nonces}(\upsilon') \triangleq {\rm nonces}(\upsilon)$.

5.2. Specification

$\mathcal{S}_{\rm NSPK}$ is first modified to obtain $\mathcal{S}^{\leq n}_{\rm NSPK}$. We suppose that there exist the functional modules BOOL, NAT, PRIN, RAND, NONCE, CIPHER, SETNONCE and NETWORK, and the sorts Bool, Nat, Prin, Rand, Nonce, Cipher, SetNonce and Network corresponding to the data types used in $\mathcal{S}^{\leq n}_{\rm NSPK}$. $\mathcal{S}^{\leq n}_{\rm NSPK}$ is specified in the module NSPK that imports the functional modules. The signature of NSPK is as follows:

```
sorts TRule OValue Sys .
subsorts TRule OValue < Sys .
op none : -> Sys .
op __   : Sys Sys -> Sys [assoc comm id: none] .
op rand :_   : Rand -> OValue .
op nw :_     : Network -> OValue .
op nonces :_ : SetNonce -> OValue .
op steps :_  : Nat -> OValue .
op send1 : Prin Prin -> TRule .
op send2 : -> TRule .       op send3 : -> TRule .
op fake1 : Prin Prin -> TRule .
op fake2 : Prin -> TRule .  op fake3 : Prin -> TRule .
op bound : -> Nat .
```

There are six rewriting rules in NSPK, which are as follows:

```
crl [send1] : send1(P1,P2) (rand : R) (nw : NW)
    (nonces : Ns) (steps : X)
  => send1(P1,P2) (rand : next(R))
     (nw : (enc1(P2,n(P1,P2,R),P1) , NW))
     (nonces : (if P2 == intr then n(P1,P2,R) , Ns
                              else Ns fi))
     (steps : (X + 1)) if X < bound .
crl [send2] : send2 (rand : R)
    (nw : (enc1(P1,N,P2) , NW)) (nonces : Ns)
    (steps : X)
  => send2 (rand : next(R))
     (nw : (enc2(P2,N,n(P1,P2,R)) ,
            enc1(P1,N,P2) , NW))
     (nonces : (if P2 == intr then N , n(P1,P2,R) , Ns
                              else Ns fi))
     (steps : (X + 1)) if X < bound .
crl [send3] : send3 (rand : R)
    (nw : (enc2(P1,N1,N2), enc1(P2,N1,P1) , NW))
    (nonces : Ns) (steps : X)
  => send3 (rand : R)
     (nw : (enc3(P2,N2) , enc2(P1,N1,N2),
            enc1(P2,N1,P1) , NW))
     (nonces : (if P2 == intr then N2 , Ns
                              else Ns fi))
     (steps : (X + 1)) if X < bound .
crl [fake1] : fake1(P1,P2) (nonces : (N , Ns))
    (nw : NW) (steps : X)
  => fake1(P1,P2) (nonces : (N , Ns))
     (nw : (enc1(P2,N,P1) , NW))
     (steps : (X + 1)) if X < bound .
crl [fake2] : fake2(P1) (nonces : (N1 , N2 , Ns))
    (nw : NW) (steps : X)
  => fake2(P1) (nonces : (N1 , N2 , Ns))
     (nw : (enc2(P1,N1,N2) , NW))
     (steps : (X + 1)) if X < bound .
crl [fake3] : fake3(P1) (nonces : (N , Ns)) (nw : NW)
    (steps : X)
  => fake3(P1) (nonces : (N , Ns))
     (nw : (enc3(P1,N) , NW))
     (steps : (X + 1)) if X < bound .
```

When there are three principals (including the intruder) denoted by constants p1, p2 and intr of sort Prin, the initial state is represented by the following term:

```
send1(p1,p2) send1(p1,intr) send1(p2,p1) send1(p2,intr)
send1(intr,p1) send1(intr,p2) send2 send3 fake1(p1,p2)
fake1(p1,intr) fake1(p2,p1) fake1(p2,intr)
fake1(intr,p1) fake1(intr,p2) fake2(p1) fake2(p2)
fake2(intr) fake3(p1) fake3(p2) fake3(intr)
(rand : seed) (nw : empty) (nonces : empty) (steps : 0)
```

The term will be referenced by the constant init of sort Sys.

5.3. Falsification

Secrecy Property can be expressed as $\forall \upsilon \in \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}.\ \mathrm{SP}(\upsilon)$, where $\mathrm{SP}(\upsilon) \triangleq \forall n : \mathrm{Nonce}\,(n \in \mathrm{nonces}(\upsilon) \Rightarrow (\mathrm{p1}(n) = \mathrm{intr} \vee \mathrm{p2}(n) = \mathrm{intr}))$, $\mathrm{p1}(\mathrm{n}(p,q,r)) \triangleq p$ and $\mathrm{p2}(\mathrm{n}(p,q,r)) \triangleq q$.

To search $\mathcal{R}_{\mathcal{S}^{\leq n}_{\mathrm{NSPK}}}$ for a counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}_{\mathrm{Ticket}}}.\mathrm{SP}(\upsilon)$, all we have to do is to feed the following line to the Maude system:

```
search [1] init =>* (nonces : (N , Ns)) S
    such that not(p1(N) == intr or p2(N) == intr) .
```

When bound is 4, command search finds a state υ in which $\mathrm{SP}(\upsilon)$ does not hold. The time spent is about 300ms. Command show path can be used to show the path to the state, which is a shortest counterexample for $\forall \upsilon : \mathcal{R}_{\mathcal{S}_{\mathrm{NSPK}}}.\mathrm{SP}(\upsilon)$ and coincides with the Lowe's attack [7].

6. Conclusion

We have described a way to search a bounded reachable state space of an OTS for a counterexample that the OTS satisfies an invariant property. We have also reported on the two case studies to demonstrate the proposed solution.

We have been developing a translator [6], which takes a CafeOBJ specification of an OTS and generates a Maude specification of the OTS. The proposed solution in this paper will be incorporated into the translator.

References

[1] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *5th TACAS*, LNCS 1579, pages 193–207. Springer, 1999.

[2] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. Maude: Spesification and programming language in rewriting logic. *TCS*, 285(2):187–243, 2002.

[3] R. Diaconescu and K. Futatsugi. *CafeOBJ Report*, volume 6 of *AMAST Series in Computing*. World Scientific, 1998.

[4] S. Eker, J. Meseguer, and A. Sridharanarayanan. The Maude LTL model checker. In *4th WRLA*, ENTCS 71, pages 143–168. Elsevier, 2002.

[5] W. Kong, K. Ogata, and K. Futatsugi. Model-checking observational transition system with Maude. In *20th ITC-CSCC*, pages 5–6, 2005.

[6] W. Kong, K. Ogata, T. Seino, and K. Futatsugi. Lightweight integration of theorem proving and model checking for system verification. In *12th APSEC*, pages 59–66. IEEE CS Press, 2005.

[7] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *2nd TACAS*, LNCS 1055, pages 147–166. Springer, 1996.

[8] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *CACM*, 21(12):993–999, 1978.

[9] K. Ogata and K. Futatsugi. Formal analysis of the iKP electronic payment protocols. In *1st ISSS*, LNCS 2609, pages 441–460. Springer, 2003.

[10] K. Ogata and K. Futatsugi. Proof scores in the OTS/CafeOBJ method. In *6th FMOODS*, LNCS 2884, pages 170–184. Springer, 2003.

[11] K. Ogata and K. Futatsugi. Equational approach to formal analysis of TLS. In *25th ICDCS*, pages 795–804. IEEE CS Press, 2005.

Implementation of CafeOBJ Specifications to Java Code

Samira Sadaoui Sudhanshu Singh
Department of Computer Science, University of Regina
Regina, SK S4S 0A2, Canada
E-mail: sadaouis@cs.uregina.ca

Abstract

Automatic code generation is a technique that ensures the correctness and consistency of programs with respect to their specifications. In addition, it provides a means for software prototyping. In this paper, we present the syntactic and semantic transformation algorithms, and their supporting tool, used to generate object-oriented code from (un)parameterized algebraic specifications. The proposed algorithms are based on the formal specification interpretation strategy. The two languages CafeOBJ and Java are chosen because of their good support for component parametrization and instantiation. We also illustrate the application of our code generation system on a generic sorting specification.

Keywords: algebraic specifications, syntactic transformation, semantic transformation, specification interpretation strategy.

1. Introduction

Automatic code generation produces code directly from specifications and hence removes ambiguities between specifications and code. In particular, generating code from formal specifications has many benefits [1, 6] including: shorter and less error prone software life cycle as human effort is required only during the specification stage; consistency in the architectural framework; high quality code as code is produced from specifications that have been proved correct, complete and consistent; consistency in the generated code prevents discrepancy in coding styles; the specification, viewed as an abstraction of code, is much easier to maintain which automatically leads to a maintained code. Code generation from formal specifications can be segregated into two strategies: (1) term rewriting strategy [4, 9] where specifications are transformed to code through a term rewriting system, and (2) specification interpretation strategy [3, 8, 1, 9, 7, 10, 13] where specifications are transformed to code through a set of transformation rules.

Our framework is based on the specification interpretation strategy to obtain object-oriented code from algebraic specifications. The implementation of this strategy is done using a tree structure [14]: operations and axioms of a specification are represented as tree manipulation functions; constructor invocation creates a new node while operation execution creates leaves of the tree. The basic structure of a code generation system using this strategy is given in Figure 1. It comprises of a syntax analyzer and a transformation engine [12]. The syntax analyzer accepts a specification, analyzes it and then generates the tree structure. The transformation engine, which contains a set of rules, takes the tree structure and applies the rules to generate the code. The obvious advantage of this strategy is the use of an incremental approach with an executable code existing at each stage. Existing work done on the interpreting strategy is as follows: in [3] a small subset of CafeOBJ constructs [11] is translated into concret Java constructs; in [8], axiomatic specifications are converted to C code; in [1] PL/I and Pascal progarms are generated from abstract specifications; [9] uses APTS to obtain C code from SCR specifications; [7] develops a rapid prototyping system to transform abstract data types to Pascal code; [10] presents a tool to translate SDL specifications to C++ code.

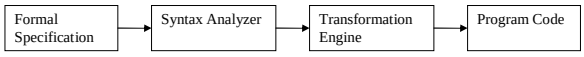


Figure 1. Specification Interpretation Strategy

Due to the object-oriented nature of most real-world applications, obtaining object-oriented code from specifications is quite useful. As parametrization and instantiation mechanisms are not taken into consideration in the existing work, we decide to incorporate them in our code generation framework. The formal specification language CafeOBJ is chosen for its support of parameterized programming and theorem proving. Java version 1.5 is used as a target language since it supports genericity and instantiation. Our code generation system is based on two conversion phases:

- Syntactic transformation which converts the syntactic part of CafeOBJ specifications into a skeletal Java code. The previous work [3] concentrates only on sorts, inheritance and operations whereas our algorithm incorporates almost all the CafeOBJ constructs as provided in figure 1. The major drawback of [3] is the use of data types for each sort and inherited module while our algorithm uses the specification interpreting strategy.
- Semantic transformation which converts the semantic part of CafeOBJ specifications. The CafeOBJ equations are used to provide the first version of the Java methods that have been generated in the first phase. This transformation is not implemented in [3]. We also note that the developer has to complete and optimize the source Java code.

2. Syntactic Transformation

2.1. CafeOBJ Constructs to Java Constructs

We now show how a CafeOBJ construct is converted into its corresponding Java construct (cf. figure 1):

- **Module Declaration and Inheritance.** *ConvSyntacticCafe2Java* algorithm performs the syntactic transformation. It invokes the function *ConvModule* which takes a module and outputs a Java class. Inheritance is also taken into account. Also *ConvModule* translates user-defined sorts and records into their corresponding classes.
- **Sort and Variable Declarations.** Sorts, sub-sorts and modules are translated into types and variables in Java using resp. the three functions *ConvSimpleSort*, *ConvSubSort* and *ConvType*. As sorts are classified as visible or hidden, they are transformed to public or private Java variables. The variables and objects for Java data types are also specified in the Main class.
- **Operation Declaration and Method Invocation.** Operation, behavioral and predicate declarations are translated into Java methods. This is done through the function *OpDec* which also segregates operations into constructors and non constructors.
- **Record Declaration.** Plain and inherited records are translated into Java classes through the function *ConvModule*. Slotnames are converted to variable names, sorts to Java data types.
- **Error Handling.** This mechanism is transformed to Java exception handlling. A search is performed for operations and equations that incorporate the error handling to extract the Java methods. The code of these methods is placed inside the Java exception handling using the try and catch block with exception being caught by an object of Java class exception.
- **Parameterization.** Parameterized modules are converted into corresponding classes using Java generics. This is done with the function *ConvParameterization*, an extension of *ConvSyntacticCafe2Java* to handle parameterized sorts and modules.
- **View.** The instantiation mechanism is translated using the function *ConvView*. A Java class is declared for the instantiated module and incorporated into the generic mechanism.

2.2. Example

We apply *ConvSyntacticCafe2Java* on a constrained generic sorting specification called GLIST defined on the left side of Figure 3. GLIST has four operations: “nil” creates an empty list, “cons” creates a new list with element specified, “insert” adds an element to the list, “sort” sorts the list. GLIST uses a formal parameter ELEMENT defined with the operation “le” which compares two elements. “le” is defined with a total order given as formal equations. The resulting Java classes and methods are presented on the right side of Figure 3.

3. Semantic Transformation

3.1. CafeOBJ Equations to Method Implementation

We now show how the CafeOBJ equations are used to provide the first version of the Java methods:

- **Generic Method Declaration.** *ConvSemanticCafe2Java* algorithm performs the semantic conversion. First, it invokes *ConvGeneric* which declares the two Java generic methods “checkempty()” and “add()” and also boolean variable “flag” and integer variable “counter”. “checkempty()” and “add()” perform the tasks as specified by their names. “flag” and “counter” determine the state of a module.
- **Equation Declaration.** Through the function *ConvSets*, the equations of each non-constructor are stored in a subset EQ_i. *ConvEq* determines the number of decisions that are to be made inside a Java method. A nested if and else statements is provided which checks if the object invoking the method is empty or it has one element or more depending on the cardinality of EQ_i. The if and else conditions are executed depending on the value of “counter”. The non constructor Java methods are invoked on object of a module in the Main method.

CafeOBJ Constructs	Function	Java Constructs
module	*ConvModule*	class
sort	*ConvSimpleSort*	data type
sub-sort	*ConvSubSort*	data type
visible or hidden variable	*ConvType*	public or private variable
standard, mixfix, behavioural operation	*OpDec*	method
predicate	*OpDec*	method returning boolean
plain and inherited record	*ConvModule*	class
error handling	*ConvModule*	exception handling
parameterized module	*ConvParametrization*	class
view (instantiation)	*ConvView*	class

Figure 2. CafeOBJ Construct Conversion Table

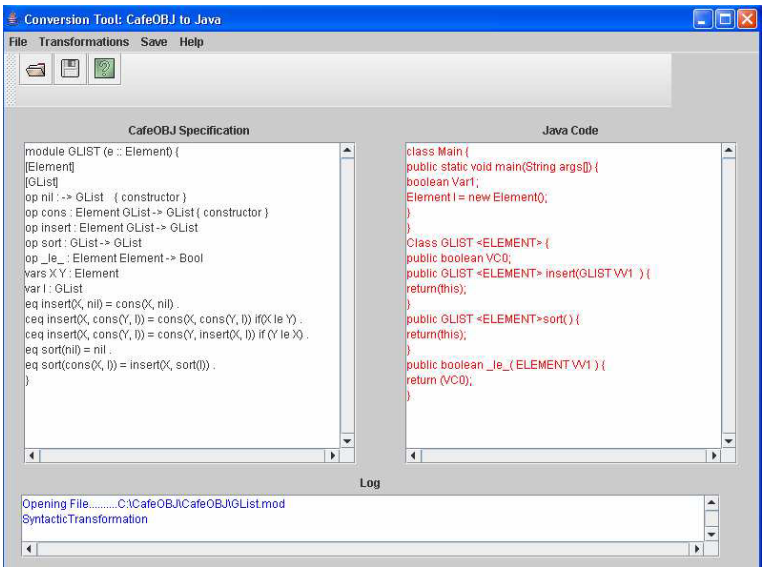


Figure 3. Syntactic Conversion of GLIST.

3.2. Example

We apply now *ConvSemanticCafe2Java* on GLIST.mod and GLIST.java of Figure 3. We also note that the formal constraints are lost during the conversion since programming languages do not support genericity with constraints.

4. Code Generation Tool

A conversion tool “Cafe2Java” is implemented in Java to demonstrate our claim. It performs the syntactic and semantic transformations. NETBEANS 4.1 is used as an environment tool for its support of Java version 1.5. We note that “Cafe2Java” has been successively experimented on several case studies. Figure 5 shows the block diagram of our tool. The graphical user interface of “Cafe2Java” is shown in figure 3. Input to the tool is a CafeOBJ module M1.mod entered through GUI. It is passed to the scanner which performs two functions: firstly, the syntactic correctness of the specification is determined by scanning, secondly the resulting tokens are segregated into valid and invalid ones, later being discarded. The valid tokens are then passed to the parser which organizes them into different categories like modules, sorts, operations, records, views, error handling, etc. These tokens, organized a tree structure, are then passed to the ”ConvSyntacticCafe2Java.java”. Based on a given path, a specific set of transformation rules are applied to obtain the corresponding code. The final code M1.java along with tokens are passed to the next analyser to parse M1.java to determine where the details of methods are to be provided. The resulting tokens are then passed to ”ConvSemanticCafe2Java.java” which applies rules on M1.java and M1.mod.

```
class ELEMENT{
     public ELEMENT VC1;
     public boolean VC2;
     public boolean le(ELEMENT VV1){
          return VC2;  } }
class GLIST <ELEMENT> {
    public boolean VC1;
    public ELEMENT VC2;
    public boolean flag = false;
    public int counter = 0;
    public boolean checkempty(){
      if(flag == false) return true; else return false; }
    public void add(ELEMENT D) {
      VC2 = D; flag = true; }
    public GLIST <ELEMENT> insert(GLIST VV1) {
     if(this.checkempty()== true && counter <1) {return(this); }
     else if(this.checkempty()== false && counter ==1) { flag = false; return(this); }
     else{  flag = false; return(this); } }
    public GLIST <ELEMENT> sort() {
     if(this.checkempty() == false && counter < 1) { return(this); }
     else{ flag = false; return(this); } }
    public boolean le(ELEMENT VV1) {
     if(this.checkempty() == false && counter < 1) { return(VC1); }
     else{ flag = false; return(VC1); } } }
public class Main{
  public static void main(String[] args) {
       boolean Var1 = false;
      GLIST <Integer> I = new GLIST <Integer>();
      ELEMENT I2 = new ELEMENT();
      I. add(new Integer(5));
      I.counter++; I.insert(I); I.sort(); }}
```

Figure 4. Semantic Conversion of GLIST.

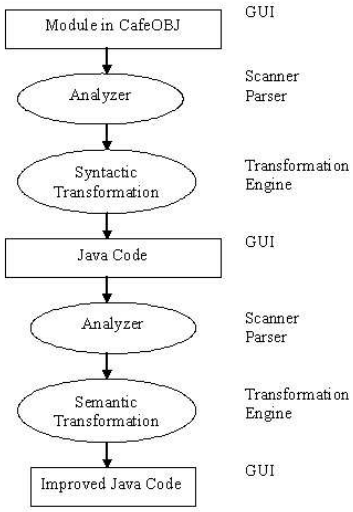


Figure 5. Code Generation Process in Cafe2Java.

5. Conclusion and Prospects

Our code generation system, based on the specification interpretation strategy along with compiler techniques, involves two transformation phases: syntactic and semantic. Parametrization and instantiation are incorporated in our system. Although the generated code is not complete, it provides an easy syntax and a good abstraction for describing complex processes and also for reusing library components. Some directions of our work are as follows: (1) include optimization techniques [2] in our transformation algorithms to generate a more efficient code, and (2) incorporate into our code generator an assistance tool for constructing incrementally CafeOBJ specifications.

References

[1] B. Belkhouche and J. E. Urban. Direct Implementation of Abstract Data Types from Abstract Specifications. In IEEE Transactions on Software Engineering, vol 12(5), pp 649–661, 1986.

[2] D. M. DeJean and G. W. Zobrist. A Definition Optimization Technique Used in a Code Translation Algorithm. In Communications of the ACM, Vol. 32, pp 94-105, NY, 1998.

[3] C. Doungsa-ard and T. Suwannasart. An Automatic Approach to Transform CafeOBJ Specifications to Java Template Code. In Software Engineering Research and Practice, CSREA Press, pp 171–176, 2003.

[4] A. P. Forcheri and M.T. Molfino. Specification-based Code Generation. In 23rd Annual Hawaii International Conference on System Sciences, Kailua-Kona, Hawaii,1990, pp 165–173.

[5] A. Geser, H. Hussmann and A. Muck. A Compiler for a Class of Term Rewriting Systems. In Conditional Term Rewriting Systems 1st International Workshop, vol 308, LNCS, Springer, July 8-10, 1987

[6] J. Herrington. Code Generation In Action. In Manning Publications Co., July 2003

[7] L. Jadoul, L. Duponcheel and W.V. Puymbroeck. An Algebraic Data Type Specification Language and its Rapid Prototyping Environment. In 11th international conference on software engineering, ACM Press, pp 74–84,1989.

[8] P. Jalote. Synthesizing Implementation of Abstract Data Types from Axiomatic Specifications. Software-Practice and Experience, vol 17(11), 1987, pp 847–858.

[9] E.I. Leonard. Program Synthesis from Formal Requirements Specifications Using APTS. Higher-Order and Symbolic Computations, vol 16, Kluwer Academic Publishers, pp 63–92, 2003.

[10] K. Miyake,Y. Shigeta,W. Tanaka and H. Hasegawa. Automatic Code Generation from SDL to C++ for an Integrated Software Development Support System. In Third International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, FORTE'90, 5-8 November, 1990, Spain.

[11] A.T. Nakagawa, T. Sawada and K. Futatsugi. CafeOBJ User Manual. In World Scientific, 1998.

[12] H. Partsch and R. Steinbrggen. Program Transformation Systems. In ACM Comput. Surv., vol 153,1983, pp 199–236.

[13] R. D. Tavendale. A Technique for Prototyping Directly from a Specification. In 8th International Conference on Software Engineering, pp 224–229, IEEE Computer Society Press,1985.

[14] E. Wegner. Tree-Structured Programs. In Communication of ACM, vol 16, 1973, pp 704–705

A Design Methodology for Tailorable Visual Interactive Systems

Maria Francesca Costabile[1], Daniela Fogli[2], Andrea Marcante[3], Piero Mussio[4], Antonio Piccinno[1]

[1] *Dipartimento di Informatica, Università di Bari, Bari, Italy*
[2] *Dipartimento di Elettronica per l'Automazione, Università di Brescia, Brescia, Italy*
[3] *ITC, CNR via Bassini 15, 20133 Milano, Italy*
[4] *Dipartimento di Informatica e Comunicazione, Università di Milano, Milano, Italy*
{costabile, piccinno}@di.uniba.it, fogli@ing.unibs.it, {marcante, mussio}@dico.unimi.it

Abstract. The paper presents an approach to the design of visual interactive systems that are tailored to users' needs, preferences, culture, skills, background, but also permit users to further personalize them. The approach is based on a new conceptual model for interactive systems, and proposes a method to perform their formal design, which leads to two kinds of specification, one suitable for end users and the other suitable for software engineers. The design specifications are then directly mapped to the implementation architecture, which is based on XML technology. The discussion is carried out with reference to an example in the mechanical engineering domain.

1. Introduction

Technology and work organization co-evolve with continuity, creating new research possibilities and challenges for design and implementation of interactive systems. To study proper approaches to the development of interactive systems, software engineers need to understand the influence of the various problematic aspects characterizing the Human-Computer Interaction (HCI) process. Among such problematic aspects we primarily consider: *communication gap between designers and users* [10], due to the fact that users and designers possess distinct types of knowledge and follow different approaches and reasoning strategies to modeling, performing and documenting the tasks to be carried out in a given application domain; *tool grain* [7], i.e. the system tendency to push the users towards certain behaviors; *implicit information* [11], embedded in user documents and notations and *tacit knowledge* users have of the application domain [13], that designers have difficulties to capture; *user diversity*, since even in the same community, users are different in culture, goals, tasks; *co-evolution of systems and users* [2][3][5][12]. See [6] for a more detailed analysis of such problematic aspects.

This work presents a novel approach and a methodology for designing interactive systems that take the above mentioned problematic aspects into account. It focuses on the conceptual model design [10], on the design specification of the system and on its implementation. The approach is based on a new conceptual model that exploits the workshop metaphor [6]: as artisans (joiners, blacksmiths, etc.) find in their workshops all and only the tools necessary for their specific activities (i.e., for shaping wood, iron, etc.), users should find in their software environments (called Software Shaping Workshops) all and only virtual tools to carry out their activities with the support of the computer system.

Conceptual model design is complemented by formal design, which leads to two forms of system design specification: 1) on one hand, a workshop is specified in terms of a visual rewriting system, including rules describing how the workshop evolves under the effect of user actions; 2) on the other hand, the computational aspects of the workshop are specified by control finite state automata, which are used to express the rewriting rules in a notation suitable for programmers, who can subsequently translate automata into programs. Such specifications result to be coherent and complementary and are directly reflected by the implementation architecture that we propose, based on XML technology.

The emphasis on the proposed conceptual model and its relationship with the formal design are the novelties of this paper, which capitalizes on the work previously performed by some of the authors (e.g., [1][4][6]). The implementation architecture is also a novel proposal in the field. Conceptual model design, formal design, design specification and implementation of an interactive system are described in Sections 2, 3, 4, 5 respectively. Section 6 concludes the paper.

2. The conceptual model design

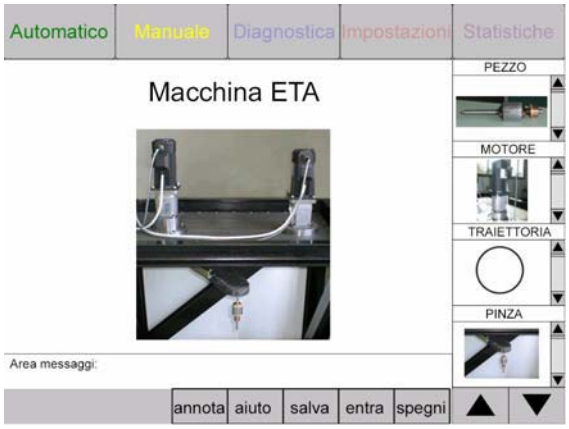
The design methodology we have developed starts by observing professional people in their work practice [4][6]. Such people, e.g. mechanical engineers, geologists, physicians, often work in a team to reach a common goal. The team might be composed by members of different sub-communities, each sub-community with different expertise. Therefore, to take into account user diversity among and within communities, our approach to the design of a software system in a certain domain sees the system as composed of various environments, each one devoted to a specific community or sub-community of end users. Each environment is organized in analogy to an artisan workshop, where the artisan finds all and only the tools necessary to carry out his/her activities. Following the analogy, end users using a virtual workshop, called

Software Shaping Workshop (SSW), find available in their own workshop all and only the tools required to perform their activities.

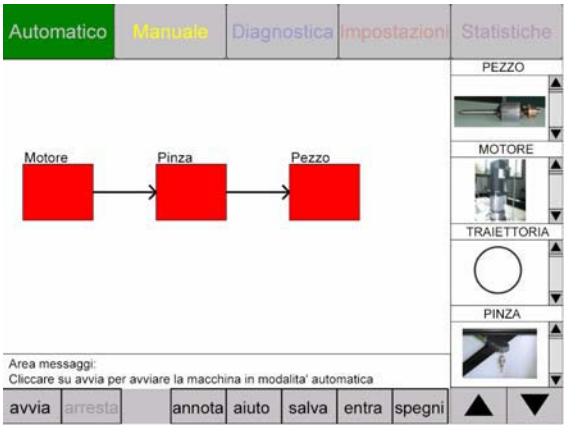
Figure 1 shows the SSW devoted to an assembly line operator controlling a pick-and-place robot, developed for a case study in the mechanical engineering field. Figure 1a shows the initial state of the workshop. The behaviour of the machine is shown in the working area. For example, after the button "Automatico" ("Automatic") has been clicked, the system presents in this area a schematised version of the robot, including the parts to be checked before its activity may start (Figure 1b). When the user clicks the button "avvia" ("start"), the robot starts in an automatic modality. The user can thus observe in the working area that the engine ("Motore"), the gripper ("Pinza") and the piece ("Pezzo") are checked.

The design methodology also requires that each SSW exploits the language and notation adopted by the specific user community to which it is devoted and its layout is organized to convey implicit information: data are organized according to user's culture and working context, and tools are presented by images, texts, or icons, expressive to the users of that particular domain. For example, images of grippers and engines in the right side of Figures 1a and 1b are meaningful to practitioners in the automation systems field. Furthermore, each SSW permits to perform actions that resemble those in the traditional real contexts. For instance, in the workshop devoted to assembly-line operators, users may push buttons to start the machine and to control its subsequent operation. In this way, users are facilitated in exploiting their tacit knowledge while performing their work task.

The methodology emphasizes a perspective of meta-design [8] that goes beyond, but includes user-centered design and participatory design [14]. Meta-design, in our methodology, consists in the design of workshops that end-user representatives may use to design themselves the environments (workshops) for end users. Meta-design permits to bridge the communication gap between end users and software designers and to support co-evolution. Moreover, involving end users in system design permits the development of software environments that do not speak a computer-oriented language and do not induce grains [7]. In fact, in our opinion, only the users themselves are aware of the needs, background, skills and habits of their user community and may build environments that are more usable and acceptable than those ones directly created by software engineers. Thus, in our approach, end users play two main roles in the lifecycle of the interactive software system: 1) they perform their work tasks; 2) they participate in the development of software environments as stakeholders of the domain. In the first role, end users interact with a type of SSWs, called *application workshops*. In the second role, as members of the design team, end users participate directly in the development of the workshops for their daily work. To this end, different workshops, called *system workshops* are made available to



(a)



(b)

Figure 1: The application workshop devoted to assembly-line operators.

Figure 2: A screenshot of the system workshop devoted to mechanical engineers.

them, which permit the customization of each application workshop to the end user community needs and requirements. For example, in our case study in the mechanical engineering field, a system workshop devoted to mechanical engineers was designed to permit the creation of the application workshop devoted to assembly-line operators. A screenshot of this system workshop is shown in Figure 2. It is a software environment tailored to the culture and background of mechanical engineers, which permits to generate application workshops just by direct manipulation, through simple drag-and-drop activities (see [4] for more details). The screenshot in Figure 2 is taken during the creation of the application workshop shown in Figure 1.

In the SSW methodology, the concept of system workshop is general: actually, system workshops are developed to allow the members of each community involved in design and validation of the system to participate in this activity. For example, system workshops for HCI experts and software engineers are used. Each member of the design team can examine, evaluate and modify an application workshop using tools shaped to his/her culture. In this way, this approach leads to a workshop network that tries to overcome the difference in language among the experts of the different disciplines (software engineering, HCI, application domain) who cooperate in developing computer systems customized to the needs of the user communities.

In general, a network is organized in levels: a) *the meta-design level*, where software engineers use a system workshop to prepare the tools to be used and to participate in the design, implementation, and validation activities; b) the *design level*, where software engineers, HCI experts, and end-user representatives cooperate to the design, implementation, and validation of application workshops; c) the *use level*, where end users cooperate to achieve a task, using application workshops customized to their needs, culture, and skills [4][5][6].

3. SSW formal design

Model-based approaches to HCI attempt to identify a unifying framework - a model - to describe the interaction process. The model is used to identify the causes of usability difficulties affecting interactive systems.

An HCI model can be thereafter used to derive design procedures, which allow the implementation of systems in which these difficulties are eliminated or at least reduced. The work presented here is based on the model of the HCI process and on the theory of visual sentences developed in [1]. In that approach, HCI is modelled as a process in which systems of different nature (the cognitive human - the 'mechanical' machine) cooperate to achieve a task. From this point of view, HCI is a process in which user and computer communicate by materializing and interpreting a sequence of messages at successive instants in time. The characterizing point of such HCI model is the recognition of two interpretations of the exchanged messages: one performed by the human, one by the computer that interprets each message according to the rules embedded in a program P. Such program synthesizes how the designers understand the activities to be performed.

The computer execution of P creates and maintains active an entity, that we call *virtual entity* (ve). A virtual entity is a virtual dynamic open system. It is *virtual* in that it exists only as the results of the execution of the program P by a computer; *dynamic* in that its behaviour evolves in time; *open* in that its evolution depends on its interaction with the environment.

P is composed by a set of programs: some of which, In (Input) programs, acquire the input events generated by the user actions; some, AP (APplication) programs, compute the ve reactions to these events; and some, Out (output) programs, output the results of this computation.

A ve manifests its state to the users as a characteristic structure (cs), i.e. a set of pixels on the screen in visual interaction. The interaction begins when the user acts on some input device to manifest his/her requirements or commands to the ve. The ve captures input events generated by user actions and reacts to them generating output events toward users. Output events are characteristic structures materialized on the output devices of a computer to become perceptible by the users.

At each step of this cycle, the ve state is defined as a *characteristic pattern* cp=<cs, u, <intcs, matcs>>, where intcs (interpretation) is a function, mapping the current cs of the ve into the current computational state u of the program AP and matcs (materialization) a function mapping u into cs.

An interactive system (an SSW in our terminology) is constituted by virtual entities interacting one another and with the user through the I/O devices. The user sees the SSW as a whole ve, whose computational state u is materialized at each instant as an image i on the screen. This association can be specified as a triple vs=<i, u, <int, mat>>, where i is the array of pixels constituting the current image, u is a suitable description of the current state of the process determining the reaction of the whole system to user activities, int and mat are two functions relating elements of i with components of u. This triple is called *visual sentence* (vs) in [1], and specifies the state of the whole virtual entity (i.e. the whole SSW).

4. Design specification

The design activity results into two kinds of specification. The first one exploits a notation understandable by end users (then also by HCI experts), based on the visual elements which should appear on the screen as a consequence of user actions and computer reactions. The second kind of specification is suitable to software engineers' experience and background, and it is used as starting point for the implementation activity.

4.1 Specification for end users and HCI experts

Being a visual sentence the state of the virtual entity representing a whole SSW, the set of admissible states of a

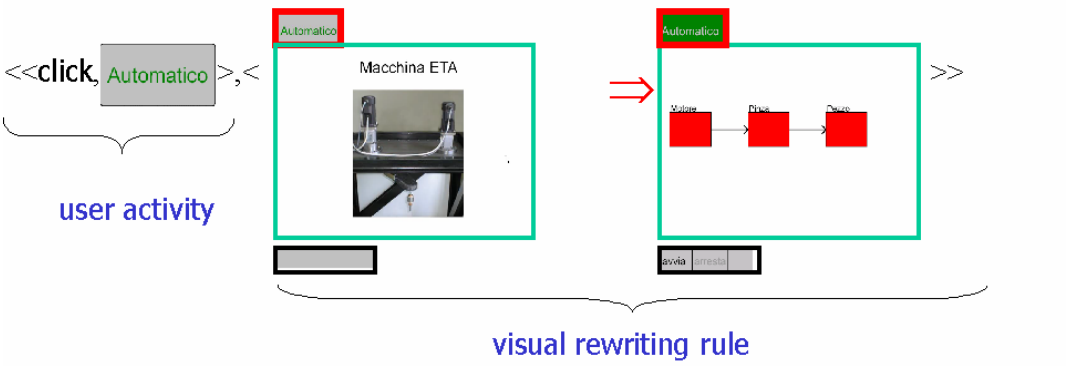


Figure 3. The pictorial part of a transformation rule.

SSW is a set of vss. Visual sentences result from the composition of simpler characteristic patterns, i.e. of the states of the virtual entities composing the SSW.

The design activity aims at specifying the set of vss of a SSW and the transformations between vss. It is thus necessary to define first a finite set (visual alphabet) of cp types, whose instances may be composed to form vss. This visual alphabet is designed in a participatory way by all stakeholders, and is based on user language and notation. Next a Visual Conditional Attributed Rewriting system (vCARW) [1] must be defined on the visual alphabet. A vCARW system contains a set R of visual rewriting rules, which are used to transform a visual sentence vs_1 into another visual sentence vs_2, by introducing new cps or modifying existing ones.

VCARWs are characterized by rules in which only the pictorial part is made explicit to the design team that includes end users and HCI experts, besides software engineers. Therefore, the physical appearance – i.e. topology, geometry and shape – of each cs involved in the rewriting step is defined. On the other hand, the computational meaning of the rules is described in a textual form, or by animations or prototypes, while the technical details about the internal representation of the rules are not explicitly discussed at this stage of development.

The rewriting rules are exploited to specify how the interaction process evolves. In each state of the interaction a finite number of user activities can be performed. As a consequence of a user activity, a visual sentence vs_1 is transformed into a visual sentence vs_2. The interaction process is specified as a sequence of such transformations. For example, the visual sentence whose image part is in Figure 1a is transformed into the visual sentence whose image part is in Figure 1b, as a consequence of the user clicking the button "Automatico".

In a transformation, vs_1 and vs_2 share a common part, while the variable part of vs_1 is transformed into the variable part of vs_2 through the application of a transformation rule in the form tr: <a, r>, where a is the user activity and r is a rewriting rule in R.

Figure 3 shows the pictorial part of the transformation rule that fires when the user clicks the button "Automatico" while interacting with the application workshop shown in Figure 1. The user activity is specified as a pair a=<op, cs>, where op is a physical operation performed by the user (a mouse click in this case) and cs is the set of pixels to which op is referred. The visual rewriting rule defines which css are transformed from one state to the other: in this case, they are three disjoint sets of pixels denoting the resources required to apply the rule. The computational meaning of such rule may be described as follow: the state of the button "Automatico" changes from non selected to selected; the state of the working area changes in order to present a schematisation representing the composition of the robot; the state of the button panel under the working area changes by presenting the operators needed to start and check the robot. This actually means that the states of three virtual entities change, and, as a whole, the system reaches a state in which it is possible to start the machine in the automatic modality and observe the ongoing automatic check process.

In summary, each SSW is formally designed in a collaborative way by software engineers, HCI experts and end users by specifying: 1) a *visual alphabet* of cp types, i.e. the initial states of virtual entities composing the SSW; 2) an *initial state*, vs_0, of the workshop, which is instantiated when the user first accesses the system; 3) a set of rewriting rules R; 4) a set of *transformation rules* TR. Even though the computational meaning of transformation rules is also discussed by stakeholders verbally or through prototypes, this kind of specification focuses more on the visual aspects of the system transformations.

4.2 Specification for software engineers

The four design components described above can be specified in a program-oriented way by using Control Finite State Machines. We use Harel's statecharts [9] to specify a SSW as a composed virtual entity. The hierarchical structure of an SSW, obtained in terms of ve composition, can be easily specified by statecharts

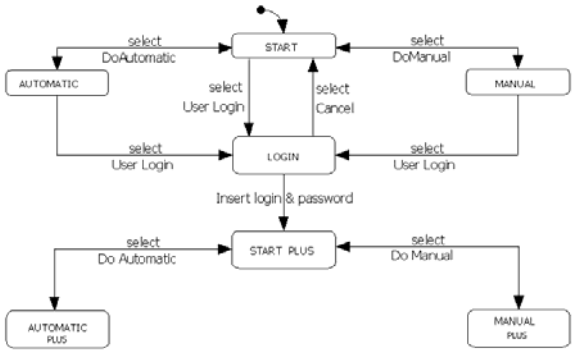


Figure 4. A portion of the statechart specifying the SSW devoted to assembly-line operators.

permitting to model systems at different levels of abstraction.

In the case of SSWs, according to the level of abstraction adopted for the description, the states of the statechart may represent classes of equivalent vss or instances of vss, which are in turn compositions of cps. For example, Figure 4 shows a portion of the statechart specifying the application workshop devoted to assembly-line operators at the highest level of abstraction. The states of this statecharts are classes of visual sentences: for instance, the state "Login" is the class of the visual sentences associated with the login screen shot. Once both login and password have been filled and "Insert" is selected, a new screen shot appears, changing the class of visual sentences representing the next state of the SSW. The transitions from one state to another occur as a consequence of user activities, namely of operations performed with reference to some css included in the image part of the current visual sentence. For example, in Figure 4, "select DoAutomatic" means a selection activity performed on the button "Automatico", i.e. a mouse click on the cs of such virtual entity.

The transformation rules are therefore translated into the transition and output functions of the statechart. In a transformation rule <a, r>, user activity a is the input event firing a transition, r is the rewriting rule to be applied, and thus it specifies which is the next state in the statechart to be reached (transition function) and which computational activity to be activated (output function).

This kind of specification gives more emphasis on the computational meaning of transformations and easily drives the software engineers in the SSW implementation.

5. System implementation

The workshops in a SSW network are implemented as IM^{2}L programs. IM^{2}L (Interaction Multimodal Markup Language) is an XML-based language that provides the rules for the definition of virtual entities that compose the SSW: its markup tags encode a description of storage organization and logical structure of the virtual entities. As shown in Figure 5, an IM^{2}L program includes some XML-based or XML-compliant (ECMAscript) documents and is interpreted by a web browser (see tick arrows), which coordinates the activities of a standard XML processor, an ECMAscript interpreter and an SVG viewer. SVG is the W3C standard for vector graphics [15].

There is a direct mapping between the design specification and the implementation.

The IM^{2}L document contains the description of the initial states of the ves composing the SSW: actually, it specifies the vs_0 of the SSW independently from its materialization. Materialization details are specified in the SVG prototypes, and the instantiation functions permit to create at runtime the vs_0 of the SSW by combining the information present in the IM^{2}L document and in the SVG prototypes.

The interaction managing functions implement the

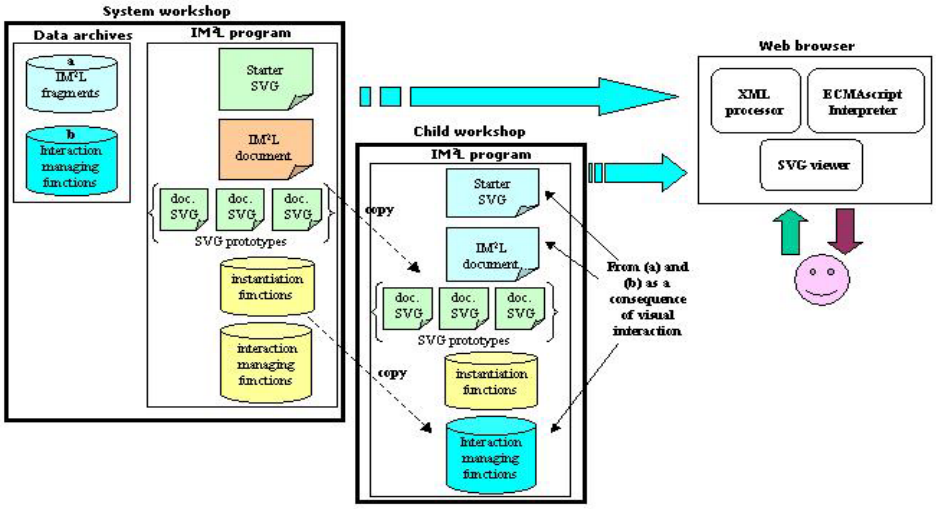


Figure 5. The web-based architecture of a system workshop and the generation of a child workshop.

transformation rules: the interaction of the user with the current state of the SSW means that a particular ve composing the SSW receives an input event and that a related function is called. Such a function may react by: 1) transforming the current state (cp_i) of the ve into another state (cp_{i+1}) so transforming the current state of SSW (vs_i) into another state (vs_{i+1}), being it a composition of the cps of the active ves; 2) generating a new ve in its initial state cp_0, so transforming the current state of SSW (vs_i) into another state (vs_{i+1}) by adding a new cp to vs_i.

We discuss now the generation of a child workshop while interacting with a system workshop. To this end, two special archives of data are exploited. Such archives contain IM^{2}L fragments defining the virtual entities that the user will select to compose child workshops, and the interaction managing functions implementing the transformation rules of such virtual entities (see top left of Figure 5). The IM^{2}L program implementing a child workshop is thus generated by the IM^{2}L program implementing a system workshop in the following way:

1. The archives containing the SVG prototypes and the instantiation functions are simply replicated. The underlying hypothesis is that only the software engineers are in charge of preparing such kinds of information, since they are written by using computer-oriented languages that are not easily manageable by end users and HCI experts.
2. The IM^{2}L document is generated as a consequence of the selection of virtual entities from those available in suitable repositories and their combination into complex virtual entities up to the whole workshop. The user interacts with the system workshop by direct manipulation dragging virtual entities from repositories visualized on the screen to the working area; as a consequence, the IM^{2}L fragments defining the selected virtual entities are automatically searched in the external archive of IM^{2}L fragments.
3. The interaction managing functions of the child workshops are also searched in the data archive as a consequence of virtual entities selection: in fact, they implement the activities to be performed when the user interacts with the virtual entities in the child workshop, so links to them are present in the IM^{2}L fragments defining the virtual entities. All interaction managing functions linked to the IM^{2}L fragments, chosen during the child workshop composition, are therefore included in the archive of interaction managing functions.
4. The starter SVG is generated in the saving phase of the child workshop to permit the initial loading of the IM^{2}L document into the browser.

6. Conclusions

The paper describes a design methodology for tailorable visual interactive systems exploiting the artisan's workshop metaphor. The paper capitalizes on the work performed in several years [1][4][6]. The objective of our work is to generate interactive systems that better fit users' needs and expectations. To this end, our approach takes into account various problematic aspects of the Human-Computer Interaction process.

The users involved in the case study mentioned in the paper understand and appreciate the novel approach of being involved in collaborative design processes, through which they can have a more active role than simple consumers of new technologies. The possibility of having software environments that they can easily adapt to their needs is a new perspective that excites them very much. The results we got so far by observing people using the developed prototypes and collecting their comments and observations are a clear sign that this approach may also determine an increase in end user productivity and performance, but also more pleasure and fun in their overall experience with new technology.

Acknowledgements

The authors wish to thank G. Fresta for his contribution to the implementation of the prototypes. The financial support of Italian MIUR is acknowledged.

References

[1] Bottoni, P., Costabile, M.F., Mussio, P. Specification and Dialogue Control of Visual Interaction through Visual Rewriting Systems, *ACM TOPLAS*, Vol. 21, No. 6, 1077-1136, 1999.

[2] Bourguin, G., Derycke, A., Tarby, J.C. Beyond the Interface: Co-evolution inside Interactive Systems - A Proposal Founded on Activity Theory, *Proc. IHM-HCI 2001*.

[3] Carroll, J.M., Rosson, M.B., Deliberated Evolution: Stalking the View Matcher in design space. *Human-Computer Interaction*, 6 (3 and 4), 1992, 281-318.

[4] Costabile, M.F., Fogli, D., Fresta, G., Mussio, P., Piccinno, A., Software Environments for End-User Development and Tailoring, *Psychnology*, 2(1), 2004, 99-122.

[5] Costabile, M.F., Fogli, D., Marcante, A., Piccinno, A. Supporting Interaction and Co-evolution of Users and Systems, *Proc. AVI 2006*, Venezia, Italy, 23-26 May 2006, in print.

[6] Costabile, M.F., Fogli, D., Mussio, P., Piccinno, A. End-User Development: the Software Shaping Workshop Approach, in H. Lieberman, F. Paternò, V. Wulf (Eds), *End User Development*, Springer, Dordrecht, The Netherlands, 2006, pp. 183-205.

[7] Dix, A., Finlay, J., Abowd, G., Beale, R. *Human Computer Interaction*, Prentice Hall, London, 1998.

[8] Fischer, G., Giaccardi E., Meta-Design: A Framework for the Future of End-User Development, in H. Lieberman, F. Paternò, V. Wulf (Eds), *End User Development*, Springer, Dordrecht, The Netherlands, 2006, pp. 427-457.

[9] Harel, D., On visual formalisms. *CACM*, 31(5), 1988, 514-529.

[10] Majhew, D.J. *Principles and Guideline in Software User Interface Design*, Prentice Hall, 1992.

[11] Mussio, P. E-Documents as tools for the humanized management of community knowledge. Keynote Address, *ISD 2003 Proc.*, Melbourne, AUS, 2003.

[12] Nielsen, J., *Usability Engineering*, Academic Press, San Diego, 1993.

[13] Polanyi, M., *The Tacit Dimension*, Rouledge & Kegan Paul, 1967.

[14] Schuler, D., Namioka, A., Preface, *Participatory Design, Principles and Practice*, Lawrence Erlbaum Ass. Inc.Hillsday, vii, N.J, 1993.

[15] W3C: Scalable Vector Graphics (SVG), [Online] 2001 <http: //www.w3.org/Graphics/SVG/

A Method for Modeling Object-Oriented Systems with PZ nets

Ying Huang Xudong He
School of Computing and Information Sciences
Florida International University
E-mail: {yhuang02, hex}@cs.fiu.edu

Abstract

Centered on data abstraction and encapsulation, object-oriented technologies have been widely accepted in software industry in the past decade. However, existing object technologies lack precise semantics, which makes them hard to analyze. In this paper, a group of patterns is presented to formalize object-oriented systems with PZ nets - a formal integration of Petri nets and Z language. These patterns use Petri nets to depict the overall system structure and control flows, class structures and collaborations, and dynamic behavior, and use Z to define the underlying specification of Petri nets.

1. Introduction

Object-oriented technologies are widely used in software industry in the past decade for better reusability, compositionality, and maintainability. However, current object-oriented development methods such as UML [13] lack precise semantics.

Recently, many efforts have been undertaken to make object-oriented development methods more precise and thus support formal analysis. For example, many different formal methods have been used to formalize UML [1], and extended notations were added to existing formal methods to define object-oriented systems. CCPN [2] defined objects with Colored Petri nets [10] and realized synchronous/asynchronous communication by transition/place fusion. CO-OPN/2 [3] (Concurrent Object Oriented Petri nets) used algebraic abstract data type specification to model data structures and algebraic Petri nets to define object behavior. In [7], Hierarchical Petri nets [5] were employed to represent object-oriented concepts and features. Elementary Object Nets [17] and Object Petri nets [11] adapted OO-concepts into Petri nets [10] by allowing tokens to be subnets. Object-Z [15] added specific constructs to Z notations [16] and introduced the concept of data scope to define object-oriented system.

In this paper, a group of patterns called PZ net object patterns is introduced to model object-oriented systems with PZ nets [6]. In these patterns, Hierarchical Predicate Transition Nets - HPrT nets [5] are used to define overall system structures, class collaborations, inheritance relationships among classes, internal structures of classes and their dynamic behavior. Z notations [16] are used in node inscriptions and arc labeling of HPrT nets to refine data abstraction and to impose constraints of object-oriented concepts in transition enabling and flow controls.

Former approaches in modeling object-oriented systems by Petri nets or by Z notations either only partially support the major features of object-oriented paradigm or do not have well organized class structure to facilitate the model analysis and simulation. By integrating the rich type definition facilities of Z notation, concurrency and distributed features of Petri nets, and modular and hierarchical development mechanism in object-oriented systems, PZ net object patterns well demonstrated the object-oriented features such as encapsulation, inheritance, polymorphism and dynamic binding. Compared to object-oriented design patterns [4], PZ net object patterns use formal methods to model overall system structures and class collaborations in a higher level.

2. PZ nets

Definition 2.1: A PZ net is a tuple $\alpha = (N, Z, ins)$ [6], in which

1. $N = (P, T, F, \rho)$ *is a hierarchical predicate transition net [5], in which*
   - *P and T are the sets of places and transitions respectively.*
   - $F \subseteq (P \times T) \cup (T \times P)$ *is the flow relation between P and T.*
   - $\rho : P \cup T \rightarrow \wp(P \cup T)$ *is the mapping that defines the hierarchical relationships among places and transitions.*
2. $Z = (Z_{P.S}, Z_T, Z_{P.I})$. $Z_{P.S}$, Z_T, *and* $Z_{P.I}$ *are Z schemas that define the states, constrains, and initial markings of N respectively.*
3. $ins = (S, C, L, M_0)$ *is a net inscription that associates a net element in N with its notation in Z as follows:*

- $S : P \rightarrow Z_{P.S}$ *is a one-to-one relation that defines the places of N.*
- $C : T \rightarrow Z_T$ *is a one-to-one relation that defines the transitions of N.*
- $L : F \rightarrow \wp Var$ *is the control flow definition of N, where Var is the set of all hidden variables in* Z_T.
- $M_0 : P \rightarrow Z_{P.I}$ *is an initial marking of N.*

In PZ nets, place *p* has two associated Z state schemas, $Z_{p.S}$ and $Z_{p.I}$. $Z_{p.S}$ defines the state components of *p*. $Z_{p.I}$ specifies the initial marking of *p*. Transition *t* has an associated Z operation schema Z_t to define its pre-condition and post-condition. Therefore, a 3-tuple (p, $Z_{p.S}$, $Z_{p.I}$) is used to define a non-super place *p* and a 2-tuple (t, Z_t) is used to define a non-super transition *t*. In this paper, we use *pz* and *tz* as the abbreviations of (p, $Z_{p.S}$, $Z_{p.I}$) and (t, Z_t) to simplify the notation and minimize redundant information.

The characteristics of Petri nets restrict the effective scope of tokens to their places. Therefore, PZ nets put data scope restrictions on Z schemas. In other words, Z_t can only access the data components of $Z_{\bullet t}$ and $Z_{t\bullet}$, which define the behavior of the input and output places of transition *t*.

In the rest of the paper, the labels of the arcs in PZ nets are omitted to present a clear view of the relationship among net nodes. Those label associated hidden variables are implicitly listed in the Δ list of Z_t.

3. Object Patterns

PZ net object patterns specify an object system that consists an object server, a group of running objects, and communication scheme. The object server supports communication and distributed object-naming. The well-defined communication scheme collaborates the running objects. Figure 1 depicts the structure of the object system.

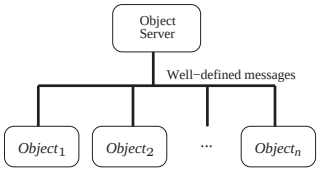


Figure 1. Object Management System

In this paper, the object server is modeled by *system net*, the running objects are specified by *class nets*, and the message passing scheme is defined by *channel nets and link nets*. Due to page limit, the complete Z specifications for these patterns are omitted. They can be found in [9].

3.1. Communication Patterns

Communication scheme is one of the fundamentals of object encapsulation and data abstraction. It helps objects interact with each other without revealing their internal structures. Message passing and dispatching occurs both inter- and intra-object in object systems. The communication patterns defined in this section handle only intra-object message passing. Inter-object message passing is realized through *system net* (section 3.3). Message dispatching is acchieved by checking the pre-conditions of transitions.

There are two kinds of communication channels: message channels and messenger channels. The formers are super places that act as shared mailboxes, and the latters are super transitions that behave like postmen who transfer messages from one mailbox to another. In PZ net specifications, they are called *channel nets* and *link nets* respectively.

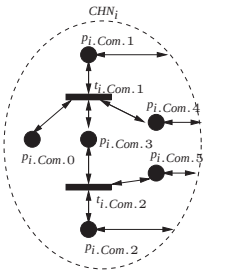


Figure 2. Pattern for Channel Net

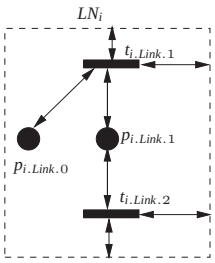


Figure 3. Pattern for Link Net

Definition 3.1: A *channel net* is a tuple CHN_i = (Com_0, Com_1, Com_2, Com_3, Com_4, Com_5, Com_6, Com_7), in which

1. $Com_0 = pz_{i.Com.0}$, *which holds an identifier for the firing of* $t_{i.Com.1}$.
2. $Com_1 = pz_{i.Com_1}$, *which contains invocation messages and the callers' information.*
3. $Com_2 = pz_{i.Com_2}$, *which holds invocation results and the callers' information.*
4. $Com_3 = pz_{i.Com_3}$, *which keeps the callers' information.*
5. $Com_4 = pz_{i.Com_4}$, *which stores invocation waiting to be forwarded to the callee.*
6. $Com_5 = pz_{i.Com_5}$, *which saves invocation results.*
7. $Com_6 = tz_{i.Com_1}$, *which splits information in* Com_1 *to caller relevant information and callee relevent information.*
8. $Com_7 = tz_{i.Com_2}$, *which integrates invocation results with the callers' information.*

Each method call invokes the following process. Firstly, Com_1 receives an invocation message with caller's information (i.e. the return address, etc). Secondly, Com_6 generates a new message identifier, splits the invocation message to caller- and callee- related information, attaches the message id to both of them, and deposits the former into Com_3 and the latter into Com_4. After the operation has been done by

the callee, Com_5 gets the result from the callee. Finally, Com_7 matches the id of the caller's information in Com_3 with the id of the result in Com_5, and combines them to get an invocation result with caller's information for Com_2.

In Figure 2, every arc is bi-directional because using Z schemas to define tokens results in schema state change when a token is consumed or produced.

Definition 3.2: A *link net* is a tuple $LN_i = (Link_0, Link_1, Link_2, Link_3)$, in which

1. $Link_0 = pz_{i.Link.0}$, *which holds an id for the firing of* $t_{i.Link.1}$.
2. $Link_1 = pz_{i.Link.1}$, *which holds information of callers.*
3. $Link_2 = tz_{i.Link.1}$, *which splits tokens in* $p_{i.Link.1}$ *to caller relevant messages and callee relevant messages.*
4. $Link_3 = tz_{i.Link.2}$, *which integrates invocation results with the information of their callers.*

A *link net* forwards external function call to the *system net* and receives the results from the *system net*. $Link_2$ receives an invocation message from Com_4, generates a new message id, splits the invocation message to caller- and callee-related information, attaches the id to both of them, saves the former in $Link_1$, and forwards the latter to *system net* (defined in 3.3). $Link_3$ matches the ids of the caller's information in $Link_1$ and the result in *system net*, and combines them to get a result with caller's information for Com_5.

3.2. Class Patterns

Object encapsulation is a unique feature of object-oriented systems. An object is known to others through its behavior. The state of an object is the aggregation of its attributes. It reflects the cumulative results of its behavior.

In this paper, *object state nets* are used to define the state of class instances, and *class nets* are used to define the state and dynamic behavior of objects. The *object state nets* of a class is enclosed in its *class nets*.

3.2.1 Object state net

Definition 3.3 An *object state net* is a tuple $OSN_i = (InsVars_i, ClassVars_i)$, in which

1. $InsVars_i = pz_{i.insAttr}$, *which acts as the memory allocation for objects of class i.*
2. $ClassVars_i$ *is a set of tuples* $pz_{i.classVar_j}$. *Each of them defines a class attribute*[1] *of class i.*

The templates of $Z_{p_{i.insAttr.S}}$ and $Z_{p_{i.classVar.S}}$ are as follows:

$$ATTRIBUTE ::= NAME \mapsto TYPE \times VALUE$$
$$OBJECT ::= (className : NAME, objID : NUMBER, [attr_1, attr_2, \ldots, attr_n : ATTRIBUTE])$$

[1] A class attribute is a static variable whose value is shared by all class instances.

$$Z_{p_{i.insAttr.S}}: obj : \mathbb{P}\, OBJECT$$

$$Z_{p_{i.classVar.S}}: classVar : (className : NAME, attr : ATTRIBUTE)$$

From now on, we use *attr*.*Type* and *attr*.*Value* to denote the type and value of attribute *attr*.

3.2.2 Class net

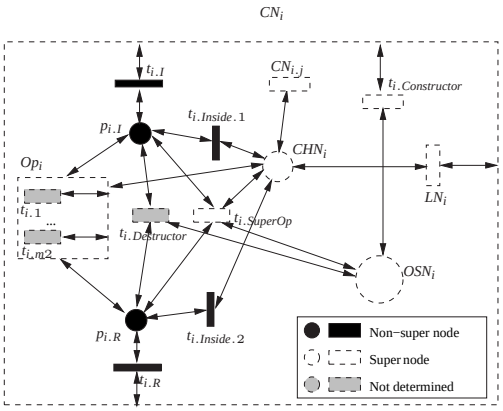


Figure 4. Pattern for Class Net

Definition 3.4: A *class net* is a tuple $CN_i = (OSN_i, Constructor_i, Destructor_i, Op_i, IncludedClass_i, In_i, Out_i, COM_i, LN_i, Inherit_i)$, in which

1. OSN_i *is the* object state net *of class i.*
2. $Constructor_i$ *is a set of tuples* $tz_{i.Constructor}$. *Each of them represents a constructor of class i.*
3. $Destructor_i = tz_{i.Destructor}$, *which is the destructor of class i.*
4. $Op_i = \{tz_{i.1}, tz_{i.2}, \ldots, tz_{i.m2}\}$, *which is a set of methods defined on class i.*
5. $IncludedClass_i = \{CN_{i.1}, CN_{i.2}, \ldots, CN_{i.m3}\}$, *which is the set of classes defined inside class i.*
6. $In_i = \{tz_{i.I}, pz_{i.I}\}$, *which receives invocation messages.*
7. $Out_i = \{tz_{i.R}, pz_{i.R}\}$, *which returns invocation results.*
8. $COM_i = \{CHN_i, tz_{i.Inside1}, tz_{i.Inside2}\}$, *which acts as the communication channel in the class.*
9. LN_i *communicates with other* class nets *via the* system net.
10. $Inherit_i = tz_{i.SuperOp}$, *which calls operations defined in one of the super-classes of class i.*

In CN_i, $IncludedClass_i$ models the classes defined inside class *i* and can be empty. Op_i contains accessor transitions[2] and transitions that define object behavior. CHN_i and LN_i are optional in CN_i. They receive method invocation messages from method defined on class *i*, route them to the destinations (inside or outside CN_i), receive the replies, and forward them back to the callers. $t_{i.Inside.1}$ receives messages from CHN_i. It initializes method invocations on the same object. $t_{i.Inside.2}$ sends the results back to CHN_i. $t_{i.SuperOp}$ is an optional super-transition that exists only in sub-classes. It accepts all invocations that are not overridden in class *i* and forward them to one of the super-classes. Its priority is lower than $t_{i.1}$, $t_{i.2}$, . . ., $t_{i.m2}$.

If class *i* is inherited from class *j*, for each object token in CN_i, there is an object token with the same object id in CN_j. The design of co-existing objects for super- and sub-classes demonstrate the concept of inheritance well. Without this mechanism, [7] and [11] duplicated the structures of super-classes in the nets of their sub-classes, which did not support inheritance well.

3.3. System Net Pattern

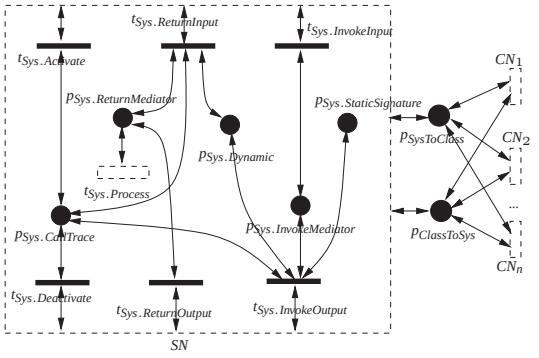


Figure 5. Pattern for System Net

Definition 3.5: A *system net* is a tuple *SN* = (*Interface*$_{Sys}$, *Process*$_{Sys}$, *Mediator*$_{Sys}$, *SysInfo*$_{Sys}$), in which

1. *Interface*$_{Sys}$ = $\{tz_{Sys.Activate}, tz_{Sys.Deactivate}\}$. $tz_{Sys.Activate}$ *receives system activation messages from the environment.* $tz_{Sys.Deactivate}$ *returns control and results to the environment.*
2. *Process*$_{Sys}$ = $tz_{Sys.Process}$. $tz_{Sys.Process}$ *defines a super-transition that realizes system processing logic*[3].
3. *Mediator*$_{Sys}$ = $\{tz_{Sys.InvokeInput}, tz_{Sys.InvokeOutput}, tz_{Sys.ReturnInput}, tz_{Sys.ReturnOutput}, pz_{Sys.InvokeMediator}, pz_{Sys.ReturnMediator}\}$. $tz_{Sys.InvokeInput}$ *receives invocation initialization messages from class nets through* $p_{ClassToSys}$; $tz_{Sys.InvokeOutput}$ *sends invocation messages to the callees through* $pz_{SysToClass}$; $tz_{Sys.ReturnInput}$ *receives results from class nets through* $pz_{ClassToSys}$; $tz_{Sys.ReturnOutput}$ *sends results to the callers through* $pz_{SysToClass}$. $pz_{Sys.InvokeMediator}$ *and* $pz_{Sys.ReturnMediator}$ *are message buffers in SN.*
4. *SysInfo*$_{Sys}$ = $\{pz_{Sys.StaticSignature}, pz_{Sys.Dynamic}, pz_{Sys.CallTrace}\}$, *which holds static and dynamic information that supports system running.* $pz_{Sys.StaticSignature}$ *contains the signatures for each methods defined in classes;* $pz_{Sys.Dynamic}$ *records class instances;* $pz_{Sys.CallTrace}$ *saves inter-object references history.*

In Figure 5, $p_{SysToClass}$, and $p_{ClassToSys}$ are message buffers between *class nets* and *system net*. In a PZ net model, the system net acts both as the object server and as the inter-object communication hub. The environment activates the system by firing $t_{Sys.Activate}$. The firing of $t_{Sys.Activate}$ deposits an initial call trace token into $p_{Sys.CallTrace}$, which saves the inter-object references history for a system activation. The net inside super-transition $t_{Sys.Process}$ is the main process of the system, whose structure is determined by the concrete system. $t_{Sys.Process}$ interacts with class nets $CN_1, CN_2, \ldots, CN_i$ through *Mediator*$_{Sys}$, $p_{SysToClass}$, and $p_{ClassToSys}$. $t_{Sys.Deactivate}$ returns the execution result to the environment.

To simplify the system design and specification, this paper focuses on single entrance systems, i.e., only one instance of the system can be running at a time. Multiple entrances can be allowed in the system by attaching thread ids to tokens and comparing thread ids in the pre-conditions of transitions.

3.4. Object System

Based on the patterns introduced above, an object-oriented system specified with PZ nets can be defined as follows:

Definition 3.6: An *object system* is a tuple $OS = (\alpha, SN, \kappa, \pi, \lambda)$, where

1. α = *(N, Z, ins) is a PZ net.* $N = SN \cup \kappa \cup \lambda$.
2. *SN is a* system net.
3. $\kappa = \bigcup_{i=1}^{n} CN_i$, *where* $CN_i (i = 1, 2, \ldots, n)$ *are* class nets *and n is the number of classes.*
4. $\pi : \kappa \rightarrow \wp(\kappa)$. π *defines the inheritance relationships among classes.* $\pi(CN_i) = \{CN_1, CN_2, \ldots, CN_j\}$, *where class 1, class 2, . . ., class j are the immediate super-classes of class i.*
5. $\lambda = \{p_{SysToClass}, p_{ClassToSys}\}$ *are the message buffers between the* system net *and* class nets.

3.5. Dynamic Semantics of Object Systems

Definition 3.7 The behavior of an *PZ net object system* is the collection of all valid execution sequences starting from the initial marking, in which

[2] An accessor transition gets or changes the value of an attribute according to the visibility of the attribute and the relationship between the class of the caller and class *i*.

[3] Its functionality is similar to that of the main function of java program.

1. $M : P \rightarrow \textbf{\textit{sig}}(Z_{P.S})$. *In other words, markings are the states of schemas* $Z_{P.S}$.
   $M_0 : P \rightarrow \textbf{\textit{sig}}(Z_{P.I})$. M_0 *is the initial marking of OS.*
2. $Var : T \rightarrow \mathbb{P}\, X$, *where* T *is the set of transition,* X *is the set of variables, and* $Var(t)$ *is the set of all hidden variables in* Z_t.
   $b : X \rightarrow VALUE$. $b(x) = v$ *binds variable* x *to value* v.
3. *Use* $Z_t : b$ *to denote the schema* Z_t *under specific binding* b. *Transition* $t \in T$ *is* b*-enabled if and only if* $\textbf{\textit{pre}}(Z_t) : b$ *evaluates to true.*
4. *If* t *is* b*-enabled at* M, t *may fire in binding* b. *The firing of* t *with binding* b *returns marking* M'*:*
   $\forall\, p \in P, M'(p) = M(p) - \bar{L}(p,t) : b \cup \bar{(}L)(t,p) : b$
5. *Any execution sequence* $M_0 t_0 M_1 t_1 \ldots t_{k-1} M_k$ *satisfies the condition:* $\forall\, i, 0 < i < k \wedge \exists\, b \bullet \textbf{\textit{pre}}(Z_t) : b = \textbf{\textit{true}} \wedge \forall\, p \in P, M_{i+1}(p) = M_i(p) - \bar{L}(p,t) : b \cup \bar{(}L)(t,p) : b$.
6. $\omega = [M_0\rangle$. *It is the collection of all valid firing sequences starting from* M_0.

4. Modeling Object-Oriented Features Using PZ net Object Patterns

Object-oriented systems have following major characteristics: encapsulation, inheritance, polymorphism and dynamic binding, etc. This section shows how PZ net object patterns capture these features.

4.1. Encapsulation

Through encapsulation, the object state components and behavioral implementation details are bundled into a single package and hidden from the outside. Encapsulation has at least three benefits [14]: reduced side effects resulting from super-classes modification (maintainability), simplified interfaces among objects (compositionality), and better component reusing (reusability).

In PZ net object patterns, objects are well encapsulated. *Object state nets* pack object attributes; *channel nets* and *link nets* provide unified communication channels for intra-object communication; *class nets* bundle object states and behaviors. Objects interact with each other via *system net*.

4.2. Inheritance

PZ net object patterns keep object tokens in both sub-class and super-class. An object token of CN_i has a correspondent object token in each element of $\pi(CN_i)$. Any non-overridden operations are captured by $Inherit_i$ and forwarded to the correct super-class via *system net*. This mechanism makes the attributes and operations of super-classes automatically available to their sub-classes and propagates any changes in the super-classes immediately to their sub-classes. Through this mechanism, the duplication of net structure and inscription is prevented. Its type casting through *system net* also facilitates the system analysis.

4.3. Polymorphism and dynamic binding

In class net CN_i, the polymorphic operations are treated as separate transitions. The token in $p_{i.I}$ contains the operation name and its parameter list. The pre-condition of each transition in Op_i is set according to the signature of the operation. Since $t_{i.SuperOp}$ has the lowest priority among $p_{i.I}\bullet$, the redefined operations in class i override the ones in its super-classes.

Dynamic binding is activating different operations according to the object type. In PZ net object patterns, multiple object tokens with the same object id are reside in different class nets to represent the same object in the inheritance hierarchy. Therefore, dynamic binding is achieved automatically by *system net* that directs function call to the corresponding *class net* .

4.4. Message passing and dispatching

The mechanism of message passing and dispatching is one of the fundamentals of object encapsulation. Only with standard message structure, objects can interact with others without revealing their internal structures.

PZ net object patterns distinguish between intra- and inter-object communication. For the former, *channel net* acts as the messenger between transitions. For the latter, *system net* simplifies the structure of message buffers between *system net* and *class nets* by acting as the communication coordinator. This simplification reduces the workload on the buffers and facilitates model analysis through centralized information recording. For message dispatching, PZ nets object patterns rely on the pre-conditions of transitions to screen out the unmatched incoming tokens.

4.5. Class containment

There are two whole-part relationship among classes: aggregation and containment. The component classes of the former are defined outside of their container class. Therefore, they are visible to other classes. The component classes of the latter are defined inside the container class. And thus they are invisible to outsiders.

PZ nets object patterns model both class aggregation and containment. For aggregation, the whole class uses component class with the coordination of the system net. For class containment, the component's class net is enclosed by the container's class net. They communicate through the channel net of the container class.

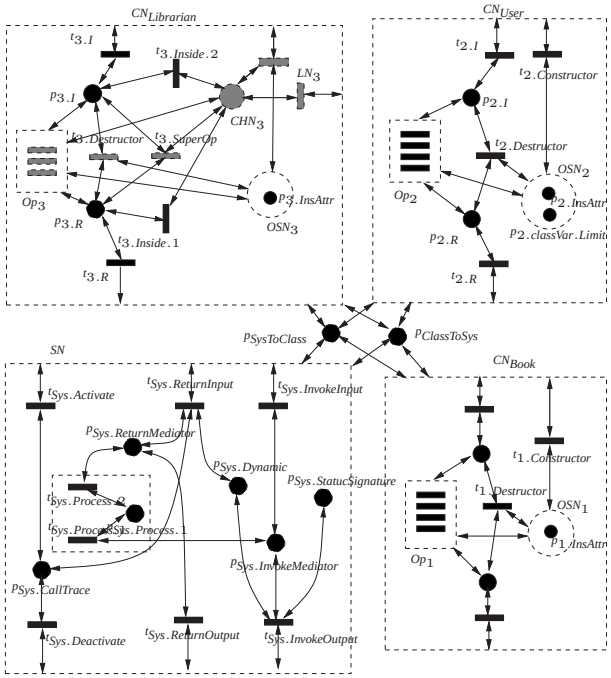


Figure 6. A PZ net specification for a simplified Library System

4.6. Concurrency

PZ net object patterns maintain the same distributed and concurrency features of Petri nets. Concurrency happens at both object level and method level. Each method call on an object removes the object token from the object state net and deposits it back once the operation is done. Therefore, intra-object concurrency is limited.

5. An example - Library System

In this section, a simplified library system is used to demonstrate how to model an object-oriented system with PZ net object patterns.

The system contains a set of ordinary patrons, librarians, and books. It supports the following transactions: check-out, return, and list. A librarian is allowed to perform all of the transactions while an ordinary patron can only list the books that he borrowed. The system can be specified as $\delta = (\alpha, SN, \kappa, \pi, \lambda)$, in which

1. $\alpha = (N, Z, ins)$. $N = SN \cup \kappa \cup \lambda$.
2. $\kappa = \{CN_{Book}, CN_{User}, CN_{Librarian}\}$. *In those class nets:*
   $Op_{Book} = \{tz_{Book.Checkout}, tz_{Book.Return}, tz_{Book.GetPatron}, tz_{Book.GetTitle}\}$.
   $Op_{User} = \{tz_{User.GetName}, tz_{User.GetList}, tz_{User.AddList}, t_{User.DeleteList}\}$.
   $Op_{Librarian} = \{tz_{Librarian.Checkout}, tz_{Librarian.CheckList}, tz_{Librarian.Return}\}$.
3. $\pi(CN_{Book}) = \pi(CN_{User}) = \phi$, $\pi(CN_{Librarian}) = \{CN_{User}\}$.
4. $\lambda = \{p_{SysToClass}, p_{ClassToSys}\}$.

Figure 6 gives the net structure for the library system. To simplify the notations, class 1, 2, and 3 are used to indicate class Book, User, and Librarian respectively. Due to page limit, this paper only gives the PZ net specification for operation *Checkout* of class *Book*. The detail net structures and inscriptions of the library system can be found in [9].

$$
\begin{array}{|l}
\hline
Z_{t_{1.Checkout}} \\
\Delta(Z_{p_{1.I.S}}, Z_{p_{1.insAttr.S}}, Z_{p_{1.R.S}}) \\
\hline
\exists(Book_1, Ops_1, Src_1) \in Z_{p_{1.I.S}} \wedge \exists\, book \in Z_{p_{1.insAttr.S}} \bullet \\
\quad Ops_1.Name =' Checkout' \wedge \mathbf{Types}(Ops_1.ParamList) = \\
\quad \langle \mathbf{NAME} \rangle \wedge book.bookTitle = \mathbf{head}(Ops_1.ParamList) \\
((book.patronID.Value = \mathbf{null} \wedge Ops'_2.Result = (\mathbf{BOOLEAN}, \mathbf{true}) \wedge \\
\quad book'.patronID = (\mathbf{NUMBER}, \mathbf{head}(Ops_1.ParamList))) \vee \\
(book.patronID.Value \neq \mathbf{null} \wedge Ops'_2.Result = (\mathbf{BOOLEAN}, \mathbf{false}) \wedge \\
\quad book'.patronID = book.patronID)) \wedge book'.objID = \\
\quad book.objID \wedge book'.className = book.className \wedge \\
\quad book'.bookTitle = book.bookTitle \wedge Ops'_2.Name = Ops_1.Name \\
\quad \wedge Ops'_2.ParamList = Ops_1.ParamList \wedge \\
\quad Z'_{p_{1.I.S}} = Z_{p_{1.I.S}} \setminus \{(Book_1, Ops_1, Src_1)\} \wedge \\
\quad Z'_{p_{1.insAttr.S}} = Z_{p_{1.insAttr.S}} \setminus \{book\} \cup \{book'\} \wedge \\
\quad Z'_{p_{1.R.S}} = Z_{p_{1.R.S}} \cup \{(Book_1, Ops'_2, Src_1)\} \\
\hline
\end{array}
$$

$$
\begin{array}{l}
BOOK : (className =' BOOK', objID : OBJECTID, \\
\quad bookTitle : (\mathbf{NAME}, NAME), patronID : (\mathbf{NUMBER}, NUMBER)) \\
PARAM\_TYPE : (Type : TYPE, Value : ANY) \\
PARAMLIST\_TYPE : seqPARAM\_TYPE
\end{array}
$$

Book.*Checkout*() checks out the book to the patron if it is available. Otherwise, an error message is given. $Z_{t_{1.Checkout}}$[4] defines the behavior of $t_{1.Checkout}$.

6. Related Work

CCPN [2] used net addition to model inheritance and node fusion to model message passing. The former might not support multiple inheritance and the latter is not supported in real systems. Object Petri nets [11] allowed token to be objects, but it duplicated the data structure of superclass in the sub-class, which does not support inheritance well. It also did not encapsulate object behavior. Object-Z [15] did not have behavior model, which makes it hard to verify. TCOZ [12] combined timed object-Z and CSP [8] to model realtime object systems. The employment of object-Z prevented it from supporting method polymorphism, plus it lacked the visual demonstration ability provided in Petri nets. Due to page limit, table 1 only gives a brief comparison of some formal methods regarding their OO-feature modeling capabilities. A detailed comparison is given in [9].

[4] In $Z_{t_{1.Checkout}}$, **Types** : seq *PARAM_TYPE* $\rightarrow$ seq *TYPE*. It extracts the Type of each (Type, Value) pair in a list of PARAM_TYPEs and composes a list of Types.

Table 1. Comparison of Formal Methods on Modeling OO Features

	Encapsulation	Inheritance	Polymorphism	Message Passing	Concurrency
CCPN	Yes	Partial	No	Yes	Yes
CO-OPN/2	Yes	Yes	No	No	Yes
Elementary Object nets	Yes	No	Yes	No	Yes
HPrTNs	Partial	No	Yes	Yes	Yes
Object PNs	Yes	Partial	No	No	Yes
Object-Z	Yes	Yes	No	No	No
PZ net Object Patterns	Yes	Yes	Yes	Yes	Yes

As the de facto industrial standard for object-oriented system design, UML represents a system in different diagrams that depict the system from many aspects: users, system structures, system component behaviors, implementation, and environment [14]. PZ net object patterns define an object-oriented system with Hierarchical Predicate Transition nets with Z schemas. Compared to PZ net object patterns, UML is more user-friendly, which helps programmers and users to understand the system. However, UML is not a formal method. Its diagrams cannot be used to verify system properties such as liveness, safeness, and fairness. While PZ nets can be formally analyzed to ensure the system properties before the implementation.

7. Conclusion

Compared to other object formalizing methods, the PZ net object patterns introduced in this paper have well-organized net structures and standard communication protocols. Their unified class net structures and standardized interfaces are well suited to define object-oriented systems.

Focusing on system net analysis, several algorithms were developed to analyze system models by checking the reference history recorded in *system net*. They can detect the dataflow abnormalities, check the inheritance and reference relationships among classes, and verify resource allocation during the system execution, etc. These algorithms can be found in [9].

A tool that simulates and analyzes PZ net object models is under construction. With pre-defined system framework and inscriptions for class nets and system net, this tool takes class specification and system processing logic as inputs, simulates the behavior of an object system, and analyzes the system by examining its simulation history.

Acknowledgements This research was supported in part by the National Science Foundation of the USA under grant HRD-0317692, and by the National Aeronautics and Space Administration of the USA under grant NAG2-1440.

References

[1] T. Baar, A. Strohmeier, A. M. D. Moreira, and S. J. Mellor, editors. *UML 2004 - The Unified Modelling Language: Modelling Languages and Applications. 7th International Conference*, LNCS 3273. Springer, 2004.

[2] J. P. Barros and L. Gomes. On the use of coloured petri nets for object-oriented design. In *Proceedings of the 25th International Conference of Applications and Theory of Petri Nets*, pages 117–136, Bologna, Italy, June 21-25 2004.

[3] O. Biberstein and D. Buchs. An Object Oriented Specification Language based on Hierarchical Algebraic Petri Nets. In *Proceedings of the IS-CORE Workshop*, Amsterdam, Netherlands, 1994.

[4] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns - Elements of Reuseable Object-Oriented Software*. Addison-Wesley Longman, Inc., 1994.

[5] X. He. A Formal Definition of Hierarchical Predicate Transition Nets. In *Proc. of the 17th Intl. Conference on Application and Theory of Petri Nets (ICATPN'96)*, LNCS 1091, pages 212–229, Osaka, Japan, 1996.

[6] X. He. PZ Nets - A Formal Method Integrating Petri Nets with Z. *Information and Software Technology*, 43(1):1–18, 2001.

[7] X. He and Y. Ding. Object-Orientation in Hierarchical Predicate Transition Nets. In *Concurrent Object-Oriented Programming and Petri Nets*, LNCS 2001, pages 196–215. Springer-Verlag, 2001.

[8] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International, 1985.

[9] Y. Huang. *PZ nets object patterns*. School of Computer Science, Florida International University, Miami, Florida, USA, December 2004. http://www.cs.fiu.edu/ yhuang02/OPZ_Patterns_Spec.pdf.

[10] K. Jensen. *Coloured Petri nets: Basic concepts, analysis methods and practical use*. Springer-Verlag, 1996.

[11] C. Lakos. The Object Orientation of Object Petri Nets. In *Proc. of the 1st Intl. Workshop on Object-Oriented Programming and Models of Concurrency*, pages 1–14, 1995.

[12] B. Mahony and J. S. Dong. Timed communicating object z. *IEEE Trans. Softw. Eng.*, 26(2):150–177, 2000.

[13] Object Management Group, Needham, MA, U.S.A. *Unified Modeling Language v2.0*, May 2005. http://www.omg.org/technology/documents/formal/uml.htm.

[14] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Companies, Inc., 5 edition, 2001.

[15] G. Smith. *The Object-Z Specification Language: Advanced in Formal Methods*. Kluwer Academic Publishers, 2000.

[16] J. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, 2 edition, 1992.

[17] R. Valk. Petri Nets as Token Objects: An Introduction to Elementary Object Nets. In *Proc. of the 19th Intl. Conference on Application and Theory of Petri Nets (ICATPN'98)*, pages 1–25, 1998.

Achieving a Better Middleware Design through Formal Modeling and Analysis *

Weixiang Sun, Tianjun Shi, Gonzalo Argote-Garcia, Yi Deng and Xudong He
School of Computing and Information Sciences
Florida International University
Miami, FL 33199
E-mail: {wsun001,tshi01,gargo001,deng,hex}@cis.fiu.edu

Abstract

User-centric Communication Middleware (UCM) is proposed to provide a high-level unified user-centric communication abstraction for upper-layer communication applications by separating and isolating the complexities of network-level communication control and media delivery from the diversity of application-dependent communication logic. Due to the complexity of UCM, it is a major challenge to ensure its correct design and implementation. In this paper we present our approach for designing UCM through formal modeling and analysis based on SAM, a general formal framework for specifying and analyzing software architectures. We present the formal model for UCM using SAM and provide a method to analyze SAM architectural specifications using model checker SPIN.

Keywords *Software Architecture, Formal Methods, Formal Model, Middleware.*

1 Introduction

The domain-specific communication application development is both time-consuming and error-prone because these applications are built on top of low-level network abstractions such as TCP/UDP socket, SIP [3] and RTP [10] APIs that are primitive and often heterogeneous. The application-dependent communication logic is mixed with the network-level communication logic. The portability of the communication applications is reduced by the heterogeneity of the low-level network. A novel User-centric Communication Middleware (UCM) [12] is introduced to separate and isolate the complexities of network-level communication control and media delivery from the diversity of application-dependent communication logic and provide a high-level unified user-centric communication abstraction for upper-layer communication applications. UCM is introduced as a layer of CVM [2]. Through its layered architecture, CVM supports separation of major concerns such as modeling application-dependent communication logic, automatic generation of scripts to drive the communication logic, and the application-independent basic communication service provided by UCM.

To evaluate the feasibility of the UCM concept, we built a prototype in Java for UCM and a simple communication application based on it. We did some experiments and evaluated the performance of UCM. The prototype of UCM, addressing a partial set of requirements, encompasses around 20,000 lines of code and is already quite complex. A complete production quality implementation of UCM will be even much more complicated. To realize a production quality UCM implementation, a rigorous design is necessary. We believe that a formal architecture modeling and analysis approach can produce a high quality design of UCM [1].

SAM [5] is a software architecture framework for specifying software architectures and providing a systematic approach to the development and analysis of software specifications. SAM has been applied to some applications [6, 11]. In this paper we present the formal model for the UCM using SAM and an approach to analyze it using the model checker SPIN [7]. By building the formal model for the UCM, we gained a deeper understanding of the system being specified, and we were able to detect inconsistencies, ambiguities and incompleteness.

The rest of the paper is organized as follows: Section 2 introduces UCM; Section 3 presents the formal model of UCM using SAM; Section 4 presents an approach to analyze the UCM formal model using SPIN; and finally, Section 5 concludes the paper.

2 User-centric Communication Middleware

User-centric Communication Middleware (UCM)[12] provides a high-level unified user-centric communication abstraction which can be shared across communication ap-

*This work is supported in part by NSF under grant HRD-0317692, by NASA under grant NAG 2-1440.

plications design and development. To satisfy the communication needs, UCM should support multiple user sessions. A user session is defined as a communication process that involves a number of participants, who can be added or removed dynamically. Within a user session, each participant can send media to all other session participants. As discussed in [12], UCM should provide four functionalities: (1) Allow a user to login to a signaling server, and to retrieve and modify his/her current contact-list. (2) Support the basic concept of a communication user session. (3) Provide a way for low-level network and system events exposed to the applications, essentially, enabling development of context-aware applications on top of UCM. And, (4) Provide the user with the capability to specify high-level self-management policies to control session and media delivery.

To provide the above functionality, the UCM core should translate a high-level communication task into a series of operations that control and coordinate the underlying networking facilities to deliver media to session participants. Based on the network configurations and conditions, the UCM core coordinates both the control plane (i.e., signaling protocols such as SIP, negotiating the communication) and the data plane (i.e., transport protocols such as RTP, delivering media) according to the requirements of communication tasks. The communication between peer UCMs relies on various communications infrastructures such as signaling servers and media gateways between them.

A unified UCM API [12] is introduced as four functionally sets of operations that reflect the four requirements of the UCM. Below the unified API, the internal architecture of UCM is outlined in Figure 1. It includes the following modules:

UCM Manager: The UCM Manager is responsible for initializing and configuring the UCM. It allows the user to register with the signaling server. It also creates a new Session Manager to handle the new communication session.

Session Manager: A session manager deals with a single user session. A session state includes the call status, the participants, and the media transfer. This module further delegates the tasks to the "Call Processing", "Session Participants", "Media Delivery" and "QoS and Self Management" sub-modules. The Session Participants module keeps the list of participants of this session. The Media Delivery module manages the media transfer by controlling on the "Media Processing and Transmission" module (see below). The Call Processing module controls the call processing logic of a session based on the underlying Signaling module (see below). The QoS and Self Management module assists the Media Delivery module in automatically adapting transmission parameters or modes, seamlessly handling network transitioning, and hiding or reporting network faults.

Media Processing and Transmission: This module is responsible for the processing and transmission/reception of different media using the low-level network service.

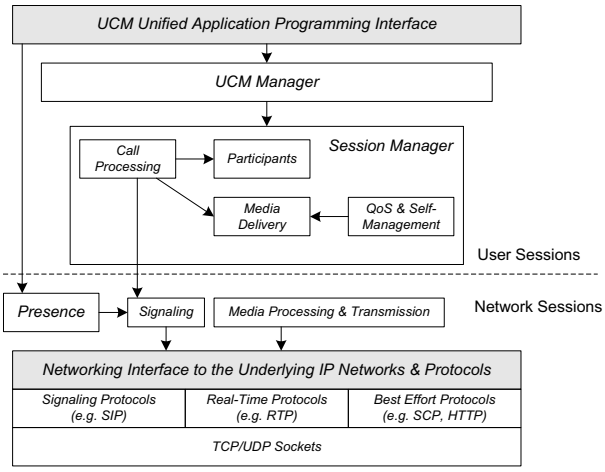


Figure 1. The UCM Architecture

Signaling: The Signaling module carries out the basic signaling operations such as registration, invite/disconnect a user, media type and parameter negotiation.

Presence: A user may need to know whether one contact is present in the system. The user may rely on mechanisms such as SUBSCRIBE/NOTIFY in SIP to ask the registration server to "push" the information to the client-side UCM.

3 UCM Formal Model

In this section, we present the formal model for the UCM using the Software Architecture Model (SAM) formalism. First, a quick introduction on SAM is provided. Then, we present a deployment example of the UCM. Next, the UCM formal model using SAM is defined with one of its components explained in detail. At last, an scenario illustrates the interaction among the components in the UCM model.

3.1 Software Architecture Model (SAM)

SAM [5] is a software architecture framework for specifying and analyzing software architectures. A SAM architecture model consists of a set of compositions, representing different design levels or subsystems. Each composition has of a set of components, connectors and composition constraints, and each component (or connector) is composed of a behavior model and a property specification. A component (or connector) can be refined by defining a mapping to a composition. We see some of these aspects in Figures 2 and 3. Predicate Transition nets (PrT nets)[4] are used to describe the behavior models of components and connectors. An example of a PrT net is shown in Figure 4. Linear Temporal Logic (LTL)[8] is used to specify system properties of components and connectors.

3.2 System Overview

Figure 2 shows an example of a communication deployment modeled in SAM. It involves two client applications, *Application1* (App1) and *Application2* (App2), each one with a copy of the *UCM*. Each *UCM* will take care of the interaction with the *Network* (e.g. media transmission) and the *SIP Server* (signaling). If the client at *App1* needs to locate the client at *App2*, *App1* sends a request to its local *UCM*, which in turn contacts *SIP Server*, to contact *App2* through its *UCM*. For media transmission, *App1* and *App2* contact their respective *UCMs* which contact the *Network* component to support such interaction.

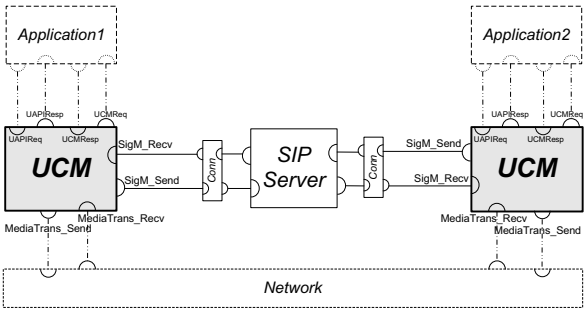


Figure 2. A sample deployment of the UCM.

The deployment in Figure 2 has four types of components: UCM, user's application, SIP Server and Network. UCM connects to SIP Server exchanging signaling messages and UCM interacts with the Network exchanging media transmission messages.

We abstract away details of the client application and by introducing the SIP Server and Network components hide the network layer . We simulate the status of the media transmission without touching the real transmission, since it rather complex and beyond the context of this paper. We explicitly model the SIP server to analyze properties related to the communication between UCMs.

3.3 UCM Architecture

The UCM component in Figure 2 is further refined in Figure 3, which represents the SAM architectural model for the UCM. There are five components in the UCM architecture: UCM Manager (*UCMM*), Presence Manager (*PM*), Session Manager (*SM*), Media Processing & Transmission (*MPT*) and Signaling Manager (*SigM*).

The interaction between components is done through the use of ports and connectors. For example, for component *UCMM* ports *SM_UCMM_in* and *UCMM_SM_out* are used to connect from and to component *SM* through connector *Connector2*. The ports in Figure 3 are mapped to the

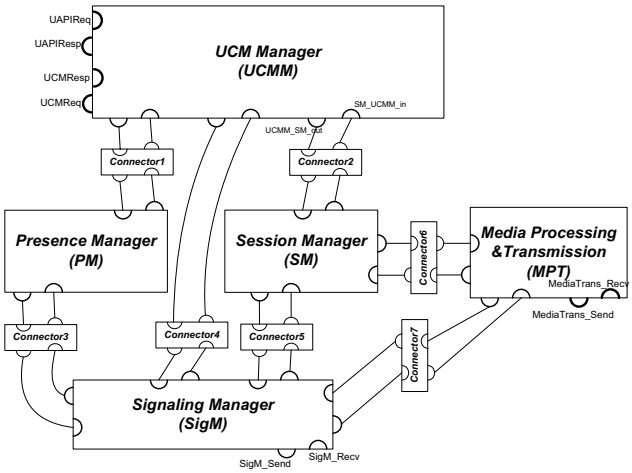


Figure 3. UCM Architecture

ones with the same name in Figure 2. *UCMM* has ports *UAPIReq*, *UAPIResp*, *UCMResp* and *UCMReq*, by which UCM communicates with the user's application in Figure 2. Component *SigM* (signaling), has ports *SigM_Send* and *SigM_Recv* that map to the ports *UCM* uses to interact with *SIP Server* in Figure 2. Finally, component *MPT* provides ports *MediaTrans_Send* and *MediaTrans_Recv* to interact with *Network* component in Figure 2.

3.4 UCM Manager Component

UCM Manager (*UCMM*) component supports the functionality that allows the user to: (1) log in and log out to and from the UCM; (2) add a new contact to the contact list and remove an existing contact from it; (3) create a new session and destroy an existing one.

UCMM is not refined further (it is not a composition), so we specify the behavior model *B* for this component using a PrT net (Figure 4). For this net, the relation φ (types) has to be defined for each place, as well as the mapping *R* (constraints) for each transition. Given that the complete specification of *UCMM* encompasses several pages, we define φ and *R* for places and transitions related to the creation of sessions only. A type is introduced to serve as the generic type for tokens flowing to and from ports in the different components and connectors.

$ReqType = CMD \times CRED \times Session \times USER \times Media \times String \times String \times \wp(USER) \times \wp(Media)$

CMD is a set enumerating all possible message types. *CRED* is the credential of the user (user id and password). *Session* defines the sessions, each session has its local session id, and the name of the initiator of the session and the session id at the initiator side. *USER* is the set of user names. *Media* is the set of media types. The types (φ) for places in the *CMMM* model are:

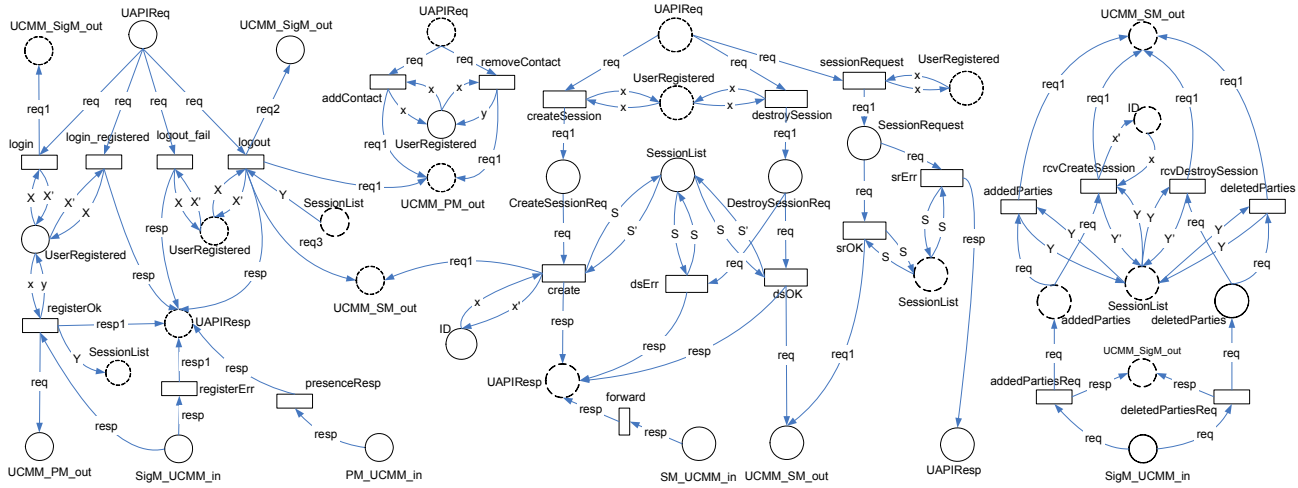


Figure 4. PrT Net Model for UCM Manager component

(a) $\varphi(UCMM\_SigM\_out) = \varphi(SigM\_UCMM\_in) = ReqType$

(b) $\varphi(UAPIReq) = \varphi(UCMM\_PM\_out) = ReqType$

(c) $\varphi(UCMM\_SM\_out) = \varphi(SM\_UCMM\_in) = ReqType$

(d) $\varphi(SessionReq) = \varphi(DestroySessionReq) = \varphi(CreateSessionReq) = ReqType$

(e) $\varphi(UAPIResp) = \varphi(PM\_UCMM\_in) = RespType$

(f) $\varphi(SessionList) = \wp(Session)$

(g) $\varphi(UserRegistered) = CRED \times BOOL$

(h) $\varphi(ID) = INT$

In (a) *UCMM_SigM_out* is a port going from *UCMM* to *SigM*, and *SigM_UCMM_in* is the port coming from *SigM* and going into *UCMM*. In (b) *UAPIReq* is the place defining the port that communicates with the user application. In (c) the two ports in the interaction between *UCMM* and *SM* are defined. In (d) are the places for holding the requests for sessions. In (e) the response type is defined. In (f) *SessionList* place defines the list of sessions. Finally, in (g), the credentials of the current user, and in (h) the type for *ID* are defined. *INT* and *BOOL* define the set of integers and booleans respectively. *RespType* is the same as *ReqType*.

There are two transitions involved in creating a session, *createSession* and *create*, relation *R* is defined for them:

(1) $R(createSession) = (req[1]="createSession") \wedge (x[2]=true) \wedge (req1=req) \wedge (req1[2][1]=x[1][1])$

(2) $R\ (create) = (x'=x+1) \wedge (S'=S \cup \{(x,(x,req[2][1]))\}) \wedge (resp[1]=req[1]) \wedge (resp[6]=x) \wedge (req1[1]=req[1]) \wedge (req1[2]=req[2]) \wedge (req1[3]=(x,(x,req[2][1])))$

Transition *createSession* fires if the request is *"createSession"* and the user is in the system ($x[2]=true$), it creates a copy of the original request (*req1*) but sets the credentials of the user requesting the action. The token is placed in *CreateSessionReq*. Transition *create* fires if there is any token placed at *CreateSessionReq*. It increments a session id counter (x), used to give unique id's to the sessions, and it adds one new session to the session list for the current user (S'). It also forwards a token to *UCMM_SM_out* the out port from *UCMM* to *SM*, and another token to *UAPIResp*.

3.5 Scenario: Adding A New Party

In this section, we describe how adding a new party scenario is realized in the UCM formal model. In this scenario, the application user adds a new party to an existing session. Each session is a communication space shared by several parties, so when a new party is added to session, *UCM* does the session negotiation where the other users in the session are notified that there is a new party added to that session.

The user application user sends the *addParty* request to *UCM* via *UCMReq* port. *UCMM* receives the request and checks if the session included in the request message exists. If the session does not exist, the transition *srErr* in *UCMM* sends the response with the error message. If the session exists, *UCMManager* forwards the request to *SM* via the *UCMM_SM_out* port by the transition *srOK*.

When the Session Manager *SM* receives the request, it asks the Signaling Manager *SigM* to send the signaling message to the other users in the session and the new user. *SM* sends the signaling message to *SM_SigM_in* port in *SM* via *SM_SigM_out* port. The information about the existing media is included in the message. *SigM* encodes the parameters and send via *SigM_Send* port the requests to *SIP Server* shown in Figure 3. *SIP Server* broadcasts the messages to the users connected with it.

SigM of the other users' *UCM* checks if it is the destination of the message. If it is, it decodes the message. Otherwise, it discards the message. When *SigM* receives the *addParty* message, it sends the message to *SigM_UCMM_in*

port in *UCMM* via *SigM_UCMM_out port*. *UCMM* sends the response message via *SigM*. If the user is new to the session, the transition *rcvCreateSession* in the *UCMM* creates a new local session for the request and sends the request to *UCMM_SM_in port* in *SM* via *UCMM_SM_out* port. Otherwise, transition *addedParties* in *UCMM* forwards the request to *SM*.

When *SM* receives the request, if the user is in the session, it updates the party list. Otherwise, it sends medium negotiation messages to the other parties in the existing session via *SigM*.

3.6 Summary of UCM Formal Model

Component	Places	Transitions	Arcs
UCMM	14	24	97
SM	21	56	177
MPT	5	9	46
PM	8	13	52
SigM	10	32	80

Table 1. Number of places, tokens and arcs in each UCM component

The number of places, transitions and arcs for each component is summarized in the table above. The descriptions in Section 3, specially the ones for transition constraints, and from the table, show the great complexity of the UCM. We first had a UCM prototype which we studied to build the UCM formal model based on the UCM API requirements. The initial formal model took 50 person-days to build (five people worked on it for 10 days). The model has been revised 5 times in a five-month period.

4 Analysis

Since the behavior models are modeled in PrT nets, general analysis techniques for Petri Nets, e.g. reachability tree or structural induction techniques can be used to analyze them [5]. However, verification by hand is a tedious work and often error prone. To alleviate this, we presented in [11, 6] how to use automated model checking tools like SMV and STeP. The basic idea was to translate a SAM model into the input models required by those model checking tools, with rules and steps for the translation process to assure the correctness of the translation. However, the translation process was still manual.

In this paper, we further automate the translation process. We provide a modeling editor and a translator named SpinTranslator. The editor is capable of exporting a SAM model in an XML-based format, which extends the Petri Net Markup Language (PNML) [9]. The translator accepts a SAM model in the XML-based format as its input, and automatically translates it to the input language of model checker SPIN [7]. One interesting feature of SPIN is that its input language *Promela* supports the use of embedded C code. Most existing model checkers support only restrictive data types and language constructs. In a SAM model, however, the type of a place or the constraint of a transition can be very complex due to its powerful expressiveness. With restrictive data types and language constructs, the translation of a complex SAM model is not an easy job if not impossible. The embedded C code feature in SPIN was one of the main reasons we choose SPIN over other model checkers.

The translation rules of SpinTranslator are as follows:[1]

1. Each component in SAM is translated to a process in SPIN. For each place in the component, we declare a local variable and a local array in the process. The variable records the number of tokens in the place, and the array stores the value of each token. The transition is defined by a repetition construct *do* in Promela. For each transition in the component, we define an option sequence in the repetition construct with the form:

   $$:: c\_expr\{\ldots\} \rightarrow c\_code\{\ldots\}$$

   The *c_epxr* part is a C expression that decides the enabling of the corresponding transition, and the *c_code* part is a piece of C code that implements the firing of the transition, i.e. updating the input places and the output places of the transition.

2. Ports in SAM are translated to a channel in SPIN. If a port is an input port of a component in SAM, then the corresponding process in SPIN only reads from the corresponding channel. Similarly, if a port is an output port, the process will only write to the channel.

3. For the properties to be verified against the model, we define a never clause in Promela, which includes the LTL formulas representing the properties.

Nondeterminism in SAM is maintained but concurrency is partly lost after the translation. By putting each transition of SAM into an option sequence in a repetition construct of SPIN, one of the enabled transitions is nondeterministically selected to fire. If there are multiple substitutions that enable a transition, one randomly selected substitution is used when the transition is selected to fire. The mechanism above guarantees that nondeterminism is maintained during the translation. Active processes in SPIN run concurrently. However, there is no concurrency within a process. By translating a component in SAM to a process in

[1] The Promela program skeleton is omitted due to the page limit.

SPIN, the inter-component concurrency is maintained, but concurrency inside a component is lost. However, even with the loss of part of the concurrency, the behavior of a PrT net model remains the same based on the definition of PrT nets. Therefore it will not affect the analysis of system properties.

From the description of the "adding a party" scenario, we can see one property of the component *UCMM*: upon receiving a *addParty* request on an existing session from port *UAPIReq*, *UCMM* should be able to forward it to component *SM* via port *UCMM_SM_out*. We checked this property using the above approach in our experiments of SpinTranslator, and proved its correctness. The automatically generated Promela file by SpinTranslator had about 800 lines of code. The *pan* analyzer generated by SPIN needed about 4.8 Kbytes of memory per state. A verification run in full state space search mode returned instantly with 3.3 Mbytes of memory usage, 12 visited states and 16 explored transitions in a search with a maximum depth of 10.

From the results, we can see the effectiveness of model checking by SPIN for a quite complex model like *UCMM*. During our use of SpinTranslator, we found additional benefits: some errors introduced during the modeling of a design such as incompleteness or inconsistence can be detected by the translator. For example, SpinTranslator complains in the case that a place refers to an undefined type or the initial marking of a place does not match the type of the place.

The current version of SpinTranslator allows complex types for places such as structured data types and sets and non-trivial transition constraints with universal or existential quantifiers. However, set operations (e.g. intersection) are yet to be implemented. For now, this type of operations is handled manually by editing the generated Promela code. Also the translation from LTL formulas in SAM to never claims is done by using the conversion tool provided by SPIN.

5 Conclusions

In this paper, we presented a formal model for UCM using SAM and an approach to analyze it by translating the SAM model to Promela and verifying the properties using model checker SPIN. By building the formal model for the UCM, we gained a deeper understanding of UCM. Some incompleteness in the model was also discovered during the process of translation. The analysis approach is demonstrated by the verification of some scenario-based properties in the UCMM model. As mentioned in the paper, the current version of the translator is still preliminary. The improvement of the translator is part of our future work. Furthermore, we plan to thoroughly verify the properties of UCM based on our analysis method. This will help us find and correct the errors at the early stage of development. We believe that by building the formal model for UCM and analyzing the model, we can achieve a precisely described and correct design for UCM.

References

[1] E. M. Clarke and J. M. Wing. Formal methods: state of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[2] Y. Deng, S. M. Sadjadi, P. Clarke, C. Zhang, V. Hristidis, R. Rangaswami, and N. Prabakar. A communication virtual machine. *Technical Report FIU-SCIS-2006-02*, School of Computing and Information Sciences, Florida International University, 2006.

[3] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. Sip: Session initiation protocol. *RFC 2543*, March 1999.

[4] X. He. A formal definition of hierarchical predicate transition nets. In *In Proceedings of the 17th International Conference on Application and Theory of Petri Nets, LNCS 1091*, pages 212–229. Springer-Verlag, 1996.

[5] X. He and Y. Deng. A framework for developing and analyzing software architecture specifications in sam. *The Computer Journal*, vol.45(no.1):111–128, 2002.

[6] X. He, H. Yu, T. Shi, J. Ding, and Y. Deng. Formally analyzing software architectural specifications using SAM. *Journal of Systems and Software*, 71(1-2):11–29, 2004.

[7] G. J. Holzmann. The model checker spin. *IEEE Trans. Softw. Eng.*, 23(5):279–295, 1997.

[8] Z. Manna and A. Pnueli. *Temporal Logic of Reactive and Concurrent Systems*. Springer, 1992.

[9] Petri net markup language. http://www.informatik.huberlin.de/top/pnml.

[10] H. Schulzrinne, S. Casner, R. Frederick, , and V. Jacobson. Rtp: A transport protocol for real-time applications. *RFC 3550*, July 2003.

[11] T. Shi and X. He. Modeling and analyzing the software architecture of a communication protocol using SAM. In *Proceedings of the 3rd IEEE/IFIP Conference on Software Architecture*, pages 63–77, 2002.

[12] C. Zhang, S. M. Sadjadi, W. Sun, R. Rangaswami, and Y. Deng. User-centric communication middleware. *Technical Report FIU-SCIS-2005-11-01*, School of Computing and Information Sciences, Florida International University, 2005.

Design Rationale in Academic Software Development: Requirements for a Representation Model

Debora Maria Barroso Paiva
Andre Pimenta Freire
Renata Pontin de Mattos Fortes

Mathematics and Computer Science Institute
University of São Paulo, Brazil
{debora, apfreire, renata}@icmc.usp.br

Abstract

Design rationale is the description of "how" and "why" an artifact has been designed in a certain way. The recording and management of this information are important issues in achieving better support for software maintenance. We have experimented with design rationale in academic setting and have perceived weak motivation of students with traditional design rationale approaches. In this paper, we introduce a set of requirements for a design rationale model within the context of academic software research.

1 Introduction

With the wide spread of internet many students and researchers are working on collaborative academic projects, in a distributed way. A major research effort has been required in order to develop new technologies to meet the diverse needs of users. This environment has suggested the characterization of new coordinated research approaches, aiming at to burst software quality and explore new mechanisms for evolution of academic software research [1].

The problem that we have perceived is in building research-oriented (not commercial) software that is something that must be maintained over the year, by different students or researchers, according to researches demand. Little academic software has the maintainability property. We have attempted to define an academic software process [2] regarding (1) our experience in using software process in academic setting [3, 4, 5, 6], (2) empirical data that we have gathered about usage of software practices in academic setting and (3) lessons learned presented in literature concerning positive results and difficulties related to software development in academic setting. Our immediate goal is to characterize an initial methodology for development of academic software projects regarding fundamental characteristics of this context. The long-term goal is to help promoting the evolution of academic software projects in a more feasible way.

In our investigations we have regarded the Design Rationale (DR) approach. In general, DR is related to the description of the reasoning behind a project [7, 8]. When system evolution is a goal, as occurs in academic setting, DR takes a particular relevance. If a system is to be evolved efficiently, as much of the specification and artifacts as possible should be reused. Accessing DR when requirements change can be useful because designers are able to understand decisions made during a previous phase [9]. We have experimented traditional DR models [10, 11] in academic setting aiming at to know how students deal with them. We have perceived problems with applying traditional approaches in such environment.

In this paper we introduce a set of requirements for DR representation within the context of academic software research. The model that we have attempted to define from requirements represents an important part of the academic software process that we have characterized. Many conventional DR models were described in literature and their concepts represent the core of our approach. In Section 2 we present an overview concerning DR. In Section 3 we discuss main results of our experience with DR in academic setting. In Section 4 we describe the requirements for a DR model. Finally, we present our conclusions in Section 5.

2 Design Rationale Overview

DR can describe the reasoning that justifies a project and why specific structures were chosen [7, 8]. According to Shipman and McCall [12], three perspectives of DR (argumentation, communication and documentation) use different representation types which influence the ability to capture, retrieve and use DR information. *Argumentation* means that DR is related to the reasoning that designers use in framing and solving problems. Those who use the DR argumentative approach aim to get designers to discuss design within a given representation model, such as IBIS (*Issue-Based Information Systems*) [10] and PHI (*Procedural Hierarchy of Issues*) [11]. The *communication* perspective is related to capturing and retrieving natural communication among members of a project team (audio, video, email, diagrams, etc). The *documentation* perspective means that DR should be the information record (what, when, who and why) about design decisions.

Overall, the majority of representation models share the same structure of "design problem", "alternative" and "argumentation". They are generic purposes and can be adapted for different disciplines. Particularly, the Potts and Bruns model [13] was specially developed for software development. Its main difference is that each *issue* is derived from an *artifact*, i.e., Potts and Bruns model is an attempt to delineate generic elements of software development and software DR, such as artifacts, issues, positions, justifications and the relations among them. This model was used as base of our work, but we have specialized it for the academy-oriented approach that we have practised.

3 Experiences with Design Rationale in Academic Setting

In the last years, some academic software projects have been developed in our department regarding capture and register of DR. We have asked students to express their feelings about DR usage in terms of reuse, maintenance and documentation of projects. Additionally, we have defined and used some metrics aiming to measure the effects of DR more objectively.

We carried out case studies in six disciplines. One hundred twenty four students were involved at whole. IBIS and PHI models were used (they are the most basic models and have influenced definition of other ones). In general, students were asked for capturing and registering DR for projects developed in groups. Maintenance activities were emphasized subsequently and we could observe the effect of DR in this sense.

After creation of an initial DR base by students, we attempted to evaluate if students were effectively registering information gathered from discussion and decision made during development stage. DR can not be useful if it is not conveniently recorded. We counted, therefore, the number of issues, positions and arguments registered in four phases of the projects developed in the context of the six disciplines. At the whole, 31 projects were evaluated. In Table 1 we present the mean values representing how much information related to each argumentation item was registered. For example, in the first line of the table we observe that, on average, 2.41 issues were recorded during analysis requirement phase for each project. For *each* issue, 1.16 positions were generated and, for *each* position, 0.67 for and against arguments were recorded on average.

Table 1. Measurement of DR information registered in different phases of 31 academic software projects - mean values.

Phases	*Issues*	*Positions*	*Arguments*
Requirements Analysis	2.41	1.16	0.67
Design	4.61	1.32	0.73
Implementation	1.8	0.91	0.84
Test	0.94	1.17	0.25

Due to space limitation and for simplicity, we will not discuss these values extensively. It is notable, however, that little DR information was effectively captured and registered by students in case studies. We observed that, in practice, students had difficulty with organizing information according to the conventional argumentation structure. We strongly believe that the minor interest of students in registering DR was due to:

* Students are not experienced software engineers. Software documentation is not a familiar task for them. Producing an elaborated argumentation network, categorizing and linking DR information, was not a trivial activity for novice developers.

* In traditional models some nodes and edges, for example *issues* and *arguments*, do not reflect the software engineering process. It is fundamental to reduce the cognitive overhead of models and tools on designers in general, i.e., high-cost models should be avoided.

As a result of our experiences with DR in academic setting, we elaborated a set of requirements for a model supporting DR representation. We emphasized the key points that we believe to be useful for motivating DR capture in academic setting and, consequently, contributing for academic software evolution.

4 Requirements for an Academy-Oriented Representation Model for Design Rationale

We have identified a set of requirements for an academy-oriented DR model. It is important to notice that our practical experience was a crucial component in determining these requirements. Overall, we have observed that an argumentation model should be composed by a simple structure and that students are not completely prepared to provide a large amount of information. Following, we present the set of requirements that we have defined.

1. Focus on objective of each process phase: capture of different types of information, according to the objective of each process phase should be feasible. In a previous study we have observed that DR is specially important in requirements analysis phase [6]. This result, in general, confirms what other authors have observed in their studies and experiments [14, 15]. Therefore, we have focused on such phase. The "what" and "why" for requirements should be reinforced.

2. Traceability: association between system requirements and DR should be feasible. By doing such association it is possible to visualize later what decisions were made for each requirement and to recover additional information about them.

3. Commentaries inclusion: inclusion of opinions, experiences and lessons learned should be feasible. Students should be allowed to include information beyond that required in the model and not be restricted to the provided structure. In academic setting, students are motivated to reasoning, reflection, experimentation and discussion. We are now asking students and researchers to register the valuable background knowledge acquired.

4. Representation of element generating the discussion: inclusion of "questions" or "problems" as initial elements generating a discussion should be feasible. The objective is to motivate students for capturing DR, providing simple structures and avoiding that the format of a discussion element be changed to adapt to the model.

5. Design rationale evolution: evolution of DR should be feasible. It should be possible to associate the DR information with versions of artifacts. In terms of software development, maintenance activity is very intensive and we want to distinguish clearly what decisions were made in each version of the project. We believe that the relationship among DR and system versions can bring many benefits. For example, novice members can understand, step by step, how a system has evolved. Another benefit is that documentation concerning what decisions are valid for each version is generated, i.e., the traceability requirement is reinforced. Certainly, communication about important issues of systems can be improved by adopting the relationship between its versions and DR.

6. Context information: information related to the context in which the design rationale was elaborated should be captured. This includes the who, when (date/time), where (in which project/version) information. This information is important when recovering the project history.

Another key requirement is related to agility in registering design rationale. According to Regli et al [16], one challenge is to reduce the cost for registering design rationale and to guarantee that its organization allows later recuperation. We have verified how agile software principles[1] could be applied in the context of design rationale. The goal is to provide a simple model for design rationale and motivate students for its usage. Following, we indicate how some elementary characteristics of agile methodology were considered in the context of the requirements for the representation model.

1. Early and continuous delivery of valuable software: we are worried about registering "small grains" of valuable design rationale information since early versions of the process. We suggest that such information be continuously captured when the software is maintained.

2. Build projects around motivated individuals (give them the environment and support they need): activities to maintain and redesign an artifact require much effort to understand the previous work. Many believe that keeping track of the design rationale will provide a great aid to designers. Design rationale systems can be used as a basis to discuss and reason among collaborating designers [16]. We are developing a specialized tool, called DocRationale [17], that promotes the capture, storage and retrieval of design rationale for software artifacts. DocRationale is under development, regarding the help of ubiquitous computing mechanisms to capture design rationale information in a non-intrusive way [18].

3. Working software is the primary measure of progress: useful design rationale is also an important measure of progress. As stated by Burge, such information can only be valuable if it is accessed to help designers making decisions or to solve some design problem [19]. We are not interested in registering a large amount of information. As stated, we have attempted to understand in which phases design rationale could be useful and what representation structure was expected by students. We have motivated the capture of design rationale only in phases in that we have observed its usefulness.

[1] http://agilemanifesto.org/

4. Simplicity: in our purpose, the rationale format has been kept simple in order to lessen the burden on the students. We suggest the register of alternatives that were discussed, however, only the one selected for implementation is really justified or described. Differently from traditional models described in literature, we avoid the register of favor and against arguments for alternatives that were not experimented by students. Another characteristic in favour of simplicity is the register of decisions made. Designers using the most traditional models need to search the decision on other artifacts, in general the code. A more simple solution is to indicate the decision more directly, together with discussion.

5. Straightforward: the representation model should be easy to understand and well documented. We have described each component of the model and to indicate clearly what information should be provided by students (the representation model is being documented).

5 Concluding Remarks

The main contribution presented in this paper is the set of requirements for an academy-oriented DR model. It was defined regarding our experience in applying DR in academic projects. We point out that this is a special characteristic of our approach: we carefully collected requirements by means of experiences involving final users.

At the time of writing, the model is in the process of being integrated with DocRationale tool [17], [18]. By using DocRationale we intend to evaluate the requirements and the model.

In our approach, we are not suggesting the register of DR in all phases of the development process. We have noted that developers, as well as students and researchers, are essentially motivated for registering discussion and decisions in preliminary phases of software processes. Indeed, usefulness of DR have been perceived for these phases. As further work, we intend to investigate DR appropriateness for other phases, for example, software tests.

Acknowledgements. The authors would like to thank to FINEP, FAPESP, CNPq and CAPES for funding this research.

References

[1] O. Chirouze, D. Cleary, and G. G. Mitchell. A Software Methodology for Applied Research: eXtreme Researching. *Software – Practice and Experience*, 35(15):1441–1454, 2005.

[2] D. M. B. Paiva and R. P. M. Fortes. Model for Academic Software Development Including Design Rationale Elements. In *I IFIP Academy on the State of Software Theory and Practice – PhD Colloquium*, 2005.

[3] R. P. M. Fortes, A. P. Freire, V. H. Vieira, and D. M. B. Paiva. An Academic Web-Based Agenda and its Engineering Process. In *VII Workshop Ibero Americano de Ingeniería de Requisitos Y Desarrollo de Ambientes de Software*, pages 151–156, 2004.

[4] D. M. B. Paiva, A. P. Freire, R. Sanches, and R. P. M. Fortes. Defining, Deploying and Improving Software Processes in Academic Setting. In *VI International Symposium on Software Process Improvement*, pages 71–82, 2004.

[5] D. M. B. Paiva, A. P. Freire, and R. P. M. Fortes. Web Engineering Process - a Case Study from Academic Development. Technical Report 79, 18p., University of Sao Paulo, August 2004.

[6] D. M. B. Paiva and R. P. M. Fortes. Design Rationale in Software Enginnering: A Case Study. In *Seventeenth International Conference on Software Engineering and Knowledge Engineering*, pages 342–348, 2005.

[7] T. R. Gruber and D. M. Russel. Design Knowledge and Design Rationale: A Framework for Representation, Capture, and Use. Technical Report KSL 90-45, Knowledge Systems Laboratory, Standford, California, August 1991. 40p.

[8] T. P. Moran and J. M. Carroll. *Design Rationale: Concepts, Techniques, and Use Computers, Cognition, and Work*. Lawrence Erlbaum Associates, New Jersey, 1996. 659 pages.

[9] S. Monk, I. Sommerville, J. M. Pendaries, and B. Durin. Supporting Design Rationale For System Evoluation. In *Proceedings of the Fifth European Software Engineering Conference*, 1995.

[10] W. Kunz and W. Rittel. Issues as Elements of Information Systems. *Working paper 131*, August 1970. Center for Planning and Development Research, University of California, Berkeley.

[11] R. J. McCall. PHI: A Conceptual Foundation for Design Hypermedia. *Design Studies*, 12(1):30–41, 1991.

[12] F. Shipman and R. McCall. Integrating Different Perspectives on Design Rationale: Supporting the Emergence of Design Rationale from Design Communication. *Artificial Intelligence in Engineering Design, Analysis, and Manufacturing*, 11(2):141–154, Abril 1997.

[13] C. Potts and G. Bruns. Recording the Reasons for Design Decisions. In *Proceedings of the International Conference on Software Engineering*, pages 418–427. IEEE CS Press, 1988.

[14] J. Conklin and K. Burgess-Yakemovic. A Process-Oriented Approach to Design Rationale. In *Design Rationale Concepts, Techniques and Use*, pages 293–428. T. Moran and J. Carrol (editors), Lawrence Erlbaum Associates, 1996.

[15] L. Karsenty. An Empirical Evaluation of Design Rationale Documents. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI'96)*, pages 13–18, Vancouver, BC, April 1996.

[16] W. C. Regli, X. Hu, M. Atwood, and W. Sun. A Survey of Design Rationale Systems: Approaches, Representation, Capture and Retrieval. *Engineering with Computers: An International Journal for Simulation-Based Engineering, special issue on Computer Aided Engineering in Honor of Professor Steven J. Fenves*, 16:209–235, 2000.

[17] S. D. Francisco, C. A. Izeki, D. M. B. Paiva, and R. P. M. Fortes. A System for Supporting Design Rationale of Software Artifacts. In *XXIX Latin-American Conference on Informatics*, 2003. In portuguese.

[18] S. M. A. Lara and R. P. M. Fortes. Supporting Informal Design Rationale Capture for Requirements Specification. In *2nd Latin American Web Congress and the 10th Brazilian Symposium on Multimedia and the Web*, pages 65–68, 2004.

[19] J. Burge and D. C. Brown. Reasoning With Design Rationale. In *Artificial Intelligence Design*, pages 611–629. Kluwer Academic Publishers, 2000.

Coverage Testing Embedded Software on Symbian/OMAP*

W. Eric Wong, Sharath Rao
Department of Computer Science
University of Texas at Dallas
Richardson, TX 75083

John Linn, James Overturf
Systems & Software Lab
Texas Instruments
Dallas, Texas 75251

Abstract

Availability of cheaper memory and faster processors has led to an increase in size, capability, and complexity of software available on embedded devices. Code-based coverage testing is a method to determine the quality of software testing. It can also be used to quantitatively monitor the progress of testing. Studies have shown that the quality of software can be improved with an increase of its code being tested. However, techniques used for coverage testing software on Windows and/or UNIX/Linux platforms in general cannot be applied, without appropriate modification, to coverage testing embedded software. This is because testing embedded software is restricted by various factors such as the environment of the target device, the limited memory available on the target, and device control dependencies. We propose a solution to overcome these problems for software running on the Texas Instruments Open Multimedia Application Platform (OMAP)-based devices and the Symbian operating system. A tool, TestingEmbedded, on top of χSuds/ATAC is developed to automate our testing procedure. A case study on five components of the multimedia framework of Symbian OS used by many cell phone manufacturers is reported. In addition to measuring code coverage, we also use the coverage information to provide useful hints for efficient test generation to increase the coverage of the program being tested effectively. We also use coverage as a filter to show how the number of test cases can be reduced significantly without sacrificing the overall code coverage. Test cases in the reduced subset have a higher priority to be executed for revalidating the program during the regression testing.

Key words: coverage testing, embedded software, Symbian, OMAP

1. Introduction

The advent of faster processors and cheaper memory has facilitated the growth of embedded systems from performing simple tasks to performing a plethora of tasks. The complexity of the tasks performed by these systems has also increased significantly. A good example of such a changing device [1] is a mobile phone which can be now equipped with cameras and perform multiple functions like playing music, recording video, email, etc. along with the normal operations of a phone. These complex devices are controlled by complicated programs. Hence, it is very important to have good testing of the software that runs on these devices because such software is embedded into the devices which are then mass produced. Any critical error in the software not caught by testing will cause erroneous functioning of the software and also render the device unusable.

Code-based coverage testing is a method to determine the quality of software testing [7,21]. It can also be used to quantitatively monitor the progress of testing. Studies have shown that the quality of software can be improved with an increase of its code being tested, and also the number of defects detected by a set of test cases has a higher correlation to the percentage of code covered by this test set than to the number of test cases in the set [21,22,23]. However, techniques used for coverage testing software on Windows and/or UNIX/Linux platforms in general cannot be applied, without appropriate modification, to coverage testing embedded software. This is because testing embedded software is restricted by various factors such as the embedded target environment, the limited memory available on the target, and device control dependencies [9,10].

There are several important questions which have to be answered before we can measure code coverage for embedded software. How should the software be instrumented? Should the same technique used for instrumenting software running on Windows and/or UNIX/Linux be used for software running on an embedded target board? How can the execution trace be collected efficiently on the target without suffering from a memory overflow? Note that the memory on an embedded system can be very limited, which is different from a desktop PC. This suggests that different instrumentation and trace collection techniques have to be adopted for embedded software to overcome this memory constraint problem. Also, how to transmit the collected trace from a target to a host for subsequent coverage analysis is another concern that should be addressed? How can we guarantee no trace data are lost either during the transmission or due to the memory overflow on the target device? How often should the host receive trace data? Is it after the execution of each probe, each test case, a set of test cases defined by the user on the fly, or only when the entire test session is completed? Also, we need to consider how to integrate trace data transmitted/received at different times.

In this paper, we propose a solution to overcome the above problems. A tool, TestingEmbedded, on top of χSuds/ATAC [8,30] is developed to automate the testing procedure. We have also applied our solution to five DLLs (Dynamic Link Library) such as the MP3 decoder. Each library is part of the multimedia framework present in the Symbian OS running on the OMAP platform [11] which is used by many cell phone manufacturers. In addition, we performed test set minimization to show how the number of test cases can be reduced significantly without sacrificing the code coverage, i.e., we can identify a small subset of test cases which gives the same code coverage as the entire test set.

2. An Overview of Symbian and OMAP

2.1. The Symbian Operating System and Symbian C++

Symbian OS is an open, standard operating system designed for the specific requirements of data enabled 2G, 2.5G, and 3G mobile phones. It is based on the idea of small mobile devices that cater to a mass market with intermittent wireless connectivity on an open platform for a range of products [13]. It includes a robust multitasking kernel, integrated telephony support, communications support and a graphical user interface framework along with a variety of applications engines. The operating system is built considering the

* This paper is funded by a research grant from Texas Instruments.

[1] In this paper, we use "device" and "embedded system" interchangeably. We also use "program" and "software" interchangeably. In addition, "a block is tested," "a block is executed," and "a block is covered" all have the same meaning. The same applies to "faults," "program bugs," and "defects."

severe limitations imposed by such small devices. These devices are small and mobile but place a very stringent requirement on the applications and also on the operating system that runs on them. These reasons make the Symbian OS distinct not only from PC or workstation-based operating systems but also from other embedded systems that are not designed with these premises.

The code developed for the Symbian OS is grouped into two major packages. An .exe is a program with a single entry point and a dynamic link library (a.k.a. DLL) is a library of program code with potentially many entry points. Another important difference is that Symbian OS does not allow DLLs to contain writable static data while an .exe can contain them. This has an impact on our proposed solution in Section 3.2. The operating system implements a preemptive multitasking so multiple applications can run simultaneously. More details of the Symbian operating system can be found in [3,16,17].

Symbian C++ differs from the standard ANSI C++ in many ways. On one hand, ANSI C++ includes features not required by the Symbian OS such as private and multiple inheritances [18]. On the other hand, Symbian C++ provides basic types such as Tint, TBool, TAny, etc. which are guaranteed to be the same irrespective of the implementation, and a new construct called descriptors for storing and manipulating both strings and general binary data regardless of the type of memory they reside. For exception handling, Symbian C++ has its own mechanism. Instead of throwing an object when an exception occurs, the execution returns to the trap harness with a 32 bit error code. The operating system has special support for processor level exceptions such as *divide by zero* and these are handled through classes provided by the operating system. Symbian C++ also provides a cleanup stack to handle memory leaks. More specifically, it consists of two phase constructors to handle memory leaks. The first phase of construction uses a normal C++ constructor, but this does not have any code that might exit. The second phase constructor contains the code with the leave and handles memory leaks utilizing the cleanup stack. Nevertheless, Symbian OS provides a rich set of APIs for the users, alleviating the problems brought by the deviation of Symbian C++ from the standard. More details of Symbian C++ can be found in [5,15,18].

2.2. The OMAP Platform

Texas Instruments Open Multimedia Applications Platform (OMAP) [11] is comprised of high performance power efficient processors with dynamic software applications for various 2G and 3G wireless handsets and other multimedia enabled devices. In our study, we use the OMAP 1710 platform, which is a single chip application processor that supports all cellular standards. It is capable of running the Symbian operating system and provides a range of software and hardware accelerators for various multimedia applications such as video and audio decode. The OMAP 1710 also comes with a software development platform which is a multiple board system consisting of a processor daughter card, a main board, an user interface board, a multiport debug board, and interface modules. The software development platform enables users to develop, test, and evaluate software for devices based on the OMAP platform.

3. Measuring and Improving Code Coverage

In this section, we first explain the two coverage criteria used in our study followed by our solution in response to the problems for coverage testing embedded software raised in Section 1. Then, we discuss how coverage information can be used to provide useful hints for test generation.

3.1. The block and Decision Coverage Criteria

Different from functional testing which only emphasizes testing the functionality of the code, coverage testing takes into account the structure of the software being tested and measures how thoroughly the code has been tested with respect to certain coverage criteria such as the block coverage criterion and the decision coverage criterion. The block coverage criterion measures how many blocks in a program are tested, whereas a block (or a basic block) is a sequence of consecutive statements or expressions containing no transfers of control except at the end, so that if one element of it is executed, all are. This of course assumes that the underlying hardware does not fail during the execution of a block. A test set which gives 100% block coverage for a program also gives 100% statement coverage with respect to the same program. However, the block coverage criterion requires less instrumentation than the statement coverage criterion which in turn has a smaller runtime and memory overhead. Similarly, the decision coverage criterion measures how many different branches of the decisions are tested. For a *compound* decision such as "if (a < b) and (c > d)," we treat it as two *simple* decisions and our decision coverage criterion requires test cases to cover the *true* and *false* branches of the simple decision "a < b" and the *true* and *false* branches of the simple decision "c > d." This criterion may require more test cases than the one which treats "if (a < b) and (c > d)" as one decision and only requires test cases to cover the *true* and *false* branches of the entire decision. The advantage of using the former is that test cases so generated may be able to reveal program bugs that cannot be detected by test cases generated based on the latter.

3.2. Measuring Code Coverage

We have had successful experience using χSuds/ATAC developed at Telcordia (formerly Bellcore) [8,30] to measure coverage for C and C++ programs running on the Windows and Unix/Linux platforms. More specifically, measurement of code coverage is a three step process involving instrumentation, execution, and analysis. The input for instrumentation is a preprocessed source file. A preprocessor converts the source code by expanding all the various macros and includes the header files present in the source code. Instrumentation inserts probes into the preprocessed code at various appropriate locations. Once the instrumentation is completed, the instrumented source is compiled, linked, and an instrumented executable which is ready for execution is created. We can run this executable against a set of test cases. During the execution, the probes inserted earlier provide information to a runtime library for creating an execution trace file which saves the execution traceability. Such information is used for subsequent coverage analysis. In addition to reporting the coverage summary, χSuds/ATAC also visually displays the coverage analysis on the original non-instrumented source code via a user friendly graphical interface (refer to Figure 4).

As explained earlier, coverage testing embedded software is restricted by various factors such as the embedded target environment, the limited memory available on the target, and device control dependencies. The χSuds/ATAC technique cannot be directly applied to coverage testing embedded software on OMAP running the Symbian operating system. Below we explain the problems that need to be resolved and our solution.

Due to the limitations imposed by Symbian/OMAP, the code instrumented by χSuds/ATAC cannot be directly used on our targeted platform. Note that the process of instrumentation inserts probes and other information required during the runtime collection of data into the source code. The information is stored as static structures. Each function present in the source code is represented by a set of structures containing information about the number of blocks, decisions, the occurrences of these blocks and decisions, etc. However, the programs used in our study are DLLs such as the MP3 decoder. Each DLL is part of the multimedia framework present in the Symbian OS. The operating system does not allow these programs to contain writable static data.

Our first attempt to solve this problem was to use Thread Local Storage (TLS) [6,14]. The Symbian OS associates a single machine word of writable static data per thread with each DLL. The TLS pointer can act as a global variable within a DLL. Since the space provided for storing the TLS is just a single word, the entire set of

static structures cannot be stored in that memory space. In order to overcome this difficulty, the set of static structures has to be encapsulated as members inside a class (for discussion purposes, let's call it α which is not part of the original DLL) and a pointer to class α is stored in the TLS. This approach does not succeed for the following reasons. The presence of a large of number of functions in each original member class of the DLL greatly increases the sets of static structures. As a result of the large number of static sets, the number of members in class α, and consequently the constructor used to initialize the members of class α, becomes bloated. The instrumented source with the bloated constructer cannot be compiled, eventually leading to failure of the build process. Also, even if the number is low enough to allow the build to be successful, the second problem involves the changes required to be made to the χSuds/ATAC runtime library which also contains global static data. These static variables cannot be stored as members of the same class containing the static structures for the instrumented source (namely, class α in our example), thereby warranting the use of another class (say β) and a corresponding TLS for this runtime library. The TLS provided for each DLL can store only a single machine word, and only one pointer to a class can be stored in this space. Therefore a conflict arises between the two classes (i.e., α and β in our example), which leads to the failure of the build. Also, the program under test might itself be using the TLS already and thus the single machine word is already occupied causing the approach to fail.

Although the Symbian operating system does not allow writable static data to be present in DLLs, it does allow it in the executable or the .exe programs [17,19]. This laxity leads us to another approach by creating an executable which contains all the different sets of static structures and also the ATAC runtime library. However, even if we can succeed in isolating all the static structures into an executable, the problem of communicating between the probes and these structures still exists. To solve this problem, we make use of a client/server paradigm. Whenever a probe invokes the ATAC runtime library, instead of calling the library we modify this probe to send a message to a server. The server will then process the message and make an appropriate call to the ATAC runtime library creating the trace file with pertinent coverage information. Stated differently, each customized probe in the revised instrumented source file acts as a client which sends a message with apt information to the server. The server then processes this message and generates the trace file using the contents of the message. The problems for this approach stem from the fact that the programs under test are part of the Symbian operating system and not a top level application. For a client residing inside the kernel to send a message to the server in the application level, the message has to pass through process boundaries. This is, however, not allowed by the Symbian operating system. In addition, the code in each client consists of functions to connect to the server and then format and send messages. These functions when present in the low level components (such as the ones used in our study) that are loaded during the operating system boot up will try to connect the server which is not yet loaded by the operating system. More specifically, the client is loaded but the server is not. This causes a critical failure in the operating system.

Another approach to solve the communication problem described above is to make use of a feature present in the OMAP 1710 development platform. This platform consists of a multi-port debug board. The Symbian OS provides us with a debug class, *RDebug*, which includes a host of methods of which the most pertinent is *Print*(). The *RDebug::Print*() method can send string messages to a debug port which is connected to a host machine (such as a PC) using a serial port (as shown in Figure 1) and the messages can be viewed through a serial port monitor. We take advantage of this facility. In other words, we use an idea similar to the client/server approach, but instead of sending a message to the server, the revised probe sends the message to a host machine using the debugger channel (i.e., via *RDebug::Print*()). The messages sent to the host machine are saved in a log file and processed later on to generate the trace file. Since the program being tested or some other applications running on the platform may also use *RDebug::Print*() to send their debug messages, we need to distinguish such debug messages from those used for trace collection. This is done by having a special prefix as part of the messages in the latter case.

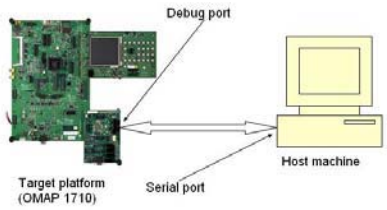


Figure 1. Communication between the debug port and the host machine

A tool, TestingEmbedded, was developed on top of χSuds/ATAC to automate the above described procedure. Below we summarize our solution. Interested readers can find more details in [12].

The TestingEmbedded tool reads a project specification (an *.mmp*) file to identify the various source files that need to be instrumented. The instrumentation has two steps. In the first step, source files are instrumented by using χSuds/ATAC. This creates an instrumented version of the source files and other files (such as *.atac* files) for subsequent coverage analysis. In the second step, for each instrumented file, a corresponding temporary file on the host machine is created to store the static structures introduced during the instrumentation. The probes are replaced by customized probes and the original probes are saved for later use.

The revised instrumented source (i.e., the instrumented version of the source files after the second step) is ready to be built for the target platform. The build follows the normal Symbian build process which creates an application of the type specified in the TARGET of the *.mmp* file. In our case, we have five DLL components of the multimedia framework built for testing. We use the Automatic Testing Framework (ATF) developed at Texas Instruments to test these components against "real life" test cases. More specifically, these test cases are the "real" test cases used for testing these five components by the testers at Texas Instruments. During the test execution, the revised probes use *RDebug::Print*() to send messages to the serial port on a host machine connected to the debugger port of OMAP 1710. The messages received on the host machine are saved in a log file and used by the TestingEmbedded tool to identify the corresponding original probes. Note that each message can uniquely identify an original probe. These (original) probes are then executed in the order of the messages presented in the log file (that is also the order of the customized probes executed on the target platform). The execution invokes the ATAC runtime library which makes use of the previously saved static structures to generate the trace file. These trace files along with the *.atac* files generated at the first step of instrumentation and the original non-instrumented source files are used in conjunction with χSuds/ATAC for coverage analysis and source code display (such as the one in Figure 4). Figure 2 gives a pictorial representation of the procedure described above.

3.3. Improving Code Coverage

Similar to our previous studies [27], we can analyze the coverage information and provide useful hints to help testers generate additional test cases to increase the coverage in an effective way. Without losing generality, we use the pseudo control flow graph in Figure 3 to illustrate how uncovered blocks of a program can be assigned with different weights computed by using a dominator analysis [1]. A similar technique also applied to uncovered decisions. In this figure, each node represents a block. The loops are given implicitly by reusing the name of a block. For example, the transition from block 6 back to block 1 indicates a loop**.** The objective is to generate a test case to cover one of the uncovered blocks with the highest weight

(i.e., one of the hot spots), if possible, before other uncovered blocks with a lower weight are covered so that the more block coverage can be added in each single execution.

A block $\mathcal{A}$ dominates a block $\mathcal{B}$ if covering $\mathcal{B}$ implies the coverage of $\mathcal{A}$, that is, a test execution cannot reach block $\mathcal{B}$ without going through block $\mathcal{A}$. The weight of a given block is the number of blocks that have not been covered but will be if that block is covered. For example, to arrive at block 18 in Figure 3 requires the execution also to go through blocks 1, 2, 4, 7, 12, and 13. This implies block 18 is dominated by blocks 1, 2, 4, 7, 12, and 13. It also means blocks 1, 2, 4, 7, 12, and 13 will be covered (if they haven't been) by a test execution if that execution covers block 18. Similarly, arriving at block 6 in Figure 3 requires only going through blocks 1 and 2, i.e., block 6 is dominated only by blocks 1 and 2. Consequently, blocks 1 and 2 will be covered (if they haven't been) by a test execution which covers block 6. Assuming none of the blocks is covered so far, we say that block 18 has a weight of at least 7 because covering it will increase the coverage by at least seven blocks, and block 6 only has a weight of 3 since its coverage only guarantees an increase of coverage by at least three blocks. A very important note is that while assigning the weight to a given block, we use a conservative approach by counting only the blocks that will definitely be covered because of the coverage of this given block. This is why we use the phrase "at least" in the previous statements. Of course, covering a block of a weight 7 (like block 18 in our example) may end up covering more than seven blocks (e.g., the execution can go through blocks 1, 2, 6, 1, 2, 4, 7, 12, 13 before it reaches block 18). Nevertheless, the additional blocks (block 6 in our example) are not guaranteed to be covered. Since block 18 has a higher weight than block 6, we say that block 18 has a higher priority to be covered than block 6. This also implies that tests that cover block 18 have a higher priority to be executed than tests that only cover block 6.

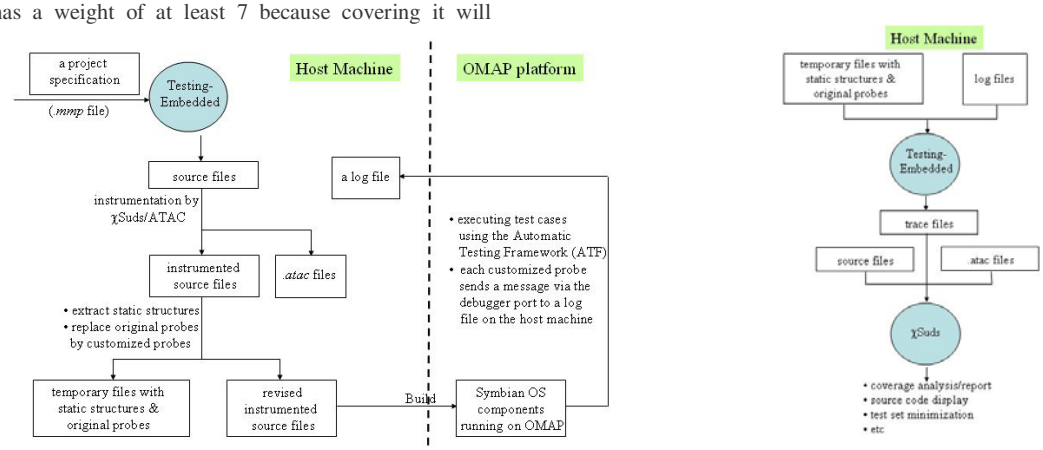


Figure 2. Code coverage analysis using TestingEmbedded and χSuds/ATAC

The execution of certain tests may change the weights of blocks that are not covered by these tests. For example, after the execution of a test (say t_α) that covers block 18 (which also guarantees the coverage of blocks 1, 2, 4, 7, 12 and 13), the execution of another test (say t_β) to cover block 6 will increase the coverage by only one block (namely, block 6 itself). This is because both blocks 1 and 2 have already been covered by test t_α and the execution of other tests (such as t_β) will not change this fact. Under this scenario, block 6 will have the same weight as block 3 even though the former has a higher weight than the latter before test t_α is executed. Hence, tests that cover block 6 now have the same priority as those that cover block 3 (assuming that these tests are designed to cover block 6 or block 3). This also implies that after a test (t_α in our case) is executed to cover block 18, the priority, in terms of increasing the coverage, of executing another test (t_β in our case) to cover block 6 is reduced. This procedure is applied recursively after the execution of each test to recompute the weight of each block that has not been covered by the current tests, as well as the priority of tests to be executed for covering such blocks.

Figure 4 gives an example of covered and uncovered blocks displayed by χSuds/ATAC. The color spectrum at the top of the window indicates the minimum number of blocks that will be covered for each weight. From this figure, code in white has already been covered by a test case and covering it again will not add new coverage, whereas code in red has the highest weight. In our example, covering these hot spots (i.e., code highlighted in red) guarantees the execution of at least eight additional blocks. Since weights in Figure 4 are displayed in different colors, it is better viewed in color.

4. A Case Study

We applied our solution as described in Section 3 to five DLL components from a multimedia framework. Overall, this framework supports audio/video recording and playback, audio/video streaming, and other features for image processing. An important point worth noting is that these components are part of the Symbian OS and not a top level application. We first measured the block and the decision coverage of each component against a set of "real life" test cases used by testers at Texas Instruments. Then, we used the coverage information to provide useful hints for efficient test generation to increase the coverage effectively. We also used such information to identify *unused* code in each component. Finally, we demonstrated how the number of test cases can be reduced with coverage as a filter.

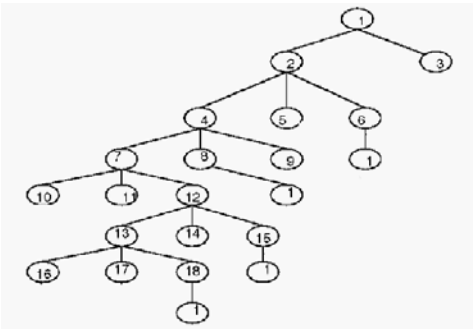


Figure 3. A sample control flow graph

The five components in our study have 584, 811, 678, 910, and 598 blocks, and 338, 546, 401, 408, and 326 decisions, respectively.

The number of test cases used for testing each component is 573, 320, 375, 490, and 585. Since the exact coverage measurements of these components are proprietary to Texas Instruments, we are not allowed to report them here. Nevertheless, we can present some sanitized data such as the block coverage of some test cases for Component 1 (refer to Figure 5). The coverage of test case 1 and test case 4 is low because they only test the initialization or the construction of the component and there are just a few blocks in the constructor.

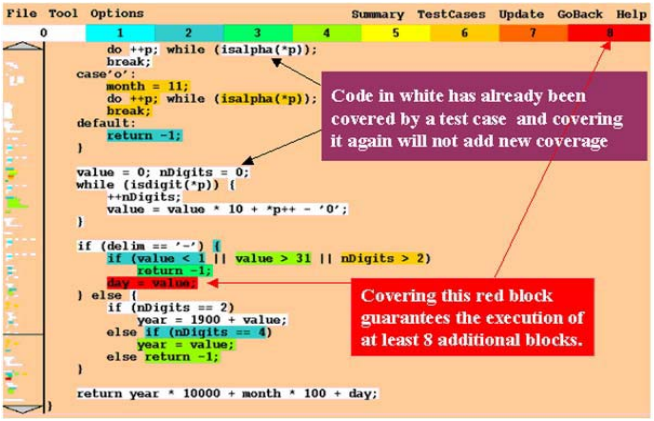


Figure 4. Source code display by χSuds/ATAC

Following the analysis discussed in Section 3.3, we can generate additional test cases to cover the "hot spots." For example, 5 test cases were generated for Component 1 to increase its block coverage by 3.6%. Note that this coverage improvement was accomplished after Component 1 was executed against 573 test cases and reached a saturation point with respect to the block coverage. In most cases, once a program reaches this stage, it is very difficult to further improve its coverage because additional test cases are likely to execute blocks which are already covered by previous tests. Although the approach discussed here can be used to generate efficient test cases for coverage improvement, we also recognize that it requires users to have certain domain knowledge of the programs being tested and the test generation by itself is still not 100% automated. More research on automatic test generation is required.

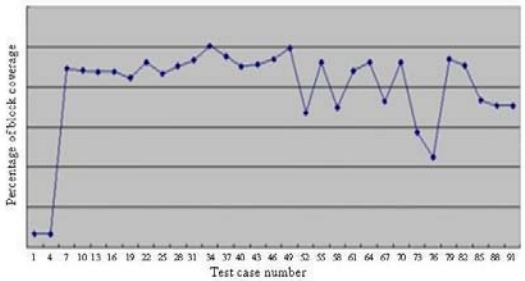


Figure 5. Block coverage of some test cases for Component 1

Studies have shown that we can apply coverage based criteria as a filter to reduce the size of a test set [21,22,23]. That is, as the size of a test set is reduced, while the code coverage is kept constant, there is little or no reduction in the fault detection effectiveness of the new test set so generated. Table 1 lists the number of test cases in the non-minimized test set, the block *minimized* test set, and the decision *minimized* test set with respect to each component. The non-minimized test set contains the "real life" test cases we received from testers at Texas Instruments. The block *minimized* test set is a subset of the non-minimized test set with the same block coverage, whereas the decision *minimized* test set gives the same decision coverage. The size reduction in terms of the number of test cases varies from 92.19% to 96.53% for the block minimized test set, and 92.18% to 98.43% for the decision minimized test set. Test cases in the minimized test sets should have a higher priority to be selected for regression testing if the resources prevent us from using every test case in the non-minimized test set [20].

By examining the uncovered code, we notice that each component has a set of unused code. For example, some code is only used for the debugging purpose. Such code is usually placed under an *if-then-else* structure to handle various kinds of error conditions. The actual debug statements can be removed by using a macro expansion during the release build of the component, but the control structure cannot be removed and this adds on to the percentage of uncovered code present in the source. Also, each component may have functions not fully implemented or supported in the current build. Such functions tend to bloat the percentage of uncovered code as well. We find that the block coverage is increased by 8.4%, 9.5%, 4.9%, 4.1%, and 10.1% for components 1, 2, 3, 4, and 5, respectively, after the unused code is excluded.

Table 1. Number of test cases in the non-minimized test set, the block minimized test set, and the decision minimized test set

	# of test cases in the non-minimized test set	Block minimized test set		Decision minimized test set	
		# of test cases	% of size reduction	# of test cases	% of size reduction
Component 1	573	32	94.42	9	98.43
Component 2	320	25	92.19	25	92.18
Component 3	375	13	96.53	7	98.13
Component 4	490	19	96.12	13	97.34
Component 5	585	23	96.07	12	97.94

5. Discussion

There are various tools available in the market that allow us to measure the code coverage for C/C++ programs. But very few tools work on the embedded platforms, and even fewer tools (e.g., CodeTest by Metrowerks [4]) work for Symbian C++ running the Symbian operating system on the OMAP platform. Even these tools have their own limitations and do not allow us the flexibility and the options that are present here. For example, although CodeTest provides the standard metrics such as statement coverage and decision coverage, it cannot prioritize uncovered code based on the coverage improvement. Nor can hot spots be identified to provide useful hints for efficient test generation. CodeTest cannot reduce the number of test cases using coverage as a filter either. Moreover, CodeTest does not help programmers take advantage of using code coverage for effective debugging and better program understanding as explained below.

In this paper, we propose that while conducting a source code-based coverage testing for embedded software, not only should we be able to report the cumulative coverage with respect to a set of tests and a given criterion (such as the block coverage criterion or the decision coverage criterion) but also decompose this coverage with respect to a selected subset of test cases on a selected part of the program being tested. For example, instead of reporting the block coverage of tests t_1, t_2, and t_3 on a C++ program with two classes C_1 and C_2, of which C_1 has two methods M_{11} and M_{12}, and C_2 has three methods M_{21}, M_{22}, and M_{23}, one might only be interested in the block coverage of tests t_1 and t_3 on method M_{12} of class C_1 and method M_{23} of class C_2. Such decomposed information can help programmers conduct effective debugging to quickly locate program bugs [28,29]. For example, suppose we have two test cases t_α and t_β, and the execution of a program $\mathcal{P}$ fails on t_α but succeeds on t_β. To find the bug in $\mathcal{P}$ we can first examine the code that is executed by t_α (the failed test) but not t_β (the successful test). Stated differently, let $\mathcal{E}_\alpha$ and $\mathcal{E}_\beta$ be the execution slices[2] with respect to t_α and t_α. A good debugging

[2] Given a program P and a test case *t*, the execution slice with respect to *t* on P is the set of code (e.g., in terms of blocks or decisions) in P executed by *t*.

heuristic is to first examine the code in $\mathcal{E}_\alpha$ but not in $\mathcal{E}_\beta$, i.e., examine the code in the execution dice obtained by subtracting $\mathcal{E}_\beta$ from $\mathcal{E}_\alpha$. Although such a heuristic is simple, our previous study found it can be effective in locating bugs [2]. However, to use this heuristic, we must be able to construct the execution slice with respect to each test case. This can be done easily if we know the coverage of each test case because the corresponding execution slice of a test case can be obtained simply by converting the coverage data collected for that test case during the testing into another format, i.e., instead of reporting the coverage percentage, it reports which blocks or decisions etc. are covered.

Similarly, code coverage information can also help programmers gain better understanding of the program structure [24,25,26]. For example, we can use execution slice-based techniques to identify code that is uniquely used by a specific feature or shared by a group features.

6. Conclusion and Future Direction

Techniques used for coverage testing software on Windows and/or UNIX/Linux platforms in general cannot be applied, without appropriate modification, to coverage testing embedded software. This is because testing embedded software is restricted by various factors such as the environment of the target device, the limited memory available on the target, and device control dependencies. In this paper, we propose a solution to overcome these problems for software running on the Texas Instruments OMAP-based devices and the Symbian operating system. To automate the testing procedure, a tool, TestingEmbedded was developed. We also report a case study to demonstrate the feasibility of our solution. In addition to coverage analysis, we explain how code coverage can aid in efficient test generation, reduction of the number of test cases, effective debugging, and better program understanding.

Our on-going research is to study how our solution can be applied to coverage testing embedded software on other platforms. In particular, we are interested in how the limitations in memory and processor speed coupled with stringent power requirements can affect the solution used for coverage testing embedded software.

Reference

1. H. Agrawal, "Dominators, super blocks, and program coverage," *Proceedings of the 21st Symposium on Principles of Programming Languages*, pp. 25-34, Portland, OR, January 1994
2. H. Agrawal, J. R. Horgan, S. London, and W. E. Wong, "Fault Localization using Execution Slices and Dataflow Tests," in *Proceedings of the 6th IEEE International Symposium on Software Reliability Engineering*, pp. 143-151, Toulouse, France, October 1995.
3. K. Dixon, "Symbian OS Version 7.0s Functional Description," June 2003,
4. "Freescale CodeTEST Software Analysis Tools", Freescale Semiconductor's Metrowerks
5. R. Harrison, Symbian OS C++ for Mobile Phones Volume 2, Wiley, 2004
6. K. Hollis, "Porting Opera to EPOC" http://www.symbian.com/developer/techlib/papers/Khopera/Opera_KeithHollis.htm
7. J. R. Horgan , S. London , L. R. Lyu, "Achieving Software Quality with Testing Coverage Measures," *Computer*, 27(9):60-69, September 1994
8. M. R. Lyu, J. R. Horgan, and S. London, "A Coverage Analysis Tool for the Effectiveness of Software Testing," *IEEE Transactions on Reliability*, 43(4):527-534, December 1994.
9. Y. W. Kim, "Efficient Use of Code Coverage in Large-Scale Software Development," in *Proceedings of The 2003 Conference of the Centre for Advanced Studies Conference on Collaborative Research*, pp. 145-155, Toronto, Canada, October 2003
10. H. Koehnemann and T. Lindquist, "Towards Target-Level Testing and Debugging Tools for Embedded Software," in *Proceedings of the conference on TRI-Ada '93*, pp. 288 – 298, Seattle, Washington, September 1993
11. "OMAP Platform: Processors, Software and Support," Texas Instruments
12. S. Rao, "Measuring and Improving Code Coverage for Embedded Systems," Master's Thesis, Department of Computer Science, University of Texas at Dallas, Richardson, Texas, December 2005
13. D. Mery, "Why is a Different Operating System Needed?", October 2003
14. M. Smith, "Symbian OS for Palm OS Developers," November 2001
15. "Symbian OS: From ANSI C/C++ To Symbian C++," March 2004,
16. "Symbian Developer Library," Symbian Ltd.
17. "Symbian Developer Network," Symbian Ltd.
18. A. Weinstein, "Symbian OS C++ for Windows C++ Programmers," October 2002
19. S. Whiteside, "Porting Simkin to Symbian OS," July 2003,
20. W. E. Wong, J. R. Horgan, S. London, and H. Agrawal, "A Study of Effective Regression Testing in Practice," in *Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering*, pp. 522-528, Albuquerque, New Mexico, November 1997
21. W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "On the Size, Coverage and Fault Detection Effectiveness of a Test Set," in *Proceedings of the 5th IEEE International Symposium on Software Reliability Engineering*, pp. 230-238, Monterey, CA, November 1994
22. W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of Test Set Minimization on Fault Detection Effectiveness," *Software-Practice and Experience*, 28(4):347-369, April 1998.
23. W. E. Wong, J. R. Horgan, A. P. Mathur, and A. Pasquini, "Test Set Size Minimization and Fault Detection Effectiveness: A Case Study in a Space Application," *Journal of Systems and Software*, 48(2):79-89, October 1999
24. W. E. Wong and S. S. Gokhale, "Static and Dynamic Distance Metrics for Feature-Based Code Analysis," *Journal of Systems and Software*, 74(3):283-295, February 2005
25. W. E. Wong, S. S. Gokhale and J. R. Horgan, "Quantifying the Closeness between Program Components and Features," *Journal of Systems and Software*, 54(2):87-98, October 2000.
26. W. E. Wong, S. S. Gokhale, J. R. Horgan, and K. S. Trivedi, "Locating Program Features using Execution Slices," in *Proceedings of the 2nd IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, pp. 194-203, Richardson, Texas, March, 1999.
27. W. E. Wong, T. Sugeta, J. J. Li, and J. Maldonado, "Coverage Testing Software Architectural Design in SDL," *Journal of Computer Networks*, 42(3):359-374, June 2003
28. W. E. Wong, T. Sugeta, Y. Qi, and J. C. Maldonado, "Smart Debugging Software Architectural Design in SDL," *Journal of Systems and Software*, 76(1):15-28, April 2005
29. W. E. Wong and Y. Qi, "An Execution Slice and Inter-Block Data Dependency-based Approach for Fault Localization," *Journal of Systems and Software* (accepted for publication)
30. "χSuds User's Manual," Telcordia Technologies (formerly Bellcore), 1998

Efficient and Effective Random Testing based on Partitioning and Neighborhood

Johannes Mayer
Ulm University
Dept. of Applied Information Processing
89069 Ulm, Germany
Email: johannes.mayer@uni-ulm.de

***Abstract*— Random Testing, i. e. testing with random inputs, is a simple and unbiased strategy for test input generation. Adaptive Random Testing (ART) denotes a family of random testing algorithms that need less test cases than Random Testing to detect the first failure. The most effective ART algorithms are Restricted Random Testing (RRT) and Distance-Based Random Testing (D-ART) which need very few test cases to detect the first failure. However, the runtime of these algorithms is at least quadratic in the number of test cases generated. In the present paper, algorithms based on partitioning and neighborhood are proposed to (approximately) implement RRT and D-ART in a way such that the average runtime is (almost) linear and the effectivity is equal to that of RRT resp. D-ART. The number of test cases necessary to detect the first failure is analyzed for the presented algorithms and the "original" methods using simulation and mutation analysis. Furthermore, the runtime of the proposed random testing methods and related methods is analyzed theoretically, complemented by an empirical analysis.**

I. Introduction

Software testing is the activity of executing a program with the intention to detect failures. This definition is old, but still valid, despite of newer trends such as e. g. Test Driven Development. An important problem in the automation of software testing is the generation of test cases. Random Testing [1], [2], [3], [4], [5], i. e. the random generation of test cases, gained much attention, since it is simple and unbiased, and can easily be automated. It has successfully been applied to test database systems [6] and a Java Just-In-Time (JIT) compiler [7], and was used to analyze the robustness of Windows NT applications [8]. It has however been criticized that Random Testing does not use information about the program under test [9].

Chan et al. [10] point out that failure-causing inputs usually cluster within the input domain. This gave rise to the assumption that wide-spread test cases have a better chance of detecting failures with less test cases. Adaptive Random Testing (ART) [11], [12] was designed based on this assumption. To compare several methods, the *F-measure* which is the number of test cases necessary to detect the first failure has been introduced by Chen et al. [12]. The most effective—in terms of the mean F-measure—ART algorithms (D-ART [12] and RRT [13], [14], [15]), however, have at least quadratic runtime and thus are quite inefficient. Therefore, a number of approaches were developed to improve the runtime using mirroring [15], [16], partitioning [17], [18], localization [19], [20], quasi-random numbers [21], lattices [22], and Voronoi diagrams [23]. The resulting methods, however, are not as effective as RRT resp. D-ART in terms of the mean F-measure or have more than asymptotic linear runtime. A higher mean F-measure is problematic, since this means more executions of the system under test (SUT). For non-trivial SUTs, the time consumed by one test execution cannot be neglected. Similarly, quadratic or "much more than linear" runtime of the test generation program is also problematic, since in this case test generation takes too long.

In the present paper, two ART algorithms are proposed that speed up the computation of RRT and D-ART, while preserving the very good mean F-measure. The runtime of the presented algorithms is (almost) linear (on average). The approach of the algorithms applies partitioning for the determination of neighboring test cases. The presented algorithms do not exactly implement RRT resp. D-ART, but only approximately. The mean F-measure of the proposed ART methods is analyzed using simulation and mutation analysis. A theoretical runtime analysis of the ART methods is conducted and the empirical runtimes are also compared.

Notation and a testing effectiveness measure are presented in Section II. Section III introduces related ART algorithms and gives a theoretical analysis of the runtime of all these methods. The proposed ART methods are presented in Section IV and their runtime is analyzed theoretically. Section V investigates the effectiveness of the presented methods and the related ART methods using simulation and mutation analysis. Furthermore, the empirical runtime of these methods is also analyzed. Section VI concludes the paper.

II. Some Preliminaries

The *F-measure* denotes the (random) number of test cases necessary to detect the first failure.

In the following, θ denotes the *failure rate*, i. e. the percentage of failure-causing inputs within the input domain. For Random Testing, the theoretical mean F-measure is $1/\theta$. For example, the theoretical mean F-measure of Random Testing is 100 if the failure rate is 0.01.

Since the F-measure depends on the failure rate, the *relative F-measure* which denotes the F-measure related to the theoretical mean F-measure of Random Testing is independent of the failure rate and therefore used for comparison.

III. Related ART Algorithms

In the following, ART algorithms important for the presented algorithms are briefly described. More details can be found in the respectively cited publications.

The first ART algorithm was Distance-Based ART (D-ART) [11], [12] with fixed sized candidate set. This method randomly generates the first test case. Thereafter, a set of k randomly chosen test case candidates is computed in each iteration. That test case of the candidate set with the greatest minimal (Euclidean) distance to all previously executed test cases (which did not exhibit a failure) is chosen as the next test case. In the following iteration, a new candidate set is chosen and the procedure is repeated until a failure is detected (or the resources for testing are exhausted). The size $k = 10$ of the candidate set has been recommended by Chen et al. The runtime of this algorithm is proportional to

$$\sum_{i=1}^{F} k(i-1) = k \sum_{i=0}^{F-1} i = k\frac{F(F-1)}{2} = \Theta(F^2)$$

and thus quadratic in F, where F denotes the number of test cases selected.

Another important ART algorithm is Restricted Random Testing (RRT) [13], [14], [15]. This algorithm chooses the first test case randomly. In each iteration a disc with radius $\sqrt{A \cdot R/(\pi \cdot n)}$ is located around each previously executed test case that did not exhibit a failure, where A denotes the area of the input domain, R denotes the coverage ratio, and n is the number of previously executed test cases not exhibiting a failure. Each randomly generated candidate that lies within one of these circular exclusion zones is rejected. The first candidate that does not lie within an exclusion zone is chosen as the next test case and the procedure is repeated until a failure is detected (or the resources for testing are exhausted). A coverage ratio of $R = 1.5$ was recommended by Chen et al. for quadratic input domains. With the assumption of a logarithmic number of candidates (in the number of previously executed test cases) in mind, the runtime of this algorithm is proportional to

$$\sum_{i=1}^{F}(i-1)\log i = \Theta(F^2 \log F)$$

and thus more than quadratic in F, where F is the number of test cases selected.

RRT and D-ART have a very good mean F-measure. However their runtime is at least quadratic and therefore not acceptable for low failure rates. Therefore, ART by Bisection was presented in [17] along with ART by Random Partitioning. These methods have the advantage that they are very efficient regarding runtime. ART by Bisection chooses the first test case randomly. Thereafter, the input domain is bisected (along the longer side). One of the two resulting partitions contains the already executed test case and the other does not. In each iteration, the "empty" partitions that do not contain a previously executed test case are selected in random order and a test case is generated randomly within each such region. As soon as all partitions contain previously executed test cases, all partitions are bisected along the longer side and partitions containing previously executed test cases are marked again. The selection of test cases within the "empty" partitions and the bisection of all partitions is repeated in each iteration. Some steps of the execution of this algorithm are illustrated in Figure 1. The "non-empty" partitions are shown hatched. Figure 1a shows the first iteration where one test case has been selected randomly. After the bisection at the end of the first iteration there remains only one "empty" partition for the second iteration. Figure 1b shows the generation of a test case within this region. After further bisection at the end of the second iteration, Test 3 is chosen within an "empty" region in iteration three (cf. Figure 1c). Finally, Figure 1d shows one step of the fourth iteration. The partitions have again been bisected after the third iteration. The runtime of ART by Bisection is linear in the number of executed test cases.

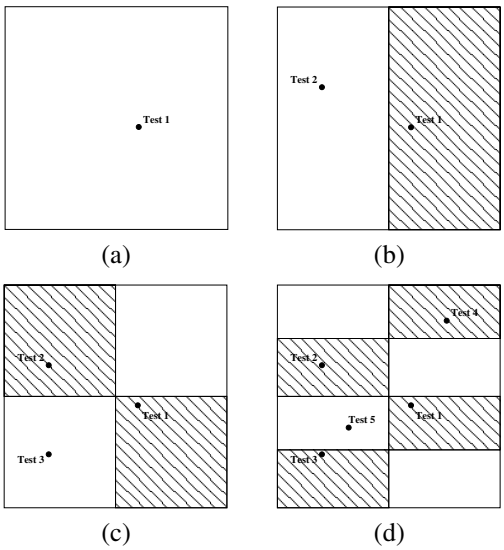
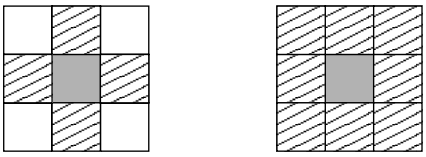


Fig. 1. Adaptive Random Testing by Bisection: Some steps of the algorithm

IV. Efficient Neighborhood-Based ART Algorithms

Although there are efficient ART algorithms, as described in the previous section, their mean F-measure is in many cases not as good as that of D-ART and RRT. Unfortunately, D-ART and RRT, the ART algorithms with the best mean F-measure, have at least quadratic runtime. The following algorithms combine ideas from partition-based ART methods and apply them to D-ART and RRT. The resulting algorithms have expected (nearly) linear runtime.

The proposed algorithms are inspired by ART by Bisection. Unlike ART by Bisection, the proposed algorithms use the partitioning only for the determination of neighbors (cf. Figure 2 for types of neighborhood), which enable us to dramatically reduce the number of distance computations (necessary in D-ART and RRT). However, test cases are always chosen from the whole input domain. The partitions are bisected each time the number of test cases is equal to the number of



(a) (b)

Fig. 2. Types of neighborhood: (a) 4-neighborhood and (b) 8-neighborhood

partitions. Therefore, the average number of test cases within one partition is small.

It is assumed that the two-dimensional input domain is rectangular with lower left corner $(x_{\min}, y_{\min})$ and upper right corner $(x_{\max}, y_{\max})$[1]. Therefore, the inputs are two-dimensional vectors (x, y) of real values with $x_{\min} \leq x \leq x_{\max}$ and $y_{\min} \leq y \leq y_{\max}$. It can trivially be adapted to a bounded region of integers or higher dimensional input domains.

The coverage ratio R must be within $[0, 1.5]$ (as for RRT).

Algorithm 1: Efficient Neighborhood-Based Restricted Random Testing (N-RRT)

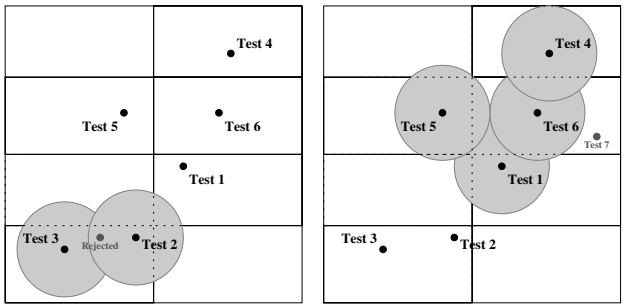
1) Let $T := \{(x_{\min}, y_{\min})(x_{\max}, y_{\max})\}$. Randomly select a point (x, y) within the region T. If (x, y) is a failure-causing input, report failure and terminate.
2) Initialize the list $L := \{(T, \{(x, y)\})\}$, where each element of this list is a pair consisting of a region and a list of test cases contained within this region. Initialize the target exclusion area $A_{excl} := \mathrm{R} \cdot (x_{\max} - x_{\min}) \cdot (y_{\max} - y_{\min})$. Let $N(L) := \sum_{(T,P)\in L} \#P$ be the number of test cases chosen so far.[2]
3) While $N(L) < \#L$ do:
   a) Set the exclusion radius $r := \sqrt{A_{excl}/(\pi \cdot N(L))}$.
   b) Randomly select a pair (T, P) from L and do *not* remove it.
   c) Randomly select a point (x, y) within the region T.
   d) For each neighbor T' of T with (T', P') in L:[3] If $\mathrm{dist}((x, y), P') \leq r$ then reject (x, y) and go back to Step 3.[4]
   e) If (x, y) is a failure-causing input, report failure and terminate.
   f) Otherwise replace (T, P) by $(T, P \cup \{(x, y)\})$ in L.
4) Initialize L_{temp} with the empty list.
5) For each element $(\{(x_0, y_0)(x_1, y_1)\}, P)$ of L do:
   a) Let $T := \{(x_0, y_0)(x_1, y_1)\}$. Furthermore, let $w := x_1 - x_0$ be the width of T and $h := y_1 - y_0$ be the height of T.

[1]Such rectangles are denoted $\{(x_{\min}, y_{\min})(x_{\max}, y_{\max})\}$ in the following.

[2]$\#L$ denotes the size of the list L.

[3]T is also considered a neighbor of itself.

[4]The distance $\mathrm{dist}((x, y), P)$ between a point (x, y) and a list of points P is defined as $\mathrm{dist}((x, y), P) := \min_{(x',y')\in P} \mathrm{dist}((x, y), (x', y'))$ if P is non-empty, and $\mathrm{dist}((x, y), P) := \infty$ otherwise.



(a) (b)

Fig. 3. Efficient Neighborhood-Based Restricted Random Testing

   b) If $w \geq h$, divide T into the two regions $T_1 := \{(x_0, y_0)(x_0 + w/2, y_1)\}$ and $T_2 := \{(x_0 + w/2, y_0)(x_1, y_1)\}$. Otherwise divide T into the two regions $T_1 := \{(x_0, y_0)(x_1, y_0 + h/2)\}$ and $T_2 := \{(x_0, y_0 + h/2)(x_1, y_1)\}$.
   c) Let $P_i := \{(x, y) \in P : (x, y) \in T_i\}$ for $i = 1, 2$
   d) Add (T_1, P_1) and (T_2, P_2) to L_{temp}.
6) $L := L_{\mathrm{temp}}$ and proceed with Step 3.

In each iteration, test case candidates are selected in Step 3. For each candidate it is checked, whether it is contained within one of the exclusion discs centered around the neighboring previously executed test cases.[5] In this case it would be rejected, otherwise it is kept and stored with the respective region. This is repeated until the number of test cases equals the number of regions. Thereafter, the partitions are bisected along their longer side and the test cases associated with a region are assigned to one of the two regions resulting from bisection. Figure 3a shows a candidate which is rejected, because it is contained in (at least) one of the discs. The next candidate in Figure 3b is then chosen as the next test case, since it is not contained in the exclusion zone. 4-neighborhood (cf. Figure 2a) has been chosen for this example (cf. Figure 3a, where there is no exclusion disc around Test 1). The test cases Test 2 and Test 3 resp. Test 6 and Test 7 illustrate that there may be more than one test case within a cell—unlike for partition-based ART methods—, since the partition is only used to determine neighboring previously executed test cases efficiently.

The search for neighboring regions can be implemented in constant time if the regions are not stored in a list, but in a two-dimensional array. In each iteration, a new array has to be allocated with one dimension having double size.

Conceptually, the presented method is not restricted to the efficient implementation of RRT. It is also possible to apply this idea to the efficient implementation of D-ART, yielding N-D-ART.

In the following, the runtime of the presented algorithms implemented with a two-dimensional array is to be investigated. Let $N_{i,j}$ denote the (random) number of test cases within the region with indices i and j at a certain time during

[5]Only the neighboring previously executed test cases are considered instead of all previously executed test cases (as for RRT).

the execution of the algorithm—just before the n th test case had been chosen. Let us furthermore assume that k candidates (including the selected test case) had been chosen for the n th test case and that there were m regions ($m \geq n$). For independently random test cases, the random variables $N_{i,j}$ would be identically binomially distributed with parameters n and $1/m$. Therefore, their mean would be $n/m \leq 1$. Due to the fact that RRT and D-ART make it harder to have two nearby test cases, it is reasonable to assume that the probability for $N_{i,j}$ being greater than 1 is less than in the case of binomial distribution. Therefore, we can argue that the mean of $N_{i,j}$ is not greater than 1. Thus, the mean number of distance computations necessary for all candidates is at most $5k$ resp. $9k$ for 4-neighborhood resp. 8-neighborhood. When the regions are bisected the first time after the n th test case has been chosen, m regions are bisected and m test cases are assigned to the $2m$ regions. Since there have been $m/2$ test cases added since the regions had been bisected last, an effort proportional to 4 can be assigned to each of the $m/2$ newly added test cases. Summarizing, the mean amortized time complexity of the selection of a single test case is proportional to $5k+4$ resp. $9k+4$ for 4-neighborhood resp. 8-neighborhood and thus in each case constant for the D-ART variant and probably logarithmically in the number of previously executed test cases for the RRT variant. The mean runtime of the algorithm is therefore in $\Theta(F)$ and thus linear in F for the D-ART variant, where F is the number of test cases selected. For the RRT variant it is no worse than proportional to $\sum_{l=1}^{F} \log l = \log(F!)$ and therefore in $\Theta(F \log F)$. (We assume that the number of candidates chosen by RRT is logarithmic in the number of previously executed test cases. Note that N-RRT chooses not more candidates than RRT.)

V. Empirical Study

In the following, the effectiveness of the proposed ART methods is investigated using simulation and mutation analysis. Besides this, an empirical analysis of the runtime of these methods is described.

A. Simulation

For the simulations, the sample size n was chosen 50,000, i. e. the algorithms were run with 50,000 randomly chosen failure patterns.

The failure patterns were randomly generated as in the related ART papers—in order to enable comparison. The area θA of the failure pattern was determined by the failure rate θ and the area A of the input domain. For the block pattern, a square was randomly chosen totally within the input domain. For the strip pattern, two adjacent sides and two points on these sides were chosen randomly. The strip was then constructed centered on the line connecting these points and its width was computed so that the strip had the desired area θA. Points near the corners were rejected to avoid overly wide strips. For the point pattern, 50 and alternatively 10 non-overlapping discs with equal radius being totally within the input domain were randomly generated to achieve the total area θA.

The simulations were conducted with $n = 50{,}000$ samples for the following

- failure rates: $0.01, 0.005, 0.002, 0.001, 0.0005$
- failure patterns: block, strip, point (with 10 discs), and point (with 50 discs)

The parameters of the various ART methods were as follows: RRT with $R = 1.5$ and D-ART with $k = 10$—as recommended in the respective publications. The parameter of the proposed algorithms were chosen according to the recommendations for the base algorithms (RRT resp. D-ART), i. e. Efficient Neighborhood-Based RRT (N_4-RRT resp. N_8-RRT; the "4" resp. "8" denotes the type of neighborhood adopted) with $R = 1.5$ and Efficient Neighborhood-Based D-ART (N_4-D-ART resp. N_8-D-ART) with $k = 10$. These parameters are also used in the analyses presented in the following sections.

The results of the simulation study for the block failure pattern are shown in Table I. This table contains the empirical mean relative F-measure and its accuracy on confidence level 99% (below).[6] In each column, the minimal value is set in bold face.

The difference between the mean relative F-measure of RRT and N_8-RRT resp. D-ART and N_8-D-ART is very small for each failure pattern and each failure rate and always within the range of the accuracies. Therefore, the simulations showed that the proposed methods are as effective as RRT resp. D-ART regarding the mean F-measure if 8-neighborhood is used.

B. Mutation Analysis

In order to confirm the simulation results and check their validity for real programs, we have used mutation analysis for the Numerical Recipes' [24] program "gammq".

Mutation testing [25], [26] is a fault-based testing method that can be used to measure the adequacy of a given test set. Small faults are introduced using so-called mutation operators [27]. Offutt [28] has shown that simple mutations are sufficient. Test sets that detect simple mutations will also detect complex ones. Recently it has been shown that mutation analysis can also be used to evaluate testing techniques [29].

We converted the program "gammq" to Java which required only little changes and used the tool MuJava [30] to automatically generate mutants. We only used "traditional" mutation operators, since "gammq" is not an object-oriented program. MuJava generated 534 mutants. 151 of them did not compile, 76 did not "survive" a 10 seconds timeout during Random Testing (since usually also non-terminating and equivalent mutants will be generated), and 72 had a failure rate of more than 0.05 (which can be determined through Random Testing, since the theoretical mean F-measure of Random Testing is the reciprocal of the failure rate). In total, we had thus 235 mutants for our mutation analysis.

[6] The accuracy is obtained through the Central Limit Theorem.

TABLE I

THE EMPIRICAL MEAN RELATIVE F-MEASURE OF THE RESPECTIVE ART METHOD FOR THE BLOCK FAILURE PATTERN

	$\theta = 0.01$	$\theta = 0.005$	$\theta = 0.002$	$\theta = 0.001$	$\theta = 0.0005$
RRT	**0.648** ($\pm$0.005)	0.633 ($\pm$0.005)	**0.612** ($\pm$0.005)	0.603 ($\pm$0.005)	0.595 ($\pm$0.005)
N_8-RRT	0.649 ($\pm$0.005)	**0.627** ($\pm$0.005)	**0.612** ($\pm$0.005)	**0.601** ($\pm$0.005)	**0.593** ($\pm$0.005)
N_4-RRT	0.664 ($\pm$0.006)	0.656 ($\pm$0.006)	0.640 ($\pm$0.006)	0.635 ($\pm$0.006)	0.630 ($\pm$0.005)
D-ART	0.673 ($\pm$0.006)	0.659 ($\pm$0.006)	0.650 ($\pm$0.006)	0.639 ($\pm$0.006)	0.635 ($\pm$0.006)
N_8-D-ART	0.673 ($\pm$0.006)	0.658 ($\pm$0.006)	0.642 ($\pm$0.006)	0.637 ($\pm$0.006)	0.633 ($\pm$0.006)
N_4-D-ART	0.676 ($\pm$0.006)	0.670 ($\pm$0.006)	0.657 ($\pm$0.006)	0.647 ($\pm$0.006)	0.644 ($\pm$0.006)

TABLE II

THE EMPIRICAL MEAN RELATIVE F-MEASURE OF 235 MUTANTS OF "GAMMQ" WITH FAILURE RATE LESS THAN OR EQUAL TO 0.05

ART Method	Mean	Minimum	Maximum
RRT	0.559	0.451	0.791
N_8-RRT	0.559	0.450	0.801
D-ART	0.520	0.405	0.767
N_8-D-ART	0.523	0.410	0.764

TABLE III

RUNTIME OF THE RESPECTIVE RANDOM TESTING METHOD FOR THE GENERATION OF A FIXED NUMBER OF TEST CASES

	Number of test cases generated		
Method	500	1000	2000
RRT	0.06 s	0.25 s	1.15 s
D-ART	0.06 s	0.23 s	0.96 s
N_8-RRT	0.0052 s	0.012 s	0.028 s
N_8-D-ART	0.0052 s	0.011 s	0.024 s
RT	0.000076 s	0.00016 s	0.00031 s

The input domain of "gammq" was chosen $\{(i, j) \mid 0 \leq i \leq 1700, 0 \leq j \leq 40\}$ as in [12].

Table II shows the average of the mean relative F-measure for each method (averaged over all 235 mutants). Furthermore, the minimum and the maximum of the mean relative F-measure is also given.

C. Runtime Analysis

The empirical runtime of RRT, D-ART, N_8-RRT, N_8-DART, and RT has been determined for the generation of test sets of different sizes, namely $500, 1000$, and 2000. The analysis was conducted on a Sun Fire V40z server with four AMD Opteron 850 CPUs each with 2.4 GHz and 16 GB main memory. The random testing techniques were implemented in Java and executed with the Java HotSpot^{TM} 64-Bit Server VM (build 1.5.0_06-b05, mixed mode). The runtime was averaged over 50,000 runs. Table III shows the runtime necessary for the generation of a fixed number of test cases by RRT, D-ART, N_8-RRT, N_8-D-ART, and RT.

D. Discussion

The simulation and the mutation analysis have shown that the proposed algorithms have the same mean F-measure as their approximated counterparts D-ART and RRT—up to the accuracy of the results.

From the theoretical analysis we know that the runtime of D-ART and RRT is at least quadratic and that of the neighborhood-based variants of D-ART resp. RRT is (almost) linear. Table III shows that the empirical runtime of RRT and D-ART is much higher than that of the proposed methods—even some orders of magnitude. As expected, the runtime of RT is the best one, also by some orders of magnitude smaller than that of N-RRT and N-D-ART.

Unlike Mirror Adaptive Random Testing [15], [16] which has nearly the same mean F-measure as D-ART resp. RRT, but only reduces the runtime by a constant factor, the proposed algorithms reduce the runtime from quadratic to linear asymptotic behavior. This is extremely important in case of small failure rates and consequently large test set sizes.

Another efficient implementation of RRT and D-ART has been proposed by Chen et al. [23]. This method is based on Voronoi diagrams. A Voronoi diagram is constructed for previously executed test cases and extended for every new test case. Then it is simple to determine the closest previously executed test case. Unlike the algorithms proposed in the present paper, this method yields exactly the same results as the original methods, since there is no approximation. The approximation is, however, exact in many cases where neighboring previously executed test cases exist. Besides this, the analysis of the methods proposed in the present paper shows that the effectivity (i. e. mean F-measure) of the neighborhood-based methods is the same as that of D-ART and RRT—up to the accuracy of the results. The runtime of the Voronoi solution is in $\Theta(F^{1+1/d})$ which is not as good as linear runtime, where d denotes the dimension. Furthermore, the idea of Voronoi diagrams can be used in the two-dimensional case. For higher dimensions it gets much more complex to work with Voronoi diagrams. In contrast, the proposed algorithms can easily be used in higher dimensions. In this case only the volume of the disc has to be replaced by the volume of the respective

hyper-sphere.

It seems to be difficult to use the Voronoi-based method for more complex input domains, whereas generalizations of our methods seem to be possible, requiring ways of distance computation and partitioning.

VI. Conclusion and Perspectives

Two ART algorithms have been presented using the idea of bisection for the selection of neighboring previously executed test cases. These algorithms are efficient approximate versions of RRT resp. D-ART. Whereas RRT and D-ART have at least quadratic asymptotic runtime, which is especially bad for small failure rates, the proposed algorithms have only (nearly) linear runtime (on average). However, these algorithms are approximations of the original RRT and D-ART methods. Not all previously executed test cases are used for the distance computations and the restriction—only neighboring ones. This is however only problematic in cases where there are no neighboring previously executed test cases, since otherwise the neighboring test cases are mostly the closest ones. It has been shown by simulation and mutation analysis that the effectiveness in terms of the mean F-measure, i. e. the mean number of test cases necessary to detect the first failure, is about the same for the proposed methods as for D-ART and RRT if 8-neighborhood is used. Therefore, the proposed algorithms are the first (expected) linear runtime ART algorithms with the excellent effectivity of RRT resp. D-ART.

Our proposed methods are quite simple to implement and yield a mean F-measure equal to that of D-ART resp. RRT. Furthermore, they have a very good runtime. Therefore, they should be preferred to Random Testing, and also D-ART and RRT.

At the moment, our research is fundamental and does not yet concern with more complex input domains. However, meaningful metrics, i. e. such that preserve the "neighborhood property" of failure-causing inputs, should be possible to obtain also for other input domains. Merkel [31] described a first approach to extend ART to more general input domains.

References

[1] V. D. Agrawal, "When to use random testing," *IEEE Transactions on Computers*, vol. 27, pp. 1054–1055, 1978.

[2] P. B. Schneck, "Comment on "when to use random testing"," *IEEE Transactions on Computers*, vol. 28, pp. 580–581, 1979.

[3] J. W. Duran and S. C. Ntafos, "An evaluation of random testing," *IEEE Transactions on Software Engineering*, vol. 10, pp. 438–444, 1984.

[4] P. S. Loo and W. K. Tsai, "Random testing revisited," *Information and Software Technology*, vol. 30, pp. 402–417, 1988.

[5] R. Hamlet, "Random testing," in *Encylopedia of Software Engineering*. Wiley, 1994, pp. 970–978.

[6] D. Slutz, "Massive stochastic testing of SQL," in *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB 1998)*, 1998, pp. 618–622.

[7] 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 (QSIC 2003)*. IEEE Computer Society, 2003, pp. 20–24.

[8] J. E. Forrester and B. P. Miller, "An empirical study of the robustness of Windows NT applications using random testing," in *Proceedings of the 4th USENIX Windows Systems Symposium*, 2000, pp. 59–68.

[9] G. J. Myers, *The Art of Software Testing*. New York: Wiley, 1979.

[10] 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*, vol. 38, pp. 775–782, 1996.

[11] T. Y. Chen, T. H. Tse, and Y. T. Yu, "Proportional sampling strategy: A compendium and some insights," *The Journal of Systems and Software*, vol. 58, pp. 65–81, 2001.

[12] T. Y. Chen, H. Leung, and I. K. Mak, "Adaptive random testing," in *Proceedings of the 9th Asian Computing Science Conference (ASIAN 2004)*, ser. Lecture Notes in Computer Science, M. J. Maher, Ed., vol. 3321. Springer, 2004, pp. 320–329.

[13] K. P. Chan, T. Y. Chen, and D. Towey, "Restricted random testing," in *Proceedings of the 7th European Conference on Software Quality (ECSQ 2002)*, ser. Lecture Notes in Computer Science, J. Kontio and R. Conradi, Eds., vol. 2349. Springer, 2002, pp. 321–330.

[14] ——, "Normalized restricted random testing," in *Proceedings of the 18th Ada-Europe International Conference on Reliable Software Technologies*, ser. Lecture Notes in Computer Science, vol. 2655. Springer, 2003, pp. 368–381.

[15] K. P. Chan, T. Y. Chen, F.-C. Kuo, and D. Towey, "A revisit of adaptive random testing by restriction," in *Proceedings of the 28th International Computer Software and Applications Conference (COMPSAC 2004)*. IEEE Computer Society, 2004, pp. 78–85.

[16] T. Y. Chen, F.-C. Kuo, R. G. Merkel, and S. P. Ng, "Mirror adaptive random testing," *Information and Software Technology*, vol. 46, pp. 1001–1010, 2004.

[17] T. Y. Chen, G. Eddy, R. Merkel, and P. K. Wong, "Adaptive random testing through dynamic partitioning," in *Proceedings of the 4th International Conference on Quality Software (QSIC 2004)*. IEEE Computer Society, 2004, pp. 79–86.

[18] J. Mayer, "Adaptive random testing by bisection with restriction," in *Proceedings of the Seventh International Conference on Formal Engineering Methods (ICFEM 2005)*, ser. Lecture Notes in Computer Science, vol. 3785. Springer-Verlag, Berlin, 2005, pp. 251–263.

[19] T. Y. Chen and D. Huang, "Adaptive random testing by localization," in *Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC 2004)*. IEEE Computer Society, 2004, pp. 292–298.

[20] J. Mayer, "Adaptive random testing by bisection and localization," in *Proceedings of the Fifth International Workshop on Formal Approaches to Testing of Software (FATES 2005)*, ser. Lecture Notes in Computer Science, vol. 3997. Springer-Verlag, Berlin, 2006, (in press).

[21] T. Y. Chen and R. Merkel, "Quasi-random testing," in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005)*. New York: ACM, 2005, pp. 309–312.

[22] J. Mayer, "Lattice-based adaptive random testing," in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005)*. New York: ACM, 2005, pp. 333–336.

[23] T. Y. Chen and R. Merkel, "Efficient and effective random testing using the Voronoi diagram," in *Proceedings of the Australian Software Engineering Conference (ASWEC 2006)*. IEEE Computer Society, 2006, pp. 300–308.

[24] W. H. Press, B. P. Flannery, S. A. Teulolsky, and W. T. Vetterling, *Numerical Recipes*. Cambridge University Press, 1986.

[25] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on test data selection: Help for the practicing programmer," *IEEE Computer*, vol. 11, pp. 34–41, 1978.

[26] J. Offutt and R. H. Untch, "Mutation 2000: Uniting the orthogonal," in *Proceedings of Mutation 2000: Mutation Testing in the Twentieth and the Twenty First Centuries, San Jose, CA*, 2000, pp. 45–55.

[27] K. N. King and A. J. Offutt, "A fortran language system for mutation-based software testing," *Software Practice and Experience*, vol. 21, no. 7, pp. 685–718, 1991.

[28] A. J. Offutt, "Investigations of the software testing coupling effect," *ACM Transactions on Software Engineering Methodology*, vol. 1, no. 1, pp. 5–20, 1992.

[29] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?" in *Proceedings of the 27th International Conference on Software Engineering (ICSE 2005)*. ACM, 2005, pp. 402–411.

[30] J. Offutt, Y.-S. Ma, and Y.-R. Kwon, "An experimental mutation system for Java," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 5, pp. 1–4, 2004.

[31] R. Merkel, "Analysis and enhancements of adaptive random testing," Ph.D. dissertation, Swinburne University of Technology, Australia, 2005.

Enhanced Anomaly Detection in Self-Healing Components

Michael E. Shin
Department of Computer Science
Texas Tech University
Lubbock, TX 79409-3104
Michael.Shin@.ttu.edu

Yan Xu
Department of Computer Science
Texas Tech University
Lubbock, TX 79409-3104
yan.xu@ttu.edu

Abstract

This paper describes an approach to enhancing the accuracy of detection of anomalies in self-healing components using both multiple notification messages from objects and relationships between threads of active objects in the components. When a component monitor finds out a latent anomaly in an object, it takes a closer look at the object with the next notification messages before it reaches its final verdict on the object. Active objects can be tightly coupled via a connector to synchronize the message communication between them. The synchronization relationship between the active objects is used to detect anomalies in the active objects.

1. Introduction

The important factors in the mechanisms for anomaly detection in software components are speed - how quickly the mechanisms detect anomalies - and accuracy - how correctly they detect anomalies. A component is anomalous if it does not behave as specified. The dependable systems [Avizienis01] are in need of anomaly detection mechanisms that are capable of providing both high speed and accurate detection of anomalies in the systems.

However, an uncertain event in a system may be classified as an anomaly prematurely to get high speed at the cost of the accuracy, whereas the high accuracy of anomaly detection may be achieved at the price of speed of anomaly detection. It is inevitable that a detection mechanism may compromise one factor in order to attain high quality of another factor.

This paper describes an approach to improving the accuracy of a mechanism that detects anomalies in self-healing components of a system. A self-healing component is a component that is able to autonomously detect and repair anomalies of objects in the component. This paper is an extension to our previous work in which the detection mechanism relies on only individual notification messages from objects being monitored.

This paper is organized as follows. This paper begins with related work in section 2. Section 3 describes self-healing components. Section 4 describes anomaly detection using multiple notification messages. Section 5 describes anomaly detection using relationships between threads of active objects in a component. Finally, section 6 concludes this paper.

2. Related Work

Considerable research for anomaly detection has been done using time testing, referred to as "heartbeating" [Felber99, HA01, Kim00, Mills04], or exception [Guerra02].

The heartbeating technique depends on message handshaking between a monitor and a component (or subsystem) being monitored, which should be performed at predefined, periodic times. An anomaly is detected if an event, expected within a specified time, does not occur. This technique can verify that a component still provides functional services to others.

In [Guerra02], anomalies in a component are detected using exceptions, which may be generated by the component when the component cannot successfully process a service request from others. To achieve this, the exceptions and their conditions for operations of each component should be defined at design-time.

3. Self-Healing Components

Each self-healing component [Shin05] can be structured into a layered architecture with the service layer and the healing layer (Fig.1). The service layer of a self-healing component provides services requested from neighboring components. The healing layer of a self-healing component takes care of anomalies detected in the service layer, notifying the status of self-healing to the healing layers of neighboring components.

The service layer of each self-healing component is composed of objects such as tasks (active or concurrent objects), passive objects accessed by tasks, and connectors between tasks. A task, depicted using a thick outline for the object box in Fig. 1, has its own thread of control, initiating actions that affect other tasks and passive objects [Gomaa00]. Unlike a task, a passive

object (e.g., an entity object) has no thread of control; thus it cannot initiate any task. A passive object is invoked by tasks and can invoke other passive objects. Since a passive object does not have its own thread, it performs its operations using the thread of the task that invoked the object. Tasks in a component can communicate with each other through connectors. On behalf of tasks that have threads, a connector between tasks synchronizes the message communication between the tasks. Passive objects and connectors between tasks, which do not have their threads, are depicted using thin outlines for the object boxes in Fig.1.

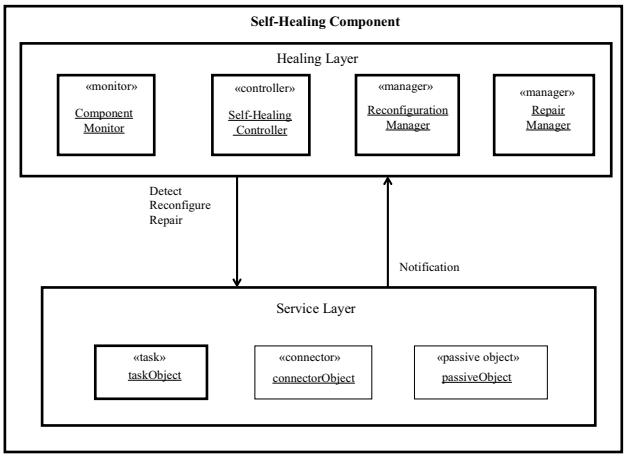


Fig.1. Self-Healing Component Architecture

The healing layer of each self-healing component (Fig. 1) is structured into four objects, such as *Component Monitor*, *Reconfiguration Manager, Repair Manager,* and *Self-Healing Controller,* which are responsible for detection, reconfiguration, and repair of anomalous objects in the service layer:

- Component Monitor. With messages notified by the service layer, the *Component Monitor* detects the anomalies of objects in the service layer using statecharts specifying the behavior of task threads.
- Reconfiguration Manager. The *Reconfiguration Manager* generates reconfiguration plans for the service layer, carrying out the plans to isolate anomalous objects from healthy objects.
- Repair Manager. The *Repair Manager* maintains information for repairing anomalous objects in the service layer, repairing anomalous objects.
- Self-Healing Controller. The Self-Healing Controller coordinates objects in the healing layer to conduct the self-healing mechanism, cooperating with self-*healing controllers* of other self-healing components for reconfiguration.

Anomalies of tasks, passive objects, and connectors within a self-healing component are detected by the component monitor [Shin05, Shin06] in Fig. 1, which watches the behavior of task threads in the component. Whenever a connector between tasks and an operation of a passive object are initiated by a task, they notify the component monitor, which uses those notification messages to detect anomalies in the tasks, connectors between tasks, and passive objects. To monitor task threads in a component, a component monitor encapsulates statecharts [Rumbaugh04, Booch05] describing the specification of each task thread. A task, connector between tasks, or passive object that is controlled by a task thread is anomalous if a statechart for the task thread does not execute within a specified time interval.

4. Anomaly Detection using Multiple Messages

The component monitor in a component detects anomalies of objects in the component on the basis of each single message notified by connectors between active objects, and by passive objects accessed by active objects [Shin06]. Anomaly detection using a single notification message may lead to a misjudgement if the notification message is just missing during delivery of the message to the component monitor. The misjudgement by the component monitor launches the recovery procedure in the system, which interrupts the normal services of the component. The mistaken recovery process just allows the component to provide partial services or, at the extreme case, not allow any more services.

Misjudgements of a component monitor depending on a single notification message can be reduced by using multiple notification messages before the component monitor has reached its decision on anomalies of objects in the component. In anomaly detection using multiple notification messages, a component monitor does not make a decision about object anomaly immediately even though a notification message has not arrived within its expected time interval. Instead, the component monitor closely observes the predefined number of the next notification messages. A component monitor defines the number of notification messages for each thread of active objects (i.e., N notification messages, which is referred to as a N-message monitoring window), which should be monitored prior to its final decision on object anomaly.

A component monitor allows some number of notification messages to be missing, but the number of missing notification messages should be less than N in the N-message monitoring window. Otherwise, the component monitor presumes that some object related to the thread of an active object is anomalous. A component monitor starts observing objects in the component at a base of the N-message monitoring window. The

component monitor shifts the base of an N-message monitoring window to the first notification message within the window if some notification message arrives. The component monitor begins observing messages in the new N-message monitoring window at the new base.

Fig. 2 depicts a two-message monitoring window and the component monitor's decisions corresponding to notification messages. In cases 1 and 2, the component monitor shifts the base of the window from the first message to the second message, observing notification message in the new window. In case 3, the monitor also shifts the base of the window to the second message though it has suspected some object in the component due to the missing of the first message. With the second message arrived within an expected time interval, the component monitor regards the first message as a miss during the communication between an object (that is, a connector between tasks or a passive object accessed by tasks) and the component monitor. In case 4, the component monitor definitely reports that a specific task, passive object, or connector is anomalous.

Case	First Message	Second Message	Monitor Decision
Case 1	The first message arrives within ETI	The second message arrives within ETI	Shifts the base of the window from the first message to the second message, and then starts the new window.
Case 2	The first message arrives within ETI	The second message does not arrive within ETI	
Case 3	The first message does not arrive within ETI	The second message arrives within ETI	
Case 4	The first message does not arrive within ETI	The second message does not arrive within ETI	Reports anomaly detected

ETI: Expected Time Interval

Fig.2. Two-Message Monitoring Window

5. Anomaly Detection using Relationships among Threads of Active Objects

By considering the relationships among threads of active objects in a component, the rate of misjudgement of a component monitor can be improved more than that in anomaly detection described in section 4. Statecharts encapsulated in a component monitor describing the specifications of threads of active objects are concurrently executed each other. However, statcharts for the threads of some active objects are executed (partially) sequentially if the message communication between the active objects is tightly coupled. The rate of misjudgement of a component monitor can be reduced using the partially sequential relationship in message communication between the active objects.

More specifically, a connector used in a tightly coupled message communication is classified as either a message buffer connector, or a message buffer and response connector [Gomaa00]. A message buffer connector is used for tightly coupled message communication without reply (synchronous message communication without reply), which adjusts the difference of the processing speeds between sender and receiver active objects. A message buffer connector encapsulates a message buffer whose size is one. Once the sender has stored a message in the buffer, it is suspended until the receiver takes the message. After the receiver reads a message from the buffer, the sender can store the next message in the buffer.

In Fig. 3, the threads of Active Object1 and Active Object2 are partially sequential if they have a tightly coupled message communication (though they are concurrently running). The message buffer connector sends the notification message "A3: Message Placed" to the component monitor using the thread of Active Object1 after a message has been stored in the buffer ((a) of Fig.3). The notification message makes a transition in the statechart ((b) of Fig. 3) for the thread of Active Object1 to the state "Waiting for Acknowledge". And then the component monitor should receive a notification message, "A5: Read Invoked" or "A6: Message Read", which is sent by the connector using the thread of Active Object2 ((c) of Fig.4). After that, the next notification message arriving at the component monitor should be "A7: acknowledged" that is sent by the connector using the thread of Active Object1. Thus the notification messages to the component monitor are partially sequential between the threads of Active Object1 and Active Object2 (Fig. 4).

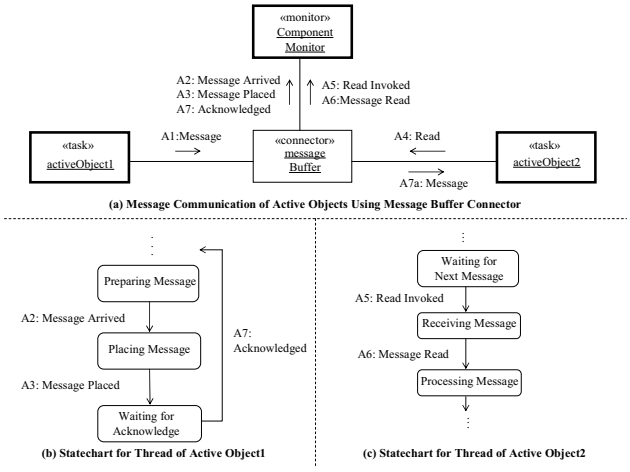


Fig. 3 Anomaly Detection in Tightly Coupled Message Communication without Reply

Sequence	1	2	3	4
Notification Message	A2	A3	A6 or (A5, A6)	A7
Used Thread	Active Object1	Active Object1	Active Object2	Active Object1

Fig. 4. Notification Message Sequence and Threads of Active Objects

The sequential relationship among the notification messages is used to enhance the accuracy of anomaly detection in the component monitor. Suppose that the component monitor has a 2-message monitoring window in section 4 and both notification messages A2 and A3 have not arrived within specified time intervals. If the notification message A6 is arrived, it means that the Active Object1 has delivered a message to the Active Object2 successfully. The component monitor presumes that the notification messages A2 and A3 are just missing, but the objects related to the thread of Active Object1 are still normal. Similarly, the component monitor checks the notification message A7 when the notification messages A5 and A6 are not received. If the A7 arrives at the component monitor, it shows that the messages A5 and A6 are just missing, and the objects related to the thread of Active Object2 are still normal until the point.

Similarly, the active objects in a tightly coupled message communication with reply (synchronous message communication with reply) communicate messages (partially) sequentially. The connector for tightly coupled message communication with reply encapsulates a single message buffer and a single response buffer (each buffer size is one), and provides synchronized operations to send a message, receive a message, and send a reply. The sender active object invokes the send message operation while the receiver active object invokes the receive message and reply operations. Once the sender has stored a message in the buffer, it is suspended until the response is received from the receiver. The sender and receiver active objects should be synchronized to communicate messages, and the message communication becomes partially sequential. The sequential relationship in the message communication between the active objects can be used to enhance the accuracy of anomaly detection in the component monitor.

6. Conclusions

This paper has described an approach to enhancing the accuracy of detection of anomalies in self-healing components using multiple notification messages from objects and relationships between concurrent threads of active objects. An N-message monitoring window has been used to take a closer look at suspected objects before the component monitor determines the status of objects. The sequential relationship between the threads of active objects in a component also has been used to enhance the accuracy of anomaly detection in the component monitor.

References

[Avizienis01] Avizienis, A., Laprie, J.C., and Randell, B., "Fundamental Concepts of Dependability", Research Report N01145, LAAS-CNRS, April 2001.

[Booch05] G. Booch, J. Rumbaugh, I. Jacobson, "The Unified Modeling Language User Guide", 2nd Edition, Addison Wesley, 2005.

[Felber99] Felber, P., D´efago, X., Guerraoui, R., and Oser, P., "Failure detectors as first class objects" in Proceedings of the International Symposium on Distributed Objects and Applications (DOA'99), Edinburgh, Scotland, 1999, pp. 132–141.

[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 of ICSE'02, Orlando, FL, May 25, 2002.

[HA01] HA Forum, "Providing Open Architecture High Availability Solutions," Revision 1.0, February, 2001

[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 Engineering., Vol.12, No.2, March/April 2000, pp.145-159.

[Mills04] Mills, K., Rose, S., Quirolgico, S. Britton, M., and Tan, C., "An Autonomic Failure-Detection Algorithm", In Proceedings of the 4th International Workshop on Software Performance (WOSP 2004), San Francisco, California, January 14-16, 2004.

[Rumbaugh04] J. Rumbaugh, G. Booch, I. Jacobson, "The Unified Modeling Language Reference Manual (2nd Edition)," Addison-Wesley Professional, 2004.

[Shin05] 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.

[Shin06] Michael E. Shin, Yan Xu, Fernando Paniagua and Jung Hoon An, "Detection of Anomalies in Software Architecture with Connectors," Science of Computer Programming, Volume 61, Issue 1, June 2006, Pages 16-26.

Program Testing Using High-Level Property-Driven Models

Isabel Michiels[1], Coen De Roover[1], Johan Brichau[1,2], Elisa Gonzalez Boix[1], Theo D'Hondt[1]

[1]Programming Technology Lab
Vrije Universiteit Brussel, Belgium

[2]Laboratoire d'Informatique Fondamentale de Lille
Université des Sciences et Technologies de Lille, France

Abstract

Testing is a crucial part of the software development life cycle that necessitates adequate techniques and tools. On the one hand, unit testing techniques are particularly easy and lightweight to use but are restricted to testing the external behaviour of a program. On the other hand, testing techniques that use property-driven models of the software are able to test behavioural properties of the entire execution of a program. However, these models are often specified in terms of low-level execution events of the program. In this paper, we present the lightweight property-driven testing platform called BEHAVE*, where tests are written as property-driven behavioural models over high-level run-time events. We validate our approach by testing behavioural properties related to the automatic memory management in the interpreter of the Pico programming language.*

1. Introduction

The increasing complexity and size of today's software systems provides ample opportunities for developers to introduce faulty and erroneous behaviour in the system's implementation. The presence of such *bugs* in many finished software products emphasizes the need for integrated testing in the development process. However, software testing is a laborious activity that is often still conducted without any adequate tools and thus consumes more than 50% of the total software development effort [3].

Model-oriented verification techniques [5] permit us to accurately verify the entire program's behaviour against its specification, but they are also time-consuming, complex and do not scale very well. This is because the entire behaviour of a software system must be specified in a complete and precise behavioural model of all execution states of that system. On the other end of the spectrum, unit testing techniques like the XUnit open source testing frameworks [9](e.g. SUnit for Smalltalk, CUnit for C, etc.) are particularly lightweight. Using these techniques, developers focus on testing small localizable parts of the system's implementation incrementally, based on particular usage scenarios. Although these incremental approaches address the scalability problem and are relatively easy to use, they do not verify all aspects of the behaviour of the executing system. Instead, they can merely test the external behaviour of particular parts of a system. Using property-driven models for testing combines some of the advantages of model-oriented verification and unit-based testing. They are lightweight in the sense that they only model particular *properties* of the program's execution. Moreover, in contrast to unit testing, they also test non-externally verifiable behaviour of the executing program with respect to the modeled properties. Nevertheless, property-driven testing does require the specification of behavioural models of these properties and these are often specified in terms of low-level execution events of the program.

In this paper, we present the property-driven testing platform BEHAVE and apply it to test particular behavioural properties of memory management in the Pico language interpreter. The main contribution of BEHAVE is that the behavioural models are specified using a declarative formalism which renders the models machine-verifiable and, at the same time, understandable to the developers [13]. More specifically, BEHAVE offers support for testing particular behaviour of an application by specifying property-driven models *over* high-level run-time events *using* temporal logic programming. We will start our paper by clarifying our case study of the Pico language interpreter and its automatic memory management in section 2. In section 3 we will introduce our platform called BEHAVE together with its underlying approach and we will demonstrate the use of our platform by testing garbage collection in Pico. We will end with related work in section 4 and a conclusion in section 5.

2. The Pico Interpreter and its Memory Model

Pico [12] is an interpreted programming language developed at the Vrije Universiteit Brussel. While it was origi-

nally conceived to teach programming concepts to students outside the realm of computer science, its interpreter implementation (which totals about 8000 lines of condense C code) is nowadays also used for teaching about programming languages, interpreters and automatic memory management. In this paper, we use the Pico interpreter as a case study to illustrate our approach. We will test a particular kind of erroneous behaviour related to automatic memory management.

The Pico memory consists of a heap that is managed by an automatic garbage collector. The garbage collection algorithm can be triggered every time a chunk of memory is requested. If no sufficient memory is available, the garbage collector will traverse the Pico environment and possibly defragment it. As a result, chunks can be moved to a completely different address in the Pico heap. This change of location happens transparently because the garbage collector also updates any references to that chunk. However, this requires that all references to that chunk are also stored on the heap, which is not always the case. In many parts of the Pico implementation, references to chunks of memory on the Pico heap need to be stored in a temporary variable inside a C function. More particularly this happens in so called 'continuation functions' which are C functions that represent a part of program execution in Pico. They can be recognized as functions with no return type and no parameters. If a garbage collect occurs during the execution of such a continuation function, those references to the Pico memory might become invalid and the Pico interpreter will crash.

Obviously, we want to detect such an unwanted behaviour through careful testing. This means that we need to detect the occurrence of a garbage collect in between the assignment and the use of a temporary variable that holds a reference to the Pico memory. Unit testing is insufficient because such a test only fails if a garbage collect actually defragments memory. Depending on the actual memory consumption and organisation, such tests might thus fail or not, although the erroneous behaviour of a *possible* garbage collect is always present. Therefore, we need to test the *possible* occurrences of a garbage collect in between the assignments and the uses of a temporary variable inside a C function. To this extent, we developed a property-driven model that is verified using the BEHAVE platform.

3. Property-Driven Testing with BEHAVE

In general, three important phases can be distinguished in application testing: specification of a model, verifying the model against the actual behaviour (running the test) and evaluating the test results. Using BEHAVE, developers specify property-driven models using a declarative programming language. These models express certain behavioural properties that must hold throughout an application's lifetime and they are automatically verified with respect to a high-level execution-trace of the program. This execution trace is *high-level* because it contains the application-specific events that are required to verify the model. These high-level events are also specified by the developer. Figure 1 provides a general overview of all constituents of this approach when applied to the testing of the garbage collector in the Pico interpreter. The Pico implementation consists of many continuation functions, of which an excerpt is shown in figure 1b. The execution of this function results in the creation of an execution-trace (shown in figure 1a) that only contains the observed behaviour in terms of high-level events. These high-level events are used to verify the property-driven model specified in figure 1c. Finally, figure 1d, 1e and 1f specify how the high-level events need to be recorded during the execution of the program. We will further explain the details of each of these parts throughout the remainder of the paper.

To summarize, the use of the BEHAVE platform for property-driven testing of software applications amounts to the following 4-step recipe:

1. Identify the high-level run-time events of the property to be tested,
2. Specify a property-driven model,
3. Specify the application-specific instances of high-level run-time events,
4. Run the test by verifying the property-driven model against the actual behaviour and evaluate the results.

In the following sections we will explain each of these steps through the application of BEHAVE for testing the garbage collector of the Pico language interpreter.

3.1. Identifying the High-Level Run-Time Events

The first step of our recipe comprises the identification of the high-level run-time events we need to verify the garbage collection property mentioned in section 2. Having analysed that description, we need to specify the following high-level run-time events: possible garbage collect (`possibleGC`), temporary variable used (`tempUsed`) and temporary variable assigned (update or initialisation) (`tempUpdated`). Indeed, we want to detect occurrences of possible garbage collection events *in between* the assignment and use of temporary variables holding a reference to the Pico memory. Figure 1a shows instances of these events in an execution trace using logic (Prolog) facts. Each event also declares additional information associated to the event. For example, each event contains a time-stamp, which is the first value that is shown in each event instance in figure 1a. The `tempUsed` event on line 2 also declares the name of the

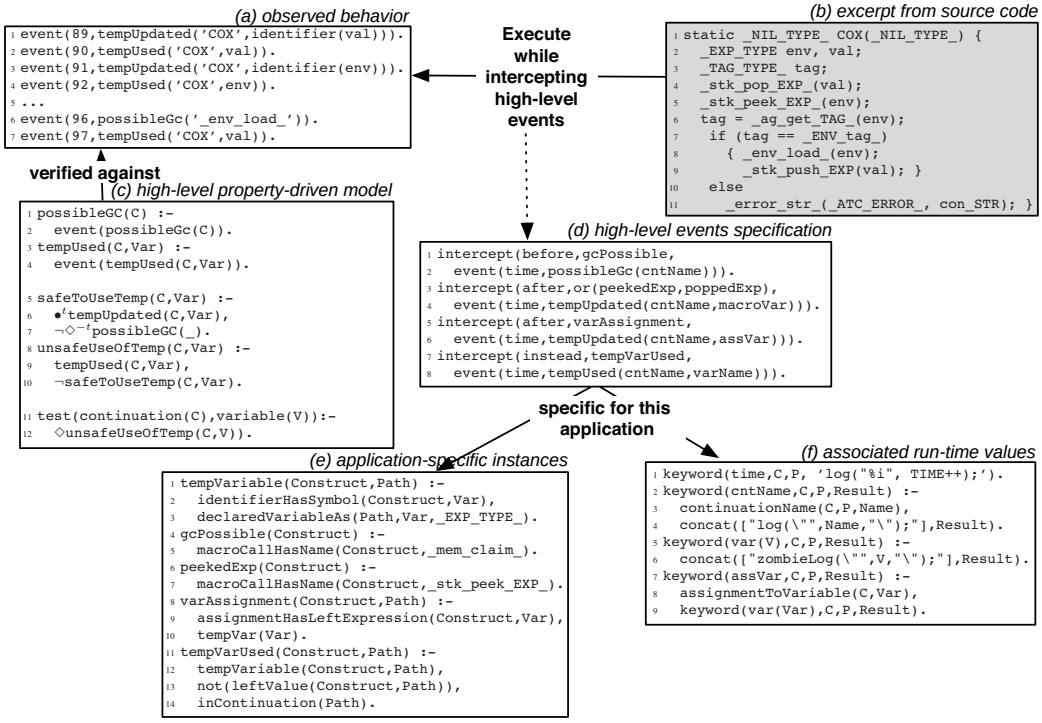


Figure 1: Using the BEHAVE platform for testing Pico memory management

temporary variable (`val`) and the name of the continuation function (`COX`) in which it is defined.

The identification of these high-level run-time events allows the property-driven model to be specified at a high level of abstraction, i.e. in terms of these high-level events instead of low-level execution traces. Furthermore, because a developer can specify which high-level events to intercept, this approach also achieves a selective and compact execution trace only containing the needed run-time information. Note that the identification of these high-level events is completely up to the developer; he has to specify *when* such a high-level event occurs and also *how* to obtain any additional run-time information. We will explain in section 3.3 how these high-level events are specified such that they are recorded during the execution of the program.

3.2. Specifying a Property-Driven Model

In the second recipe step of BEHAVE, we specify a property-driven model using a temporal logic programming language based on Prolog. We chose a logic language as specification language because of its declarative nature. This ensures that a model will be human-readable and at a high level of abstraction, which is needed for specifying a property in terms of concepts rather than low-level programming constructs, such as a call to a function. In Prolog, a program consists of logic clauses representing knowledge about a particular problem at hand. As we have explained in the previous section, the high-level execution trace of the C program also consists of a set of logic facts. This means that we can implement the property-driven model as a set of high-level assertions *over* the execution trace, expressed using logic clauses.

Our choice for using the temporal logic programming language MTL [4], a temporal variant of Prolog, is particularly useful because temporal logic programming comprises all functionality of logic programming and provides some additional logic operators for expressing temporal constraints. Such temporal constraints are of extreme importance when reasoning about events representing run-time behaviour. The temporal operators which we will further use in our property-driven model are: $\Box$ (always), $\Diamond$ (sometimes), $\bullet$ (previous) and $\circ$ (next). Using these operators, we can describe temporal relations between events contained in the execution trace in an even more expressive manner as opposed to when you use traditional logic programming. However, although temporal logic formulae have been successfully applied in the program verification domain, they are sometimes hard to understand. Therefore users not familiar with temporal logic are still free to specify their models using plain Prolog assertions.

Finally, since we use our platform for testing an application's behaviour, we are always looking for occurrences of *unwanted* behaviour. Testing is an activity that tries to find

out where the behaviour of a program is broken. Therefore, our property-driven models will need to specify unwanted behaviour and *where* in the source code this unwanted behaviour occurs. This approach is of importance for correcting the source code of the application in case a test fails, as we will explain in section 3.4.

Using the high-level events identified in section 3.1, we can now specify a property-driven model using temporal logic clauses that describes the desired as well as the undesired behaviour of the program with respect to these events. In section 2, we described that a possible garbage collect event may not occur between the assignment (which is either an update or an initialisation) and the use of a temporary variable. In other words, *after a possible garbage collection event you should not use temporary variables unless they have been updated in between.* Expressed as unwanted behaviour, we will look for all uses of temporary variables that do not obey this last saying. The property-driven model that specifies this behaviour is shown in figure 1c. Lines 1-4 introduce additional abstractions over the events in the execution trace, i.e. `possibleGC` and `tempUsed` [1]. On lines 5-10 the concepts of unsafe and safe uses of temporary variables are defined as the logic assertions `unsafeUseOfTemp` and `safeToUseTemp` respectively. This last assertion (defined on lines 5-7) states that it is safe to use a temporary variable `Var` within a continuation function `C` if within `t` timesteps in the past from now, the variable `Var` has been updated (or given an initial value if it didn't have a value before) and if within that time frame no possible garbage collection could have occurred. The `unsafeUseOfTemp(C,Var)` assertion on lines 8-10 then captures the unsafe use of a temporary variable that states the complement. The actual test can be seen on lines 11-12 and expresses the wish of finding a variable `V` within a continuation function `C` that is used in an unsafe way, as it is defined above.

3.3. Specifying Application-Specific Instances of High-Level Events

In the previous sections, the high-level events that we need for verifying garbage collection were identified and the property-driven model was expressed as assertions over these high-level events using temporal logic programming. In this subsection, we describe how a developer that uses the BEHAVE platform can specify *which* events have to be intercepted and *how* they should be recorded. This consists of specifying *what* source-code constructs raise these high-level events (figure 1e), how associated run-time values need to be extracted in order to be recorded with the event (figure 1f) and the specification of the events themselves (figure 1d). Once again, logic programs are used to describe these actions.

The declarations in figure 1d describe the high-level events that need to be recorded. All declarations are of the form `intercept(When,What,RecordAs)`. We will explain these declarations by example. Consider lines 1-2 where we declare that the high-level event of a possible garbage collect is recorded in the execution trace as `event(time,possibleGc(cntName))`. More precisely, this event will be recorded right `before` the execution of the source code construct that is identified by the `gcPossible` assertion. This assertion is defined by the logic clause on lines 4-5 in figure 1e. This clause specifies that the high-level event of a possible garbage collect is triggered by the execution of the source-code construct `Construct` if it is a C macrocall that is named `_mem_claim_`. There are three other macrocalls in the Pico implementation besides _mem_claim_ that can trigger the garbage collection property. However, due to space limitations we did not include all of them here but there exists a very similar logic clause for each of them. Furthermore, instead of merely logging the occurrence of a possible garbage collect event, we also log the `time` at which the event occurred and the name of the continuation function `cntName` [2] in which the possible garbage collect event occurs. We will explain how the run-time values of these so-called *keywords* `time` and `cntName` are computed later on.

In lines 3-6, the same high-level event (`event(time,tempUpdated(cntName,varName))`) is described by two different logic declarations. This is because the high-level event of a temporary variable being (re)assigned a value can be manifested in the source code in various ways. More specifically, this event is triggered by the execution of the source code construct that is specified by the `peekedExp`, the `poppedExp` or the`varAssignment` assertions. Another difference between the two clauses has to do with the different run-time values that are needed. These are represented by the keywords (`macroVar` and `assVar`.

The source code constructs can be identified because BEHAVE makes an entire application's parse tree available. Let's have a look at the logic clause on lines 1-3 in figure 1e that defines a temporary variable as the `tempVariable(Construct,Path)` assertion. This rule has access to each parse tree node through the `Construct` variable, while the `Path` variable represents the path from the tree's root that leads to that `Construct` node. This defines what construct is a temporary variable pointing to a Pico memory chunk in the Pico implementation. In essence, some source-code construct is such a temporary variable if it is an identifier with name `Var` and if it has been declared

[1] Note that we also need the same abstraction for `tempUpdated(C,V)`, but we didn't include it here due to space limitations.

[2] Note also that `cntName` and `time` are not logic variables here; in Prolog logic variables are denoted with a capital letter

as being of type _EXP_TYPE_. The tempVarUsed rule on lines 11-14 needs some more explanation. This rule states that a temporary variable (denoted by the variable Construct) has been used if first of all it is a temporary variable (line 12), if it is found to be used within a continuation function (line 14) and if it is not part of the left side of a variable assignment (line 13), because then it is not used but updated.

We still need to explain how the run-time information associated with each high-level event can be retrieved. In figure 1d on line 2 we denoted that for the high-level run-time event possibleGc we wanted to log the name of the continuation function where the possible garbage collection event could occur through the cntName keyword. These run-time values will have to be obtained by the execution of application-specific source code. The most important keywords used here together with the associated C code can be found in figure 1f. On line 1, the time keyword is defined as a call to a C log function that will merely write a number to the execution trace file and increment a time counter. The keyword cntName on lines 2-4 uses a continuationName rule, which is not included here, but this rule looks for the continuation function node in the Path of this Construct and takes its name. The two rules on lines 5-9 are used in combination to be able to log the name of a variable to which a variable has been assigned [3].

3.4. Running the Test and Evaluating the Results

Verifying the property-driven model against the actual behaviour and then evaluating the results forms the last step of our recipe. Our platform first instruments the source code of the application under test to be able to record all possible occurrences of the defined high-level run-time events. Second, the instrumented source code gets executed according to a particular scenario which creates the execution trace containing the high-level run-time events that occurred during execution. Afterwards, the property-driven model can be verified by launching the logic query `:- test(Function,Variable).` A fail of this query would mean that no variables can be found that are used in an unsafe way. However, if this query yields results, the results will describe which variables in which continuation functions are used in an unsafe way.

Applying our approach to the complete Pico implementation, we were able to find three occurrences of unsafe usage of variables in two different continuation functions. Let us consider one of the two results: `continuation('COX'), variable(val);`. This result means that in the COX continuation function the temporary variable val is used in an unsafe way. Depending on the state of the heap representing underlying Pico memory, if a garbage collect event occurred that triggers a heap defragmentation during the execution of the COX continuation function, the system would crash completely. The C code fragment representing this function is depicted in figure 1b: on line 9 the temporary variable val is used and apparently the statement on line 8, a call to the function _env_load triggered a possible garbage collection event (as can be seen in figure 1a on line 6).

[3] Note that we need two completely analogous rules to the one on lines 7-9 for the keywords macroVar and varName as used in figure 1d

4. Related Work

A lot of research has been done on dynamic analysis approaches, differing in the domain they are applied, analysis time, expressiveness of the medium used to express behavioral models, what run-time events can be intercepted, etc. We briefly discuss those most closely related to our approach.

CCI [14], is a general program monitor notification tool for C programs. At run-time, an execution monitor is notified of events through calls to a user-implemented macro which takes an integer indicating the type of event that occurred and an event-specific associated value. As CCI is a pure program monitor notification tool, it provides no language specifically tailored to reasoning about intercepted events, which our platform does provide. JMonitor [10] provides a Java API for specifying event patterns and associating them with user provided event monitors. JMonitor doesn't offer any mechanism either to analyse observed events.

Coca [6] performs a dynamic analysis on C programs by having them executed in a stepwise fashion by a debugger. Coca runs alongside the debugging process to steer the execution of the program. Analysis is done on-line, so considering alternative matches for a run-time event without advancing the application is not possible. This makes it difficult to express assertions about nested events. Coca is however suited to debugging purposes as queries can be expressed in a Prolog variant augmented with predicates to request future run-time events.

Auguston et al. [1, 2] present the interesting procedural assertion specification language FORMAN. Atomic low-level events occur at a time point, while composite events occupy an interval in time. An event grammar formally specifies the low-level constituents of composite events and their mutual ordering on the time line. This allows for automatic low-level event selection according to a given assertion over composite events. The information recorded about each composite event is however dependent on its atomic constituents. In this setting we prefer the more declarative nature of the property models resulting from our temporal logic programming specification language. Our experiments have also indicated the benefits of being able to freely determine the run-time values associated with high-

level events. Due to the lack of a formal event grammar, our approach however requires more user involvement.

TestLog [7] focuses on testing Smalltalk applications. It also uses a Prolog-like logic language, but reasons about whole-program execution traces comprising method invocations and the recursive state of each receiver before and after the invocation.

5. Conclusion

We presented the BEHAVE platform for testing behavioural properties of software applications using declarative property-driven behavioural models. These models are formulated as declarative high-level temporal assertions over high-level run-time events freely chosen by the application tester. Our approach features the following characteristics:

- Because our testing approach is property-driven, we can verify a specific part of program execution; this makes our approach applicable to larger programs as well;
- The most commonly used dynamic analysis approaches demand a model being written over a fixed set of low-level programming language constructs; using the BEHAVE platform an application tester can specify himself what run-time events should be recorded; consequently verification is again computationally less expensive because of the selectively chosen events in the execution of a program;
- Moreover, the high-level property-driven models are written in a declarative medium which makes them more human-readable; hence they can also serve as documentation artefacts between successive application testers during software evolution;

To cope with the large degrees of freedom our platform offers, we outlined a four-step recipe which guides a user of our platform for testing a specific system property. We validated this recipe through testing a particular kind of behaviour of the garbage collector in the Pico language interpreter and were able to find some undiscovered bugs in the implementation. The BEHAVE platform has also been validated in the context of documenting and verifying high-level behavioral program documentation [13].

Up until now, our approach has only been applied to the Pico case study for two different properties. However it will be applied onto other case studies as well in the near future. It would also be fairly straightforward to apply our approach to object-oriented applications. So far we only applied our approach to C code, but we have the technology available to apply it to programs written in the object-oriented programming paradigm as well using SOUL [11, 8].

Furthermore we are planning experiments to use our platform in a teaching environment to aid a student in grasping the concepts of a particular program under study.

References

[1] M. Auguston. Building Program Behavior Models. In *Proc. of the European Conf. on Artificial Intelligence Worksh. on Spatial and Temporal Reasoning (ECAI98)*, pages 19–26, 1998.

[2] M. Auguston. Tools for Program Dynamic Analysis, Testing, and Debugging Based on Event Grammars. In *Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 159–166, July 6-8 2000.

[3] F. Brooks. *The Mythical Man-Month*. Addison-Wesley, 2nd edition edition, 1995.

[4] C. Brzoska. Temporal Logic Programming with Metric and Past Operators. In *Proc. of the Worksh. on Executable Modal and Temporal Logics*, pages 21–39, 1995.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[6] M. Ducassé. Coca: An automated Debugger for C. In *Proc. of the 21st Intl. Conf. on Software engineering (ICSE99)*, pages 504–513, 1999.

[7] S. Ducasse, M. Freidig, and R. Wuyts. Logic and Trace-based Object-Oriented Application Testing. In *Proc. of the Intl. Worksh. on Object-Oriented Reengineering (WOOR04)*, 2004.

[8] J. Fabry and T. Mens. Language-Independent Detection of Object-Oriented Design Patterns. *Elsevier International Journal on Computer Languages, Systems & Structures - Proceedsings of the ESUG 2004 Conference.*, 30(1-2):21–33, 2004.

[9] P. Hamill. *Unit Test Frameworks*. O'Reilly, November 2004.

[10] M. Karaorman and J. Freeman. jMonitor: Java Runtime Event Specification and Monitoring Library. In *Proc. of the 4th Intl. Worksh. on Run-time Verification (RV04)*, volume 113 of *Electronic Notes in Theoretical Computer Science*, pages 181–200, 2004.

[11] K. Mens, I. Michiels, and R. Wuyts. Supporting Software Development Through Declaratively Codified Programming Patterns. In *Proceedings of the 13th International Software Engineering and Knowledge Engineering Conference (SEKE01)*, 2001.

[12] W. D. Meuter, T. D'Hondt, and J. Dedecker. Pico: Scheme for Mere Mortals. In *Online Proc. of the 1st European Lisp and Scheme Worksh.*, 2004.

[13] C. D. Roover, I. Michiels, K. Gybels, K. Gybels, and T. D'Hondt. An Approach to High-Level Behavioral Program Documentation Allowing Lightweight Verification. In *Proceedings of the 14th International Conference on Program Comprehension (ICPC 2006), Athens, June 14-16 (To be published)*, 2006.

[14] K. Templer and C. Jeffery. A Configurable Automatic Instrumentation Tool for ANSI C. In *Proc. of the 13th IEEE Intl. Conf. on Automated Software Engineering (ASE98)*, page 249, 1998.

A Multi-Agent Based Architecture For Distributed Testing

Mohammed Benattou

Institut d'Ingénierie Informatique Limoges
Laboratoire XLIM - UMR CNRS 6172 -
123, avenue Albert Thomas - 87060 LIMOGES - France
E-mail: benattou@3il.fr

Abstract

Software agent components promote a variety of abstractions, protocols and mechanisms that aim to make them easier to design, develop and implement complex distributed collaborating systems. They extend the traditional software component by including mobility, autonomy, collaboration, persistence and intelligence. In this paper, we firstly show, how software agent components used in distributed testing architecture contribute to separate the complex monitoring tester behavior tasks. Secondly, by reducing the synchronized exchanging messages in the distributed testing framework, we show the powerful characteristics of mobile agent paradigm.

keywords: Multi-agent, Mobile agent, Distributed testing, Component, Synchronization, Coordination

1. Introduction

Nowadays, component-based software engineering has become the key issues in modern system design. It provides a high-level abstraction and services for developing, integrating and system managing of distributed system applications. The component-based software engineering has promised, and indeed delivered significant improvements in software development [4].
However, in practice, the development of distributed systems using component technologies is more complex. The design process must take into account the mechanisms and functions required to support interaction as long as communication and coordination between distributed components. Examples of such applications are systems comprising a set of components that broadcast commands among themselves, multicast controllers that coordinate messages between components, the systems using the alarm signals in non deterministic order, network and application systems, event queues, etc. [11]. The typical reaction of such systems is the generation of errors sets: time-outs, locks, channels and network failures. The most often programming models used to implement these environments rely on an event loop with call-backs, or using the CORBA services.
Our preliminary experience, in the use of the CORBA services is the implementation of the distributed testing application [1]. The basic idea of the proposed work [10] is to coordinate the testers by using a communication service parallel to the Implementation Under Test IUT through a multicast channel. Object-oriented based, the implementation of the proposed model has shown that the execution of distributed testers arise many time-outs problems influencing fault detection during the testing process. In this context, software agents are specialized kinds of distributed components, providing greater flexibility than traditional components [3]. Agent technologies promote a variety of abstractions, protocols and mechanisms that aim to make them easier to design, develop and implement complex distributed collaborating systems. They extend the traditional software component by including mobility, autonomy, collaboration, persistence and intelligence [7].
This paper addresses some technical issues for using software agent in distributed testing framework. Firstly, we show how Multi-Agent based system used in distributed testing architecture contributes to separate the complex monitoring tester behavior tasks. Secondly, by reducing the synchronized exchanging messages in the distributed testing framework, we show the powerful characteristics of mobile agent paradigm. The paper is structured as follows. Section 2 describes the architecture and modelling concept of distributed testing systems. Section 3 is dedicated to describe some synchronization problems arisen in distributed testing execution and proposes some solving issues. Section 4 describes our multi-agent based architecture. Section 5 gives some conclusions and identifies future works.

2. Distributed Conformance Testing

Conformance testing may be seen as a mean to execute an IUT by carrying out test cases, in order to observe

whether the behaviour of the implementation is conform to its specification. In the context of open distributed system IUT represents the executable implementation of the distributed software system to be tested. It can be seen as a black-box with points of control and observation (PCO) to which the test system can apply the input events and observe the output results during the testing process. The basic idea of the proposed work [10] is to coordinate the testers by using a communication service parallel to the IUT through a multicast channel. As shown in figure 1, each tester interacts with the IUT only through the attached port and communicates with the other testers through the multicast channel. The execution process of a tester consists of: apply input events to the IUT interface related to it; observe the output results; and to provide its local verdict. Conformance of an IUT to its specification may be

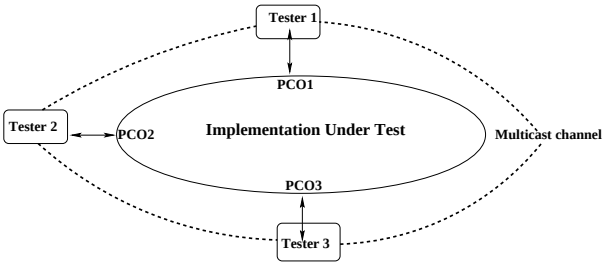


Figure 1. Distributed test architecture

defined by means of relations between the IUT model and specification model [6]. I/O FSM (Input/Output Finite State Machines) are widely used in the communication protocol area [6], and may be easily adapted with some extensions for modeling distributed systems. In a communication protocol, a protocol entity communicating with a peer entity is described by an I/O FSM with one input queue and one output queue. Distributed applications are supposed however to communicate with multiple partners. This leads to the notion of a multi-port FSM [9] which may use several input/output queues called ports (For more theoretical details, see [10, 1]). Figure 2 gives an example of 3p-FSM with set state $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$, q_0 being the initial state, Σ_k is the *input alphabet of port* k: $\Sigma_1 = \{a\}$, $\Sigma_2 = \{b\}$, $\Sigma_3 = \{c\}$, and Γ_k is the *output alphabet of port* k: $\Gamma_1 = \{w, x\}$, $\Gamma_2 = \{y\}$, $\Gamma_3 = \{z\}$. Transitions are represented by arrows. A test sequence for 3p-FSM of the example illustrated in figure 2 is:

$$!a?\{x,y\}!b?\{x,y\}!c\{z\}!a?\{x,z\}!b?\{x,y\}!c?\{w,z\}!a?\{x,z\}!c?\{y,z\} \quad (1)$$

$!x_i$ means sending message x_i to the IUT and $?y_i$ means receiving the messages belonging to y_i from the IUT. Generally test sequences are generated from the IUT specification

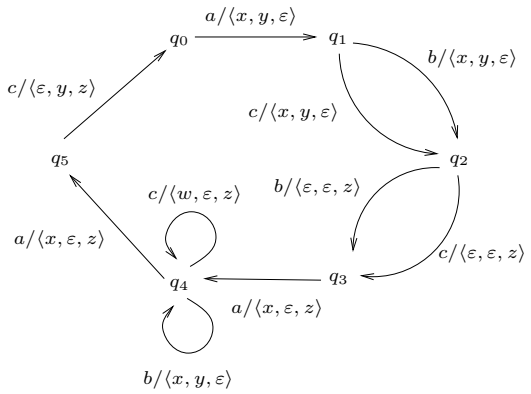


Figure 2. An example of 3p-FSM

and Characterized by their fault coverage (output faults and transfer faults [9]). In order to synchronize the test execution, testers exchange coordination messages. These messages encapsulate the information that allows the test system to solve controllability and observability problems. As pointed out in [10] each local test sequence executed by a given tester includes the coordination messages to be sent or to be received from other testers. Applying the proposed algorithm to the test sequence (1), we get the following local test sequences:

$$\begin{cases} \omega_1 & = & !a?x?x?C_3?C_3!a?x?x?w!a?x?C_3 \\ \omega_2 & = & ?y!b?y!C_3?C_3!b?y!C_3?C_3?y \\ \omega_3 & = & ?C_2!C_1!c?z!C_1?z!C_2?C_2!c?z?z!C_{\{1,2\}}!c?z \end{cases} \quad (2)$$

Where $!C_{h_1,\ldots,h_r}$ is sending of coordination message C to testers $h_1, \ldots, h_r$ and $?C_h$ is receiving of coordination message C from tester h.

3. Test control

Our preliminary experience, in the use of the event CORBA service for the implementation of distributed testing applications have shown that the parallel testers execution arise many time-outs problems influencing fault detection during the testing process .

As we have show in [1] the execution of the first fragments of the local test sequences ω_1, ω_2 and ω_3 (2) gives the failed result. Indeed, if the tester 2 sends the message b to the IUT before the tester 1 receives the output message x from the IUT, and thus the execution of the local sequences is not conform to the ω specification (1), where the message b must be sent to the IUT after all output messages caused by the sending of the message a by tester 1 has been received. The execution example illustrates [1] that the synchronisation between tester objects allows some failed errors, which can be solved by blocking the sending of the message b as

long as all outputs, caused by the last message, have been received by all testers. It should be noticed that, the local test sequences analyse of (2) gives:
-Any message sending by tester to the IUT must be blocked as long as all output events, caused by the last sending message, have been received by all testers. This behavior is not described in any a local test sequences.
-Difficulties begin with output events identification, and the relating input source causing them.
In order to solve these problems we follow the steps outlined below:
1. Transform the general test sequence generated from the given *multi-port FSM with n ports* by indexing the execution order of input and output events. This order gives the natural execution of each general test sequence transition.
For example, the test sequence given in (1) is transformed to:

$$!a_1?\{x_1,y_1\}!b_2?\{x_2,y_2\}!c_3?\{z_3\}!a_4?\{x_4,z_4\}!b_5?\{x_5,y_5\}!c_6?\{w_6,z_6\}$$
$$!a_7?\{x_7,z_7\}!c_8?\{y_8,z_8\} \quad (3)$$

2. Identify the output events that must be observed before sending any output event to the IUT. In this context, each input event must include the output events caused by the last sending message. We denote by $!M_i^{\{x_{i-1},y_{i-1}\}}$ the sending of message M of the transition i to the IUT after the output events $\{x_{i-1}, y_{i-1}\}$ has been observed in related testers. This new representation is incorporated in the test sequence given in (4):

$$!a_1?\{x_1,y_1\}!b_2^{\{x_1,y_1\}}?\{x_2,y_2\}!c_3^{\{x_2,y_2\}}?\{z_3\}!a_4^{\{z_3\}}?\{x_4,z_4\}!b_5^{\{x_4,y_4\}}$$
$$?\{x_5,y_5\}!c_6^{\{x_5,y_5\}}?\{w_6,z_6\}!a_7^{\{w_6,z_6\}}?\{x_7,z_7\}!c_8^{\{x_7,z_7\}}?\{y_8,z_8\} \quad (4)$$

3. Apply the algorithm [10] to the test sequence achieved by the last step to generate local test sequences related to each tester. The applying of the proposed algorithm to the test sequence (4), gets the following local test sequences:

$$\begin{cases} \omega_1 = !a_1^{\{\}}?x_1?x_2?C_3?C_3!a_4^{\{z_3\}}?x_4?x_5?w_6!a_7^{\{w_6,z_6\}}?x_7?C_3 \\ \omega_2 = ?y_1!b_2^{\{x_1,y_1\}}?y_2!C_3?C_3!b_5^{\{x_4,y_4\}}?y_5!C_3?C_3?y_8 \\ \omega_3 = ?C_2!C_1!c_3^{\{x_2,y_2\}}?z_3!C_1?z_4!C_2?C_2!c_6^{\{x_5,y_5\}}?z_6?z_7 \\ \qquad !C_{\{1,2\}}!c_8^{\{x_7,z_7\}}?z_8 \end{cases} \quad (5)$$

Notice that, the form of coordination messages has not been changed.

4 Agent Based Architecture

Agent technologies are a potentially promising approach for building complex systems that require intelligent behavior from a collection of collaborating, autonomous software components [2]. This section presents our Multi-Agent architecture of distributed testing application. As shown in figure 3, on a high level abstraction allowing the organizational requirements of the distributed testing environment to be captured, we can distinguish three actors: Manager Agent, Tester Agent, and Multicast channel. *Multicast*

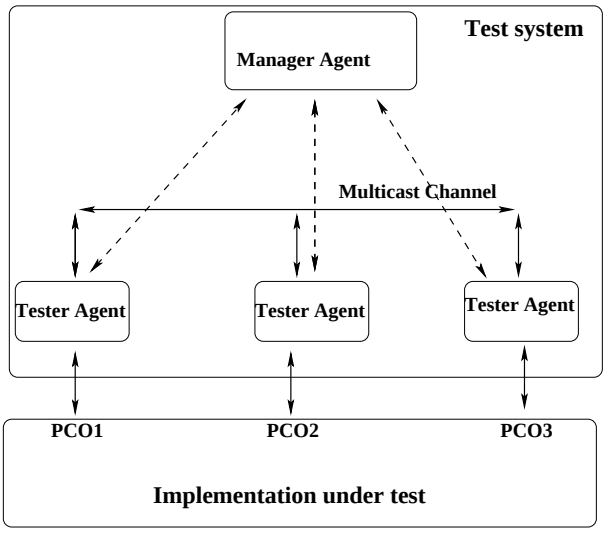


Figure 3. Multi-Agent architecture

channel: as we have described above, Multicast channel allows a coordination message sent by a given tester to be delivered to the interested testers. Almost Multi-Agent platforms provide both communication mechanisms: synchronous and asynchronous messaging. Cougaar [5] supports inter-agents communications through mechanism known as *allocation*. Its architecture supports one-way, asynchronous, and one-to-one communication. In Aglet [8], both synchronous and asynchronous communication is provided. Aglet also supports multicasting in publish-subscribe manner.
Manager Agent: Its behavior can be described by a set of ordered actions. The generation of local test sequences and the analysis of a global verdict constitute the principal actions that compose its internal activity.The distribution of test sequences and the commands to start and to end the test were considered as the actions allowing the *Management agent* to control Agent Testers.
Tester Agent: represents the engine component of our system. Due to the complex analysing tasks made by tester for detecting output faults on related IUT PCO, *Tester Agent* delegates performed tasks to well defined agents. As shown in figure 4, Tester Agent delegates predetermined performed tasks to three agents:
-*Coordination agent*: it allows a *Tester* to exchange coordination messages with other Tester. According to its local sequences, *Coordination agent* is firstly responsible to send coordination messages to well defined testers using

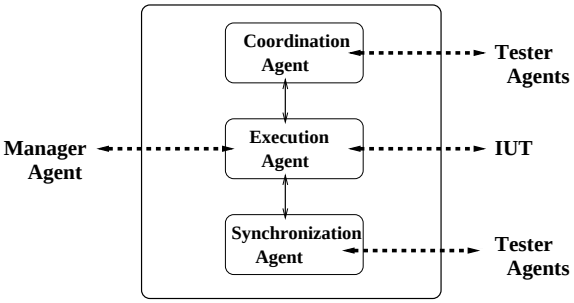


Figure 4. Tester Agent Structure

Multicast channel. Secondly, it informs tester to block or to continue its execution according to the reception of coordination messages.

-*Synchronization agent*: as we have shown above, any message sending by tester to the IUT must be blocked as long as all output events, caused by the last sending message, have been received by all testers. Firstly, it is responsible to search and to confirm the output event in specified testers. Secondly, it must inform its related tester to send the input event to the IUT, when all synchronized messages have been received by specified testers.

-*Execution agent*: allows tester to apply input event actions to IUT and to receive output results from the IUT. On the other hand, it allows tester to send its verdict to The *Manager agent.*

The message sending $!M_i^{\{x_{i-1},y_{i-1}\}}$ by some tester to IUT can not be executed only if the messages $\{x_{i-1}, y_{i-1}\}$ have been received by related testers. And then, related testers receiving these messages must inform the considered tester. In this context, we redefine the behavior of each tester described above only for the sending message to IUT as follow:

For each message sending with the form $!M_i^{Waiting\_messages}$ each *Tester agent* execute the following steps using delegate agents:

-*Synchronization agent*: detect for each output message belonging to the set of $Waiting\_messages$ its related tester and send a mobile agent to set of detected testers.

-*Execution agent*: block the message sending M_i to IUT until receiving the mobile agent. that confirms the observation of all considered output events and send the message M_i to the IUT.

5. Conclusion and future works

Not only distributed computing becomes the key issue in modern system design, but it provides new high possibilities for Internet-based applications. However, in practice the development of distributed component systems is more complex. Especially where the implementation must take into account some synchronization rules and the coordination of distributed components, as it is often the case in complex information systems. In this paper, we firstly show, how software agent components used in distributed testing architecture contribute to separate the complex monitoring tester behavior tasks. Secondly, we have described how we can use the concept of mobile agent to reduce synchronized exchanging messages in the distributed testing framework. Our work is now oriented toward the development of an environment, which take into consideration the functional requirements of time constraints in this context.

References

[1] M. Benattou and Jean-Michel Bruel, *Active Objects for Coordination in Distributed Testing*, Proceedings of the 8th Int. Conf. on Object-Oriented Information Systems OOIS'02, Lecture Notes in Computer Science, Vol 2425, pp 348-357, 2002.

[2] I. Gorton, J. Haack, D. McGee, A. J. Cowell, O. Kuchar, J. Thomson, *Evaluating Agent Architectures: Cougaar, Aglets and AAA*, SELMAS 2003: 264-278, 2003.

[3] Martin L. Griss, *Software Engineering with Java Agent Components*, Invited paper, Borcon, Nov 2003.

[4] G. Heineman and Bill Council,(eds), *Component-Based Software Engineering: Putting the pieces together*, edited by Addison-Wesley, June 2001.

[5] http://www.cougaar.org/introduction/overview.html

[6] ISO/IEC, *Information retrieval, transfer and management for OSI, Framework: formal methods in conformance testing*, CD 13345-1, 1996.

[7] W. A. Jansen, *Determining Privileges of Mobile Agents*, 17th Annual Computer Security Applications Conference, pages 149-160, 2001.

[8] D. Lange and M. Oshima, *Programming and Deploying Java Mobile Agent whith Aglets*, Addison Wesly, 1998.

[9] A. Petrenko, G. v. Bochmann and M. Yayo, *On fault coverage of tests for finite state specification*, Computer Network and ISDN Systems 29, 1996, pp. 81-106.

[10] O. Rafiq, L. Cacciari, M. Benattou: *Coordination Issues in Distributed Testing*; Proceeding of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99), Las Vegas (Nevada), USA, 1999, CSREA Press.

[11] A. J. Restrepo-Zea, C. Petitpierre: *A Simple, Modeling prone CORBA Architecture*

Software Defect Data and Predictability for Testing Schedules

Rattikorn Hewett & Aniruddha Kulkarni
Dept. of Comp. Sc., Texas Tech University
rattikorn.hewett@ttu.edu
aniruddha.kulkarni@ttu.edu

Catherine Stringfellow
Dept. of Comp. Sc.
Midwestern State University
catherine.stringfellow@mwsu.edu

Anneliese Andrews
Dept. of Comp. Sc.
University of Denver
andrews@cs.du.edu

Abstract

Software defect data are typically used in reliability modeling to predict the remaining number of defects in order to assess software quality and release decisions. However, in practice such decisions are often constrained by availability of resources. As software gets more complex, testing and fixing defects become difficult to schedule. This paper attempts to predict an estimated time for fixing software defects found during testing processes. We present an empirical approach that employs well-established data mining algorithms to construct predictive models from historical defect data. We evaluate the approach using a dataset of defect reports obtained from testing of a release of a large medical system. The accuracy obtained from our predictive models is as high as 93% despite the fact that not all relevant information was collected. The paper discusses detailed methods of experiments, results and their interpretations.

1. Introduction

Software testing requires rigorous efforts, which can be costly and time consuming. It can easily take 50% of a project life cycle. As software projects get larger and more complex, testing and fixing defects become difficult to schedule. Making decisions on suitable time for testing termination and software release often requires considerations of tradeoffs between cost, time and quality.

Software defect data have been used in reliability modeling for predicting the remaining number of defects in order to assess quality of software or to determine when to stop testing and release the software under test. Testing stops when reliability meets a certain requirement (i.e., number of defects is acceptably low), or the benefit from continuing testing cannot justify the testing cost. While this approach is useful, it has some limitations. First, decisions on testing activities involve three aspects of software testing processes: cost, time and quality but reliability is mainly concerned with the "quality" aspect. Second, although the number of defects is directly related to the time required for fixing the defects found, they do not necessarily scale in linear fashion. A large number of simple defects may take much less time to fix than a few sophisticated defects. Similarly, the time required to fix the defects could also depend on the experience and skills of the fixer. Finally, defect reports are often under utilized. Existing approaches to reliability modeling tend to exploit only quantitative measures. However, there are many factors, other than the number of defects, which could influence the time to fix the bugs. Many of these factors are qualitative. During the testing process, data about defects are documented in a software defect (bug) reports. These reports collect useful historical data about software defects including locations and types of defects, when they were found and fixed, and names of the testing and fixing teams. An approach to analysis that can utilize both quantitative and qualitative data in the defect report would be useful.

This paper proposes a novel approach to use software defect data for predicting a "time" aspect as opposed to the "quality" aspect of software testing process. In particular, we propose an approach to create a predictive model, from historical data of software defects, for predicting an estimated time to fix the defects during software testing. Such predictions are useful for scheduling testing and for avoiding overruns in software projects due to overly optimistic schedules.

Estimation of the time required to fix the defects is a difficult problem. We have to understand more about defects to be able to make predictions about fixing them. Causes of software defects include design flaws, implementation errors, ambiguous requirements, requirements change, etc. Defects can have varying degrees of complexity (which is proportional to the amount of time required to fix the defect) and severity (the extent to which the defect impairs the use of software). Furthermore, defects can be found by various groups and in various phases in the life cycle of software development. Defects found by the system testing group may be harder to fix than those found in an earlier stage of software life cycle. In some cases errors encountered by users may be difficult to reproduce. To make the matter worse, predicting time to fix defects is hard, as it is possible that fixing one problem may introduce a new problem into the system

This research proposes an empirical study to investigate the above problem by means of advances in data mining. Our main contributions include:

(1) formulation of a challenging new problem and approaches for the solution that are potentially useful

for scheduling and managing the software testing process

(2) empirical approaches to increase utilization of defect data by exploiting both quantitative and qualitative data factors

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 describes the defect data set followed by the details of the proposed approach in Section 4. Section 5 evaluates the proposed approach by experimentation on a real-world data set and comparing accuracy of results obtained from various predictive models using different data mining algorithms. The paper concludes in Section 6.

2. Related Work

To achieve effective management in testing process, much work has been done on using mathematical models for estimating "testing efforts," including the widely used method COCOMO [2]. However, testing efforts often consider only resources required for testing design and execution, and do not include the resources spent on fixing defects themselves [3]. Thus, estimating testing efforts is very different from our work, which involves estimating the time required for fixing defects.

Most of the work related to the utilization of software defect data has been in determining fault-prone components [1, 9, 14, 15] and in reliability modeling to predict the reliability of the software (the remaining number of defects) [5, 11, 12]. Reliability modeling makes limited use of the data as it only uses the number of defects found during different testing periods. Biyani and Santhanam [1] illustrated how defect data can be used to compare release qualities and relation between number of defects pre-release and post-release. In addition, empirical study in [9] uses the data to construct models to classify quality of software modules during software testing. Our work offers a new way in which one can make use of defect data. Furthermore, our models can incorporate both quantitative and qualitative values.

3. Data

To evaluate our approach, we experiment with a defect data set obtained from testing of three releases of a large medical record system as reported in [13]. The system initially contained 173 software components and 15 components were added over the three releases. Each data instance represents a defect with various corresponding attribute values including the defect report number, the release number, the phase in which the defect occurred (e.g., development, test, post-release), test site reporting the defect, the component and the report date.

Name	Description	Data Types
Prefix	The development group, which found the defect.	8 discrete values, e.g., development, system testing
Answer	Response from fixer indicating the type of the defect	17 discrete values, e.g., limitation, program defect, new function
Component	The component to which the defect belongs	75 discrete values
State	Status of the defect	5 discrete values, e.g., canceled, closed, verify
Severity	The severity of the defect	4 values: 1, 2, 3, 4 (1 = low, 4 = high)
OriginID	Person who found the defect	70 discrete values
OwnerID	Person who is fixing the defect	57 discrete values
AddDate	Date on which defect was added	Numeric
AssignDate	Date on which it was assigned to someone to work on	Numeric
Response Date	Date on which the fixer responded with an "answer" (attribute answer)	Numeric
EndDate	Date on which the defect was closed	Numeric
LastUpdate	This is the date on which the defect was last updated	Numeric

Table. 1 Data Characteristics.

For the study in this paper, we use data from release 1, which contains 1460 defects (or data instance or rows). There are a total of 30 attributes, 12 of which are selected to be relevant to our problem. Table 1 summarizes the attribute descriptions and their corresponding data types.

4. Approach

4.1 Problem formulation

Unlike most work in software quality, in this study we propose to predict the time required for fixing defects. For clarity, we describe relevant terms in our defect report related to different stages of a defect life cycle as summarized in a timeline in Figure 1. These terms are defined below in more details.

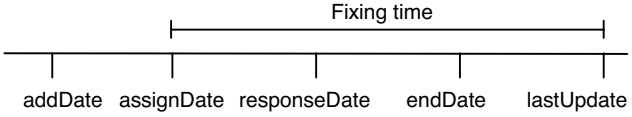


Fig. 1 Timeline for different dates in the data.

AddDate is the date on which the defect was found and added in the defect report. *AssignDate* represents the date on which a defect was assigned to be fixed. Because we are concerned with the time spent on fixing defects we can safely ignore the time spent in between addDate and assignDate. *Responsedate* is the date on which the person

assigned to fix the defect determines the suitable course of action with respect to the defect. *EndDate* specifies the end of the major fixing period that starts from the ResponseDate. *LastUpdate* signifies the end of the fixing process with respect to a particular defect. After the major fixing period if anymore problems are discovered, the developer fixes them and updates the endDate to the lastUpdate attribute.

A person assigned to fix a defect spends the time from assignDate till lastUpdate in fixing the defect. Our interest is to predict the time period from assignDate to lastUpdate as opposed to the period from addDate to lastUpdate. We ignore the period between addDate and assignDate, which is the time spent on dealing with management issues rather than the time that is actually spent on fixing the defect. Thus, our problem is to construct a predictive model that best fits a defect data set. The data set has a class attribute referred to as *fixing-time,* which represents a number of days required for fixing a particular defect until release. The fixing-time is computed by subtracting assignDate from lastUpdate. Because the responseDate and the endDate occur during the period that we want to predict, knowing their values would certainly affect the predicted value and consequently makes the prediction easier. Thus, using responseDate and endDate as condition attributes for predicting the fixing-time would be tantamount to cheating, as the predicted values are part of these values that are known prior to the prediction. As a result, for model prediction, in addition to the class attribute we employ all but the last three of selected relevant attributes (as shown in Table 1) in our analysis.

4.2 Relevant analysis issues

One common issue in data analysis in software domains is that the data is very scarce or when it is available it tends to be incomplete. Our problem is no exception. There are a number of factors that influence the time required to fix software defects including number of lines of code or modules changed or added, complexity of algorithms or logical controls or interactions, skills of fixers and testers, etc. Unfortunately, not all information about these factors is available in the defect report.

Another issue concerns the data value types, which include both quantitative and qualitative data. Most existing modeling approaches (e.g., statistical or time series techniques) employ quantitative models, where all variables have continuous or discrete values. However, many of the relevant factors in the defect report can be qualitative (e.g., names of person who fix the defect, component where the defect was found, etc.) A common approach to deal with this problem is to convert qualitative values into quantitative values. While this is useful in some cases, it imposes an order constraint on values and not all categorical values are ordinal. It is desirable to have an analysis technique that can directly analyze qualitative (or symbolic) as well as quantitative data.

Advanced research in data mining has produced many data analysis techniques that can analyze both quantitative and qualitative data including association rule mining, Naïve Bayes classification technique and decision tree learning [7, 10, 12, 16]. The decision tree learner is one of the most prominent machine learning techniques that been applied successfully in classification problems, where a class attribute has discrete value. Thus, viewing our problem as a classification problem, we need to have discrete class attribute values in order to be able to apply these modeling techniques (e.g., decision tree learning and Naïve Bayes technique). In particular, we need to discretize the class attribute, fixing-time. In this study, we use equal frequency bins to discretize the class fixing-time. The resulting values are in three categories: 0-59 days, 60-103 days and more than 103 days. This method reflects natural clusters of the data set and appears to be meaningful in real software practices. The large grain size of this fixing-time reflects the fact that in real practices a fixer may have to fix multiple defects in the same time period.

4.3 Approaches to data analysis

In this study, we employ four supervised learning algorithms based on three different approaches for constructing predictive models from a data set. We select these approaches since each is based on a different model representation and learning method as described below.

Decision tree learner [12] is one of the most popular methods in data mining. A decision tree describes a tree structure wherein leaves represent classifications (or predictions) and branches represent conjunctions of conditions that lead to those classifications. To construct a decision tree, the algorithm splits the data source set into subsets based on an attribute value of the splitting attribute (selected by a rank-based measure called ratio gain). This process repeats on each derived subset in a recursive manner until either splitting is not possible, or a singular classification is applied to each element of the derived subset. See more details for decision table learners in [10].

Naïve Bayes Classifier [7] is based on probability models that incorporate strong independence assumptions that often have no bearing in reality, hence are "naïve". The probability model can be derived using Bayes' theorem. Bayesian classifiers are called active learning, as they require little or no training time. The classification is computed based on the likelihood for each attribute belonging to a particular class. See more details in [7].

Neural net approach is based on a neural network, which is a set of connected input/output units (perceptrons) where each connection has a weight associated with it [8]. During the learning phase the network learns by adjusting the weights so as to be able to predict correct target values. Back propagation is one of the most popular neural net learning methods. The net learns by iteratively processing a set of training samples and comparing the networks prediction for each sample with the actual known class label. For each training sample, the weights are modified so as to minimize error between the predicted and the actual values. The weight modifications are made in backward direction that is from output layer to input layer, hence the name back propagation. The neural net approach generally takes a long time to train and requires parameters that are best determined empirically. They do, however, have a high tolerance to noisy data as well as the ability to fit complex (non-linear) patterns.

5. Experiments and Results

To validate our proposed idea described in Section 4, we perform experiments to create and evaluate predictive models for predicting estimated time to fix defects.

5.1 Data preprocessing

Our experiments involve three types of data preprocessing: *data selection, data conversion*, and *data discretization.*

Because we are interested in estimating the time to fix defects, we select only data points representing the defects that are caused by malfunctions or faulty components as opposed to those that are reported as defects due to misuse of test cases, fixing postponement or software limitation. In other words we consider only defects that are identified by fixers and are completely fixed (i.e., state attribute value is closed). After the data selection our data set remains with 1357 data points.

In data conversion, several attributes with date values are converted into numbers by subtracting all the dates from January 1, 1995. We chose this date partly because all the dates in the data start from 1996, and thus the resulting numbers are all positive, which are easier to deal with and to interpret. Interestingly, Microsoft Excel by default converts dates into numbers by subtracting the dates by January 1, 1900. If we used this default setting, the model accuracy obtained would not have been as accurate as the cutoff date we proposed. This is because the resulting numbers using the default cutoff date are so large that they reduce the power to differentiate different dates and their impacts on the fixing-time.

The final step is data discretization. Our defect data set contains both continuous and nominal values. Certain analysis algorithms (e.g., Naïve Bayes classifier) require continuous attribute values to be discretized. In this study, we apply two discretization methods: the *equal frequency bins* and the *entropy-based discretization* [4]. Unlike the binning approach, the entropy-based discretization uses class information for discretization and has shown to perform well [7]. For comparison purpose, we will also create models that do not require discretization.

5.2 Experimentation

We apply four data mining algorithms: *NaiveBayes, MultilayerPerceptron, DecisionTable* and *J48* (a variation of C4.5 [12]), provided by the data mining tool WEKA [16], as representative systems for the Naïve Bayes classifier, the Neural Net approach, and the decision tree learners, respectively. The *MultilayerPerceptron* is based on a back propagation learning algorithm described earlier. For our experiments we use a default setup of a network with two hidden layers, a learning rate of 0.3, a momentum of 0.2 and 30 iteration cycles as a bound for termination. The network configuration and parameters are obtained by using standard empirical procedures. See more details on relevant parameters and their meaning in [16].

To avoid overfitting, *n-fold cross-validation*, a standard re-sampling accuracy estimation technique, is used [10]. 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 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* (i.e., ratio of a number of correct predictions to a total number of test cases) obtained from 10-fold cross validation 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, the accuracy result is an average accuracy obtained when $n = 10$, as suggested in [10].

5.3 Attribute Selection

Although our set of relevant attributes is not extremely large, we apply a standard technique for attribute selection to maximize possible accuracy obtained. Our attribute selection technique is based on a *wrapper method* [6]. The method can be viewed as a greedy search to find a state associated with the subset of attributes that gives the highest heuristic evaluation function value (in hill-climbing, depth first search fashion). The heuristic function value here is the estimated future accuracy obtained from an average

accuracy from predictive models obtained from a 10-fold cross validation on the attribute set of the next state. The search starts from a state with an empty set of attributes, repeatedly moves to a state containing a larger set of attributes, and stops after a fixed number of node expansions does not yield a future state with a higher estimated accuracy than those of the current state.

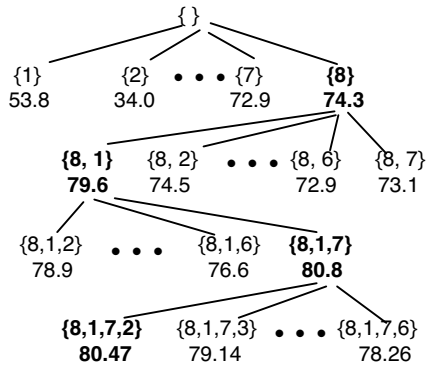


Fig. 2 Greedy search for the "best" set of attributes.

Figure 2 shows a partial search state space resulting from the wrapper method using the average accuracy obtained from decision tree models as the heuristic evaluation function. We apply the method to a total of eight relevant condition attributes (the top nine attributes of Table 1, according to the data selection, with an omission of the state attribute). Each state represents a set of attributes, each of which is represented by a number corresponding to its order in Table 1 (e.g., 1, 7 and 8 represent the attributes Prefix, AddDate and AssignDate, respectively). The best set at each search level is shown in bold. As shown in Figure 2, the optimal set of attributes obtained is {8, 1, 7}. We obtain the same set of attributes using predictive models learned from Naïve Bayes classifier.

5.4 Results

Table 2 shows average percentages of accuracy (over 10-fold cross validation) obtained from four different types of data analysis algorithms: Naive Bayes classifier, Decision Tree learner, decision table learner, and neural net approach. The ZeroR learner predicts values based on the majority of the class distribution and is commonly used as a measure of baseline performance. Table 2 compares results obtained with (bottom part of the table) and without (top part of the table) attribute selection using two different techniques for discretizing continuous attribute values. With an exception of the "Neural Net (cont. class)", all results are obtained using the data set with discrete class attribute values as discussed in Section 4.2. To give a fair assessment of a learning approach that can predict a continuous value (e.g., neural net approach), we include the results obtained by using the data set with continuous class values as well (e.g., the "Neural Net (cont. class)" shown in Table 2). The accuracy in such case is based on the number of correct predictions using the same range of the discrete intervals (i.e., 0-59 days, 60-103 days, more than 103 days) for original value and predicted value in order to compare with other learning approaches. Note that the entropy-based discretization requires class information (or labeled data), which is not available for the "Neural Net (cont. class)" case.

No attribute selection	**Discretization Method**		
Learning Approach	**None**	**Binning**	**Entropy-based**
NaiveBayes	77.22	74.42	**79.14**
Decision Tree	**93.51**	78.11	92.33
Decision Table	**92.4**	78.18	92.33
Neural Net	33.33	55.63	**59.32**
Neural Net (cont. class)	58.8	**65.58**	na
ZeroR	33.16	33.16	33.16

With attribute selection	**Discretization Method**		
Learning Approach	**None**	**Binning**	**Entropy-based**
NaiveBayes	80.84	76.78	**80.91**
Decision Tree	**93.44**	77.89	91.96
Decision Table	**92.4**	78.18	92.33
Neural Net	85.7	78.85	**89.16**
Neural Net (cont. class)	**83.79**	67.13	na
ZeroR	33.16	33.16	33.16

Table. 2 Average accuracy of predictions.

As shown in Table 2, with or without attribute selection, Naïve Bayes and neural net techniques perform best when entropy-based discretization is used. However, neural net with continuous class value has the best accuracy when using no discretization or equal frequency binning. In general, the results obtained with or without attribute selection are consistent except for those obtained from the neural nets approach, where the accuracy increases by close to 30% when a selected set of attributes is used. In fact, the model obtained by the decision tree algorithm has the highest accuracy of 93.5% followed by 92.4 % accuracy of the decision table model. Both of the top resulting models when all attributes are used are almost the same as those obtained when a selected set of attributes is used. Both use no discretization. However, the results obtained from decision tree and decision table learners using the entropy-based discretization are no more than 1.2 % less than the best two accuracies. Even though the accuracy obtained from the neural net model increases when we use a selected set of attributes, it still lags behind the top two approaches by about 3%. Using a selected set of attributes, the neural net approach with discrete class outperforms that with continuous class values. The overall results obtained from various algorithms are far better than an accuracy obtained from a random guess of 33.3% as indicated by the ZeroR approach in Table 2.

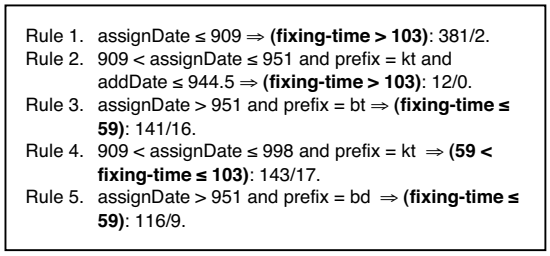


Fig. 3 Example rules from a predictive model.

Figure 3 illustrates examples of the rules extracted from a decision tree model constructed from overall 1357 training instances. Each rule represents a path from the root to a leaf. Our resulting model is a tree of size 31 with a total of 23 rules. The model has 93.5% accuracy on the training set and has a root mean squared error of 0.203. As shown in Figure 3, the end of each rule follows by ": *x*/*y*", where *x* and *y* represent a number of training instances (with the same condition as the rule condition) that are correctly and incorrectly predicted by the rule, respectively. Recall that each date is converted into number of days in a period between the date and a fixed cutoff date (January 1, 1995). Thus, if the assignDate is "small" then the defect was assigned "early" for fixing. For example, Rule 1 gives an implication that if the defect has been assigned "early," the fixing-time is likely to take longer than 103 days (with support evidences of about 28%). In practice, this may be the case that the fixer delays fixing defects as the deadline is not close or that the type of defects found early may be due to missing functionality that requires time to implement. Rule 2 suggests that if the defects were not assigned "late" and were found by a customer testing group, then the fixing-time is likely to be long. Rule 3 reflects the situation when the defect was assigned "late" and found by a system testing group then the fixing-time is likely to be short, i.e., no more than 59 days. The type of defects is not likely to be a major functional defect and therefore is likely to take less time to fix. Similarly, Rule 5 is concerned with the defect found by the developers. Thus, it is likely to be found in an early stage, which takes short time to fix. As shown in Figure 3, Rule 2 is the weakest of the five rules in terms of support degree as evidenced by the number of matching data instances.

6. Conclusion

We present a novel approach to utilize software defect data to create predictive models for forecasting the estimated time required for fixing defects by means of advances in data mining. Such estimation has potential benefits for planning and scheduling in software testing management. Although approaches to estimating testing efforts exist, estimation of time required for testing and fixing problems are two different problems. We illustrate an approach to the solution of the latter. The results of our predictive models obtained from various data mining algorithms are promising with an average of the best to be over 90%. However, like any other modeling approach that is based on historical data, our approach has some inherent limitations in that the resulting models can only be applied to software projects that share the same characteristics (i.e., schema or a set of condition attributes and the attribute domain). One remedy for this issue is to build predictive models from a set of defect data sets collected from different class of projects (e.g., organic, embedded as in COCOMO [2]). This should extend the generality of the model application.

References

[1] Biyani, S., P. Santhanam, "Exploring Defect Data from Development and Customer Usage on Software Modules over Multiple Releases," in *Procs of Int'l Conf. on Software Reliability Eng.,* Paderborn, Germany, pp. 316–320, 1998.

[2] Boehm, B., *Software Engineering Economics,* Prentice Hall, 1981.

[3] Culbertson, R., C. Brown and G. Cobb, *Rapid Testing,* Prentice Hall, Dec 29, 2001.

[4] Usama M. Fayyad and Keki B. Irani, "Multi-interval discretization of continuous valued attributes for classification learning, " in *Proceedings of IJCAI-93, volume 2,* Morgan Kaufmann Pub., pp. 1022-1027, 1993.

[5] N. E. Fenton, and M. Neil, "A critique of software defect prediction models," *IEEE Trans. Software Engineering,* 25(5): pp. 675–689, 1999.

[6] John, G.H., Kohavi, R., Pfleger, K., Irrelevant Features and the Subset Selection Problem, Proc. of the 11th International Conference on Machine Learning ICML94, pp. 121-129, 1994.

[7] Jiawei Han, and Micheline Kamber, Data Mining: Concepts and Techniques*,* Morgan Kaufmann*,* 2000.

[8] Simon Haykin, Neural Networks: A Comprehensive Foundation (2nd Edition), Springer, Springer-Verlag New York Inc June 1, 1995,

[9] Khoshgoftaar, T., R. Szabo, , T. Woodcock, , "an Empirical study of Program Quality During Testing and Maintenance," *Software Quality Journal,* 137-151, 1994.

[10] Kohavi, R., "The Power of Decision Tables," In European Conference on Machine Learning, Springer Verlag, 1995.

[11] Musa, J., A. Iannino and K. Okumoto, *Software reliability: measurement, prediction, application,* McGraw-Hill, Inc., New York, NY, 1987.

[12] Quinlan, R., *C4.5: Programs for Machine Learning,* Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[13] Stringfellow, C. and A. Andrews, "An empirical method for selecting software reliability growth models," *Empirical Software Engineering,* 7(4): pp. 319–343, 2002.

[14] Stringfellow, C. and A. von Mayhauser, "Deriving a Fault Architecture to Guide Testing," Software Quality Journal, 10(4), December, 2002, pp. 299-330.

[15] Stringfellow, C. and A. Anfrews, "Quantitiative Analysis of Development Defects to Guide Testing," Software Quality Journal, 9(3), November 2001, pp. 195-214.

[16] Witten, I. and E. Frank, *Data Mining practical Machine Learning Tools and Techniques,* Morgan Kaufmann, San Francisco, CA, 2005.

Using UML Designs to Generate OCL for Security Testing

Orest Pilskalns
School of Engineering and Computer Science
Washington State University
Vancouver, WA
orest@vancouver.wsu.edu

Anneliese Andrews
Department of Computer Science
University of Denver
Denver, CO
andrews@cs.du.edu

Abstract

Security attacks continually threaten networked systems. Early detection and correction of security flaws in the software design phase can reduce security risks. The majority of exploits occur by circumventing security rather than breaking encryptions. How can designers ensure that their modeling efforts will result in secure implementations? This paper describes an approach to systematically generate Object Constraint Language (OCL) security constraints from UML design models by employing security test criteria. The constraints can be used for development and test validation throughout the software lifecycle.

1 Introduction

Software engineers are beginning to recognize the importance of security at the design level [3]. Using formal methods to aid with security has become more common, yet many companies neither have the time nor the skill to employ this technique [4]. Many software projects employ the Unified Modeling Language (UML), to describe their system before implementation. Work has been done on generating Object Constraint Language (OCL) statements from UML Use-Cases for security purposes, yet the automation of security constraints from designs has not been addressed. The OCL is a language that describes expressions and constraints concerning object-oriented artifacts. Current practice for OCL creation relies on a manual analysis of the system. We provide an automated approach that uses security criteria and an UML designs to generate OCL that can be used for testing.

With the advent of agile methods a test driven approach to development is becoming much more common. OCL fits perfectly into this paradigm because it can be used for validating tests. Our research focuses on defining security criteria used for generating OCL artifacts. The OCL artifacts can be used for both code and design testing. We address the following research questions: (1) Can we define security based test adequacy criteria for UML designs? (2) Can we generate OCL from security test adequacy criteria and UML designs?

2 Background and Related Work

Work on UML security has generally focused on security extensions to the UML and integrating security aspects into UML designs. Jurjens [5] describes several extensions that can be added to the UML via stereotypes. These extensions include entities such as encrypted links, fair exchanges, and guarded access. Jurjens discusses the importance of disseminating security knowledge throughout a software product's lifecycle. The intent of Jurjens's work is to provide designers with better tools for specifying security in designs. However, it does not provide any tools for generating OCL.

Secure Coding: Principles and Practice [2] describes many common faults with code that are often exploited. This publication does not focus on design UML issues, however, it does present many ideas that could be helpful to designers. In Chapter 4.2, entitled "Architectural Principles" Graff explains several fundamental principles of a secure system. The following principles are adapted from [2] and listed below: (1) *Principle of least privilege.* A program should run with the minimum privilege necessary to complete its task. (2) *Principle of economy of mechanism.* The simplest design should be employed to allow for thorough testing. (3) *Principle of separation of privilege.* Use all appropriate mechanisms when available for allowing access. These principles can be applied to UML designs. In section 3, we will apply these principles to security criteria for UML designs.

3 Security Based Test Adequacy Criteria

Test adequacy criteria tell us when we have sufficiently tested a system. Security test adequacy criteria should be based on UML structural and behavioral models as well as

security principles. All of the following security criteria are based upon the premise that it is possible to use elements from both Sequence Diagrams as well as Class Diagrams. None of the criteria will necessarily differentiate between the UML diagrams; therefore the criteria should be applied to an integrated view. Integrated views are covered in section 4. In addition, we assume that objects are tagged as either remote or local.

The *principle of least privilege* is the first security principle we will apply to UML models. If a local object has access to a remote object, then it should only be able to access methods that are applicable to its designated activity. Often in web applications, a remote object is created with several available methods. An example is Java Server Page (JSP) that uses Enterprise Java Bean (EJB). The EJB may have several methods available for use, however, the JSP should probably only access a few of those methods. A malicious user could exploit the system by modifying the JSP on the local side to call methods not intended to be used by this particular JSP. This results in a security breach. Therefore, a test should execute all methods available on a remote object at any point during a sequence of events. Hence, our first security test adequacy criterion (c-1) can be stated in the following manner:

c-1 Given a test set T, a local object o and a remote object r, T should call every available method in r.

The *principle of least privilege* can also be applied to classes that contain methods that can be overridden. If a remote object is passed to a malicious user, the user may extend the remote object and override a class. This will allow the malicious user to change the functionality of the remote object.

c-2 Given a test set T, a remote object r, T should try to override every method in r.

The *principle of economy* is the second security principle we will apply to UML models. A design should be as simple as possible. The structural specification of a design should only contain multiplicity-pairs that are exercised in the behavioral views of the design. For example a Class Diagram may contain a multiplicity that allows several objects to be created; however, the Sequence Diagram shows that only one object is needed. By extending this to a web application example, it is possible for someone to modify a JSP to create several objects remotely, when it should only be able to create one object. This could potentially crash the server, causing it to reboot. If the defaults are not fail-safe, this could cause the remote server to enter a vulnerable state. Therefore, if a test suite, based on all path coverage of a Sequence Diagram, does not cover the multiplicity-pair criterion then the structural design needs to be modified.

c-3 Given a test set T, based on all message path coverage (see Section 2), and a System Model SM, T must cause each local to remote multiplicity-pair in SM to be created.

The *principle of separation of privilege* can be applied to sending message-objects between remote and local objects. No object that is created remotely and passed to a local object should be modified except through the specified methods. Here is an anecdote where this exploit was used. A book sales company uses a web application for sales. It passed an object to a remote client. It relied on the remote client not to modify the remote object manually. Of course, someone modified the sale price, allowing for books to be sold for pennies instead of their actual prices. Therefore, a test suite should try to modify any objects that are passed over a communication link. This leads to the following criterion:

c-4 Given a test set T, and two associated objects, one being local and the other remote, T should modify all methods in any objects passed between the local and remote objects, using the remote object to execute the modifications.

These test adequacy criteria answer our first research question. The test criteria are not comprehensive, but are a start to security testing UML. Applying these criteria will help show how they can be extended. In addition, new criteria can be created if designers and programmers share their experience and document how design flaws lead to exploits in practice.

4 Generating OCL Using Security Criteria and an Integrated Model

We use the security criteria to generate OCL constraints. Our approach requires a model that can effectively deal with both the behavior and structure of a design. Pilskalns et al. [6] provide a testing model that transforms Sequence Diagrams into an Object Method Directed Acyclic Graph (OMDAG). This approach uses test-cases to traverse different paths in the graph. Here we adapt this approach for OCL generation. Our approach to OCL generation consists of the following steps: (1) Build an integrated model (OMDAG) from Class and Sequence Diagrams. (2) Define a path based input table by using path coverage of the Sequence Diagram. (3) Define a path based input table based upon security criteria coverage of the OMDAG. (4) Identify the differences between the input tables and use the differences to generate OCL statements that can be used to constrain the design.

Sequence Diagrams show developers the sequence of events that need to take place when implementing a design.

The scenarios in the Sequence Diagram usually do not take into consideration security issues such as malicious users. The integrated model is used to generate test input for scenarios that can compromise the system. OCL statements are then generated to remove these scenarios by constraining the system. Therefore, they can be can be used in development and then later to validate tests.

4.1 Building the integrated model (OMDAG)

Building the integrated model consists of three steps. The first step maps Class Diagrams into tuples, called Class Tuples (CT). The CT consists of a class name, attributes (represented as tuples if non-primitive) from the class and super classes (if applicable), and methods (represented as tuples) for the class and super classes (if applicable). A class tuple is a mathematical representation of a class and is similar to the idea of representing a class using the XMI specification. The second step consists of mapping Sequence Diagrams into an Object Method Directed Acyclic Graph (OMDAG). The OMDAG maps the dynamic information in a Sequence Diagram to a directed acyclic graph. The OMDAG is created by mapping object and sequence method calls from a Sequence Diagram to vertices and arcs in a directed acyclic graph. The mapping between Sequence Diagram and OMDAG preserves the relationships in the Sequence Diagram. The mapping consists of (1) associating methods in the Sequence Diagram with their originating objects, (2) traversing the Sequence Diagram for the purpose of mapping successive method executions to edges of the OMDAG. These edges are also annotated with any conditions the Sequence Diagram may impose on their execution. The third step consists of combining each OMDAG with the CT information. This results in a modified Object Method Acyclic Graph (OMDAG). The OMDAG now represents the integrated model that combines Class Diagrams and Sequence Diagrams.

4.2 Object-Method Path Tables for Sequence Diagrams

All-path coverage for the Sequence Diagram can be obtained by finding all paths in the OMDAG. This can be accomplished by executing a depth first search on the OMDAG. The result is a sequence of object method calls for each path. This can be recorded in a table (see Table 1).

4.3 Object Method Path Tables for the OMDAG

Our approach, like partition testing, focuses on coverage via testing criteria. We use our security criteria to generate paths based on inputs that are not available when only using a Sequence Diagram. For example if we use the security criterion c-1 we must execute every available method when communicating with a remote object. All method information is not available in a Sequence Diagram. However, when we communicate with a remote object in the OMDAG we have the additional information provided by the class tuple (CT). This allows us to create method calls to every method publicly available in the object. For every security criteria we give a rule for discovering new traversable paths in the OMDAG. While conducting a depth first search there are calling objects and target objects associated with each method in the OMDAG. For security criteria c-1 we give the following rule:

r-1 While traversing an OMDAG for path discovery, a method call will be created from the calling object to the target object for each public method in a CT associated with the target.

For security criteria c-2 we give the following rule:

r-2 While traversing an OMDAG for path discovery, a method call will be created from the calling object to the target object that overrides each public method in a CT associated with the target.

For security criteria c-3 we give the following rule:

r-3 While traversing an OMDAG for path discovery, method calls will be created from the active local object to all associated remote objects according to the multiplicity located in the remote CT.

For security criteria c-4 we give the following rule:

r-4 While traversing an OMDAG for path discovery, whenever a method call passes an object to a remote object, the remote (target object) should exercise all public methods on the passed object.

4.4 Generating OCL

The object-method path tables for Sequence Diagrams will always be a subset of the object-method path table for the OMDAG. We assume that the Sequence Diagram paths are the desired paths. Therefore we remove from the OMDAG object-method table all Sequence Diagram paths. This results in partial paths generated by our security criteria rules. These paths should not be allowed to execute (thus we generate constraints) unless some behavioral diagram specifies that the communication is valid.

The next step requires that we create OCL constraints for each object-method-target path remaining in the table. OCL 2.0 introduces messages (previously referred to as action semantics) which can represent calling operations and sent signals. OCL 2.0 allows for creating a sequence of

Path 1			. . .			Path N		
Object	Method	Target	Object	Method	Target	Object	Method	Target
ob1	sort()	arrayList				ob1	equals()	ob2
ob1	getName()	arrayList				ob1	setName()	ob2
.								
.								
.								

Table 1. Example Object-Method Path Table .

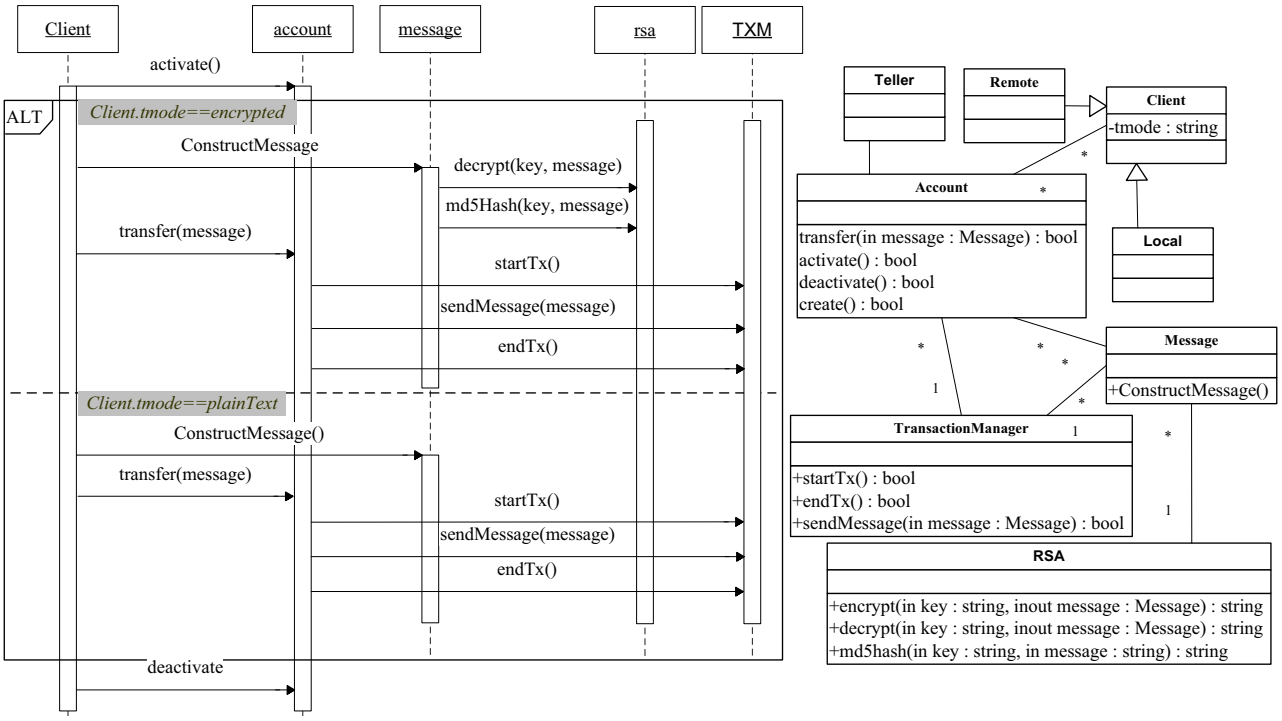


Figure 1. Banking System Sequence and Class Diagrams.

messages which can represent all messages sent to an object during the life of some contextual object. Contextual object is the object for which the invariant applies. This is accomplished using an OCL message operator "^^".

The following steps can convert each table path entry into an invariant:

1. **Create Context:** Each class of the target object can be the context. (A context should be a UML classifier.)

2. **Create Invariant:** The invariant needs to specify that the messages being called during the life of the target object are invalid.

   (a) Create an OCL variable called "message".

   (b) Assign to the "message" variable an OCL Sequence, representing all the specific messages invoked by the calling object.

   (c) Assert that this Sequence of messages should be empty or create a set that should be excluded from the Sequence.

If the object method table contains a calling object, $cObj$, a malicious method, $msg()$, and a target object, $tObj$, then the following invariant can be written using OCL syntax:

context: tObj: TargetObject
inv: let message: Sequence(OclMessage) = cObj^^msg() in
message→isEmpty()

Alternatively, if there is a sequence of messages that needs to be excluded, e.g. $msg1$, $msg2$, then the following form should be used:

context: tObj: TargetObject
inv: let message: Sequence(OclMessage) = cObj^^msg() in
message→excludes({msg1, msg2})

An OCL constraint can be generated for each invalid method call or calling sequence. Each path will result in one invariant that uses the message operation to specify an invalid message or message Sequence throughout the life of an object.

5 An Example

This example, adapted from [1], represents a design for a banking system. Figure 1 shows the Class Diagram and Sequence Diagrams. The diagrams outline the structural and behavioral aspects of a simplified banking system. The system contains two types of clients, a local client and a remote client. The clients interact with the system by sending messages. If the client is remote then the message needs to be encrypted and signed using an RSA encryption algorithm. Once the message is encrypted (if remote), it is transferred to the bank account using a transaction manager. The transaction manager can begin and end a transaction. A transaction such as a transfer is not finalized until the transaction manager indicates an end to the transaction. The designer of the system wants to make sure the system is secure. The Class Diagram contains one generalization where the $Remote$ and $Local$ classes inherit from the $Client$ class. The RSA class provides encryption (decrypt) and signing (md5hash). The $Account$ class contains the information about a banking client's account. The $TransactionManager$ class starts and stops transactions to provide protection against system failure during a transaction. If the transaction is not complete and the system fails, the transaction manager backs off the transaction. Two scenarios are represented as two paths in the Sequence Diagram. One scenario shows a client interacting with a bank account locally and the other shows a client interacting remotely.

5.1 Build An Integrated Model

The first step of the approach is to build an integrated model by combining the Class and Sequence Diagrams into an OMDAG. Each class from Figure 1 is mapped into a CT according to the mappings outlined in section 4.1. The lower right side of Figure 2 contains the CT information. The Sequence Diagram in Figure 1 is traversed and mapped into the OMDAG according to the procedure in section 4.1. The CT combined with the OMDAG provides an integrated model.

5.2 The Object Method Tables

An object method table needs to be built for an all-path coverage scenario. This can easily be derived from the vertex table in Figure 2. Table 2 contains an OMDAG path table using rules r-1 through r-4. We purposely made the table a subset of all path possibilities to make it readable. Table 2 contains a *create()* message from the remote client to an instance of a *BankAccount*. This was created by using rule r-1. Most likely the designer wanted only the teller to create new bank accounts, but unfortunately a malicious client can also create accounts. Path 2 in Table 2 contains a call to the *transfer()* method, however, the next method call is to *endTransaction()*. In this scenario the *transfer()* method was overridden due to rule r-2, causing a possible security problem. Path 3 in Table 2 contains two calls in succession from the remote client activating two *BankAccounts*. This was accomplished using rule r-3. Rule r-4 was not used since there were no objects passed that had more than one public method available.

5.3 Generating Constraints

Using the approach from section 4.4 and the table in Figure 2 it is possible to generate OCL constraints. The first constraint states that any method invocations to create a $BankAccount$ from a $Remote$ object is unacceptable.

context: b: BankAccount
inv: let message: Sequence(OclMessage) = remote^^create() in
message→isEmpty()

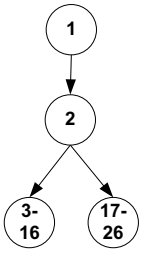
The next constraint states that all sequence of messages in the life of a $BankAccount$ cannot contain a sequence of methods where a $transfer()$ is immediately followed by a $endTransaction()$.

context: b:BankAccount
inv: let message: Sequence(OclMessage) = TxmManager^^endTransaction()
message→excludes({transfer, endTransaction})

The next constraint states that all sequence of messages in the life of a $BankAccount$ cannot contain a sequence of methods where there are more than one activation.

context: b:BankAccount
inv: let message: Sequence(OclMessage) = TxmManager^^activate()
message→excludes({activate, activate})

These constraints will provide extra security guidelines when implementing and testing the system.



Vertex = <Object Name, <Method, return type, visibility, condition, <parameters>>, CT>

Vertices →

1 <client, <activate, bool, +, null, <null>>, <Client>>

2 <account, <return>, <Account>>

3 <client, <Message, Message, +, <null>, [tmode==*Encypted*]>, <Client>>

4 <message, <decrypt, String, +, null, <{key, String}, {message, String}>>, <Message>>

5 <rsa, <return>, RSA>

6 <message, <md5hash, String, +, null, <{message, String}>>, <Message>>

7 <rsa, <return>, RSA>

8 <message, <return> Message>

9 <client, <transfer, bool, +, null, <{message, Message}>>, <Client>>

10 <account, <startTx, bool, +, null, <null>>, <Account>>

11 <TXM, <return>, <TransactionManager>>

12 <account, <sendMessage, bool, +, null, <{message, Message}>>, <Account>>

13 <TXM, <return>, <TransactionManager>>

14 <account, <endTx, bool, +, null, <null>>, <Account>>

15 <TXM, <return>, <TransactionManager>>

16 <account, <return>, <Account>>

17 <client, <Message, Message, +, <null>, [tmode==*PlainText*]>, <Client>>

18 <client, <transfer, bool, +, null, <{message, Message}>>, <Client>>

19 <account, <startTx, bool, +, null, <null>>, <Account>>

21 <TXM, <return>, <TransactionManager>>

22 <account, <sendMessage, bool, +, null, <{message, Message}>>, <Account>>

23 <TXM, <return>, <TransactionManager>>

24 <account, <endTx, bool, +, null, <null>>, <Account>>

25 <TXM, <return>, <TransactionManager>>

26 <account, <return>, <Account>>

CT = <class name, <Parent CT>, {<Attribute>}, {<Method>}>

CTs→

<RemoteClient, <Client>, <tmode>, <encrypt(String key, String message), decrypt(String key, String message)>>>
<LocalClient, <Client>, <tmode>, <null>>>
<Account, <null>, <null>, <activate, transfer(Message message)>>
<RSA, <null>, <null>, <encrypt, decrypt, md5hash(Message)>>
<Message, <null>, <null>, <Message>>
<TransactionManager, <null>, <null>, <endTx, startTx, sendMessage(Message)>

Figure 2. OMDAG

Path 1			Path 2		
Object	Method	Target	Object	Method	Target
r:Remote	create()	b:BankAccount	r:Remote	transfer()	b:BankAccount
			b:BankAccount	endTransaction()	txm:TransactionManager
Path 3			Path 4		
r:Remote	activate	b:BankAccount			
r:Remote	activate	b:BankAccount			

Table 2. Example OMDAG Path Table.

6 Conclusion

We have presented security test adequacy criteria that can be applied to testing UML Models. We have also adapted a UML testing approach to generate security constraints by using the test adequacy criteria. In addition we presented a simple example that showed how the criteria and OCL algorithm can be applied. We plan on adapting a UML design tool to automatically generate OCL constraints, which can be used for implementation and testing purposes.

References

[1] R. Binder, *Testing Object-Oriented Systems Models, Patterns, and Tools*, Object Technology Series, Addison Wesley, Reading, Massachusetts, 1999.

[2] M. Graff, *Secure Coding: Practice and Principles*, O'Reilly, 2003.

[3] P. Devanbu and S. Stubblebine, "Software Engineering for Security: a Roadmap", *Future of Software Engineering. ICSE 2000 Special Volume*, 2000.

[4] C. M. Holloway, R. W. Butler, "Impediments to Industrial Use of Formal Methods", *IEEE Computer*, pp. 25-26, April, 1996.

[5] J. Jurjens, "UMLsec: Extending UML for Secure Systems Development" *Proceedings from the 5th International Conference on the UML*, Dresden, Sept. 30 - Oct. 4, 2002.

[6] O. Pilskalns, A. Andrews, R. France, S. Ghosh, "Rigorous Testing by Merging Structural and Behavioral UML Representations", *Proceedings from the 6th International Conference on the UML*, pp 234-248, Oct 20–24, 2003.

Applying Mutation Testing to XML Schemas

Ledyvânia Franzotte
ledyvania@inf.ufpr.br

Silvia Regina Vergilio
silvia@inf.ufpr.br

Computer Science Department, CP: 19081, 81531-970
Federal University of Paraná (UFPR), Brazil

Abstract

XML language is largely used by Web-based applications to exchange data among different components. XML documents, in most cases, follow a grammar or schema that describes which elements and data types are expected by the application. These schemas are "translated" from specifications written in natural language, and consequently, in this process some mistakes are usually made. Because of this, faults can be introduced in the schemas and incorrect XML documents can be validated. This work proposes the use of mutation testing to reveal faults in schemas. A set of mutation operators is introduced by considering the most common errors present in the project of schemas. A process and a testing tool to support the operators are described. By using this tool some experiments were accomplished. Results from these experiments show the applicability of the operators, as well as, their efficacy to reveal faults.

1. Introduction

XML (eXtensible Markup Language) [12] is a markup language largely used by Web-based applications to exchange information among different components. All XML documents need to be well formed and valid. A well-formed document obeys XML syntax and this can be detected using a XML checker. A valid document follows rules defined in a schema that specifies the grammar expected for the document. Examples of schemas that are generally used are: DTD [2] and XML Schema [13].

Since developers are responsible to write the schemas, some mistakes can be made and incorrect schemas can be generated. A faulty schema can validate an incorrect XML document. If such incorrect document is used, this can cause a failure in the application that uses the schema.

Due to this scenario, the use of testing techniques and criteria to reveal faults in the schemas is fundamental. There are three basic testing techniques that can be used: functional, structural and fault-based. The fault based technique and criteria consider the main faults that are generally present in the programs being tested. The Mutation Analysis criterion is fault-based and has been considered as the most efficacious in terms of the number of revealed faults [11].

The idea of the Mutation Analysis criterion [1] is to insert small modifications in the program by creating new programs named mutants. Test data must be generated to kill all the created non-equivalent mutants. To kill a mutant it is necessary to execute a test input that produces different outputs for the mutant and for the original program. If there is no test case to distinguish the mutant from the original program, this mutant is called equivalent. At the end of the process, a mutation score is obtained. It is calculated according to the number of dead mutants and number of non-equivalent mutants generated.

There are in the literature some works that explore fault-based testing in the context of Web-based applications. The works described in [5, 7, 15] introduce mutation operators with the goal of testing the interaction between Web components, mainly web-services and have a different focus. In [3, 6], the authors explore types of faults in schemas. However, the type of faults introduced in both works is not used in a typical mutation testing. Emer et al, do not generate mutant schemas. Li and Miller do no apply the criterion. Some questions are not addressed, such as: "How to produce different behaviors for the mutant and original schema?" or "How to obtain the mutation score?"

Our work explores the use of Mutation Analysis for testing schemas, in the case, XML Schema that has become a very popular type of schema for XML documents. To allow the application of this criterion a set of operators and a supporting tool are necessary. Hence, we introduce a set of mutation operators based on the type of faults described by works from the literature, mentioned above. We also propose the use of XML documents as test data to be, or not, validated by the schemas. In this way, it is possible to kill the mutants and to obtain the mutation score to evaluate test cases. A tool, named XTM (Tool for **X**ML Schema **T**esting Based on **M**utation), was implemented. By using this tool, we accomplished two experiments to evaluate the proposed operators. The results allowed comparison with works from literature.

This paper is organized as follows: Section 2 shows related work. Section 3 introduces the mutation operators.

Section 4 describes the tool XTM and how to evaluate the mutants. Section 5 presents the results from the experiments and, Section 6 contains the conclusions and future work.

2. Related Work

There are in the literature some works that explore fault-based testing in the context of web applications [3, 5, 6, 7, 15]

Lee and Offutt [5] introduce two operators: *lenghtOf* and *memberOf* for XML documents. The created mutants are used to test the communication between components of a web application.

Offutt and Xu [7] explore the use of perturbation operators to test interaction in web-services. Data value and interaction perturbations are explored. Data perturbation operators modify data values by applying the idea of boundary value analysis. Interaction perturbation addresses RPC and data communications.

Xu et al [15] propose some mutation operators to schema perturbation, in this case, using XML Schema. The schema is represented by a tree T and operators that change sub-trees and nodes of T are introduced to create incorrect XML messages used in the test of web services.

The above mentioned works do not have the goal of testing schemas. The works described in [3, 6] address the test of XML Schema documents and have similar objective to our work.

Emer et al [3] introduce classes of faults and suggest the creation of mutant XML documents to be queried with the goal of revealing faults in XML Schema. The results from these queries are confronted with the schema specification. The authors do not apply the Mutation Analysis criterion and mutant schemas are not generated. They show results about efficacy by using four schemas of two systems.

Li and Miller [6] proposed 18 mutation operators for schemas. Each operator changes a value or an attribute and was specified based on the most common faults made by developers. These operators are described in Table 1. However, the authors do not address some questions related to the application of the Mutation Analysis criterion, such as: "How to kill a mutant?" or "How to obtain the mutation score and to evaluate a test set?". These questions are important because the goal of a testing criterion is to help the tester in the task of generation and evaluation of a test set. The authors do not evaluate the proposed operators with respect to efficacy. They only conduct an experiment to show that common used parsers do not reveal most faults described by their operators.

By analyzing the works mentioned in this section, we observe that the operators proposed by Li and Miller need to be evaluated. They do not consider some faults described by Emer et al, as well as faults described by the operators proposed to test web services, that can be also adapted to the context of schema testing. Because of this, in next section, we introduce a new set of operators and in Section 5, we present evaluation results.

Table 1 – Mutation operators proposed by Li and Miller[6]

Oper.	Description
AOC	Changes the value of attribute occurrence constraint.
SPC	Changes the regular expression declared by using the facet pattern.
RAR	Changes the value range declared.
EEC	Reduces or increases the number of arguments one by one.
SLC	Changes the value of the facets: *length, minLenght* and *maxLenght.*
NCC	Changes the value of the facet *fractionDigits* and *totalDigits*.
DTC	Replaces complex type deriving type modifiers.
UCR	Replaces the control modifier.
SNC	Changes XML Schema namespace
TDE	Exchanges the XML Schema target namespace and attribute
ENR	Replaces the namespace identifier of the element.
ENM	Removes the prefixed element namespace identifier.
QCR	Replaces the definition of the qualification of local elements and attributes.
QIR	Similar to QCR, but with control qualification on a declaration-by-declaration basis using the *form* element.
CCR	Replaces the complex type compositors.
COC	Changes the order of elements one by one.
CDD	Reduces the number of elements one by one.
EOC	Changes the value of an element occurrence constraint.

3. Mutation Operators

In this section, mutation operators to XML Schema documents are introduced. These operators were defined by considering the most common faults made by developers when writing schemas and the works described in last section.

The operators are divided in two groups that generate elementary and structural mutations. Each operator represents a possible fault and produces a simple change in the schema being tested. Each operator is illustrated with a fragment of a XML Schema document. For this fragment, possible mutations are presented.

The XML Schema structure will be represented as a tree T and, each schema element will be called node (terminal and non-terminal). Non-terminal nodes in T will also be called sub-tree.

3.1 Elementary Mutation Operators

These operators modify attributes values and elements in the document. This group contains the following operators:

- **Group_Order (GO):** changes the order in which the elements are expected to be. The operator domain is: {sequence, one, many, choice}
  Example:
  <group order="**ONE**">
  Possible mutations:
  <group order = "**SEQ**">;
  <group order= "**CHOICE**">;
  <group order= "**MANY**">
- **Required (REQ):** changes type of the attributes, if it must be obligatory or not. The operator domain is: {yes, no}
  Example:
  <attributetype name="bar" dt:type="int" required="**YES**"/>
  Possible mutations:
  <attributetype name="bar" dt:type="int" required="**NO**"/>
- **DataTypes (DT):** changes data type of elements and attributes. The operator domain contains all types defined, for instance: {integer, string, float, decimal, int, enumaration}.
  Example:
  <attributetype name="bar" dt:type="**INT**" required="yes"/>
  Possible mutations:
  <attributetype name = "bar" dt:type="**STRING**" required="yes"/>;
  <attributetype name="bar" dt:type="**FLOAT**" required="yes"/>
- **LenghtOf (LO):** changes the name size of the elements. Similar to *lengthOf* [5].
  Example:
  <element type="**A**">
  Possible mutations:
  <element type="**AXXX**">
- **ChangeSingPlural (CSP):** changes the element size by adding or removing the char 's' at the end of the string.
  Example:
  <element name="**A**">
  Possible mutations:
  <element name="**AS**">
- **ChangeTag (CTP):** changes the most common node tag used. Similar to: CCR, DTC and RAR [6].
  Example:
  <**attributetype** name="bar" dt:type="int" required="yes"/>
  Possible mutations:
  <**ELEMENTTYPE** name="bar" dt:type="int" required="yes"/>
- **SizeOccurs (SO):** changes the size of maximum and minimum occurrence, either from the fixed types (string) or from the types defined by the user. (Similar to EOC [6]).
  Example:
  <element type="B" minOccurs="**1**">
  Possible mutations:
  <element type="B" minOccurs="**0**">

3.2 Structure Mutation Operators

In this group, there are three operators to generate mutants by modifying the tree structure of the schema under test.

- **SubTree_Exchange (STE):** inverts the sub-trees below some node. Similar to: changeE [15] and COC [6]. Figure 1 shows a part of an XML as an example (original and mutant).

- **Insert_Tree (IT):** adds a node in the structure of the sub-tree. Similar to: insertN, insertND [15] and EEC [6]. Figure 2 shows a part of an XML as an example (original and mutant).

- **Delete_Tree (RT):** removes the node (or the sub-tree) from the structure of the tree. Similar to: deleteN, deleteND [15], CDD and EEC [6]. Figure 3 shows a part of an XML as an example (original and mutant).

4. Applying Mutation Testing

To apply the Mutation Analysis criterion and the operators introduced in last section, we propose the process presented in Figure 4. Mutation operators are used to perturb a given XML schema, and several mutants are generated. After this, the tester needs to provide a set of test data, XML documents. Each mutant schema as well as the original one are used to validate all the documents in the test set. A mutant is considered dead if the validation of a test case by using the mutant produces a different result from the validation of the same test case against the original schema.

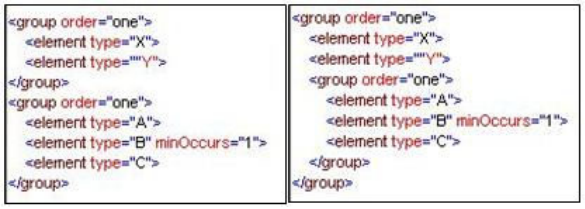


Figure 1: SubTree_Exchange example

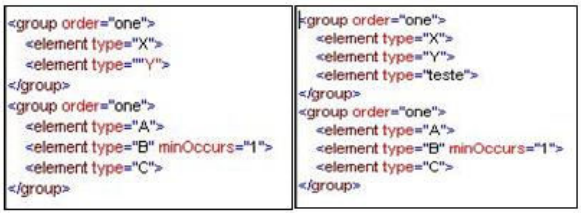


Figure 2: InsertTree example

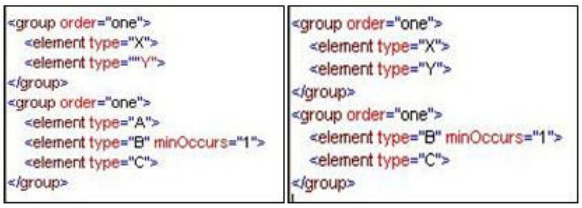


Figure 3: Delete_Tree example

At the end, the mutation score is obtained for the test set provided by the tester. The tester can use the mutation score to stop testing or to compare different test sets. If necessary, additional XML documents can be used to improve the score and to kill the remaining alive mutants. Among these mutants, there can be some ones that are equivalent to the original schema. The tester needs to determine the equivalence of schemas.

XTM (Tool for **X**ML Schema **T**esting Based on **M**utation) is a tool that supports the described process. It was developed in JAVA language using JDOM API [4] to read and manipulate the schemas. During the tree reading, the operators are applied and the generated mutants are validated using SAX API [8]. These schemas are used to validate the XML documents, used as test data.

XTM supports all the operators described in last section: GO, REQ, DT, LO, CSP, CTP, SO, STE, IT, RT. To allow comparisons with the work of Li and Miller [6], the following complementary operators were also implemented: SNC, QCR, AOC, QIR, CCR, UCR, SLC, RAR, NCC, DTC, SPC. The operator CTP was not individually implemented. The mutants generated by CTP are included in the set of mutants generated by CCR, DTC and RAR.

5. Experiments

We conducted two experiments to evaluate the proposed operators and the testing process implemented by XTM. To allow comparison with the approach proposed by Emer et al., we used the same schemas and systems mentioned in [3]: Enroll system (schemas: student.xsd (E1) and discipline.xsd (E2)) and Library system (schemas: books.xsd (E3) and users.xsd (E4)). These schemas, however, are simple. Because of this, we also use the schemas sia.xsd (E5) and sih.xsd (E6) from the real system of a hospital. Table 2 shows, for each schema, the total number of mutants generated by each operator implemented by XTM.

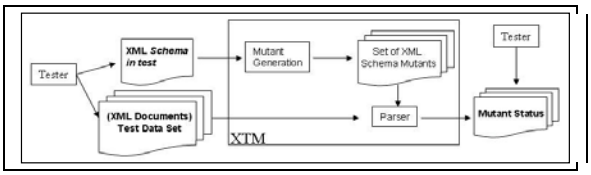


Figure 4: Using the Mutation Operators

The first experiment, also conducted by Li and Miller, shows that the most faults described by the operators produce valid schemas and are not detected by most parsers. The second one evaluates the operators and the testing process, according to cost and efficacy.

5.1 Experiment 1

The goal of this experiment is to show that usual schema parsers are not able to identify the faults described by the proposed operators. To do this, the same methodology described in [6] was used. All the generated mutants were submitted to the following parsers: W3C [14], XMLSpy [9] e Stylus Studio [10]. As a result of this step, the mean percentage of valid mutants was obtained by considering all the parsers. Table 3 shows these percentages for each schema and operator. In this table, the results of the elementary mutation operators (Group 1) are grouped in column G1.

The total (Tot1) presented in Table 2 does not consider the document validation. The total (Tot2) presents the number of valid mutants according to W3C [14] parser. In this table, the first ten operators were proposed by Li and Miller [6] and the other ones were proposed in this work. Observe that, it was not possible to apply some operators proposed by Li and Miller to all the schemas. According to Table 2, most mutants were generated with the operators proposed in this work. This is due to the structure of the used schemas, that are simple. A better analysis should be done to verify the impact of these operators and evaluate the frequency of use of these structures in the schemas.

Considering Table 3, we observe that around 60% of the mutants are valid. This means that the major faults they represent are not revealed by the parsers commonly used to check XML documents.

The InsertTree operator generated a large number of invalid mutants due to its implementation characteristics. A node is inserted without analysis of its use context, generating, in this way, several invalid mutants. This operator needs improvement to avoid this.

Table 2: Mutants Generated by Mutation Operator

Oper.	XML Schema						Tot1	Tot2
	E1	E2	E3	E4	E5	E6		
SNC	1	1	1	1	1	1	6	0
QCR	1	1	1	1	1	1	6	6
AOC	0	0	0	0	0	0	0	0
QIR	0	0	0	0	0	0	0	0
CCR	4	4	6	4	16	16	50	50
UCR	0	0	0	0	0	0	0	0
SLC	0	0	0	0	0	0	0	0
RAR	0	0	0	0	3	3	6	6
NCC	0	0	0	0	0	0	0	0
DTC	0	0	0	0	1	1	2	2
SPC	0	0	0	0	0	0	0	0
GO	0	0	0	0	0	0	0	0
CSP	10	10	11	8	36	34	109	107
REQ	0	0	0	0	0	0	0	0
DT	12	12	10	8	52	48	142	139
SO	23	20	32	25	80	75	255	218
LO	10	10	11	8	35	36	110	110
IT	22	28	28	20	80	83	261	62
STE	28	28	28	15	77	96	271	268
RT	4	15	15	12	54	57	157	153
Total	115	129	143	102	436	451	1376	1021

5.2 Experiment II

The goal of the proposed operators is to describe faults related to semantic and not just syntactic faults. As shown in Experiment I, the faults are not revealed by just using parsers. To reveal the faults described by the operators it is necessary to apply the mutation testing, which was done in a second experiment, named Experiment II.

For each schema, only the mutants that were validated by the parser implemented by XTM were used (mechanism using SAX API [8]). Table 4 presents the number of valid mutants generated for each schema, number of necessary test cases to kill them and the obtained score. All the generated mutants were dead (Score=1), that means that it was not generated equivalent mutants.

To evaluate the efficacy of the operators, we considered the same faults revealed by the approach proposed by Emer et al [3] for schemas E1, E2, E3 and E4. For this experiment, a fault was seeded in the schema E5. The number of known and revealed faults is also presented in Table 4.

Table 3: Percentage of Valid Mutants

XML Schema	% Valid Mutants Media				
	G1	SBT	IT	RT	Total
E1	60.10	100	32,14	100	68,24
E2	68.39	100	14,29	100	72,83
E3	68.12	100	14,29	100	71,97
E4	67.23	100	15	100	69,15
E5	65.57	100	12,50	96.3	69,62
E6	59.43	100	14,46	96,49	74,62
Total	64.80	100	17,11	98,8	71,02

Table 4: Results from Experiment II

XML Schemas	Valid Mutants	Score	Number of Test Cases	Revealed/ Known Faults
E1	123	1	3	10/10
E2	120	1	4	8/8
E3	133	1	3	4/4
E4	100	1	3	9/9
E5	420	1	4	0/0
E6	455	1	4	1/1

The type of revealed faults of Enroll and Library systems were about: size, patterns and fields data type, like login and password, occurrence size and limit values allowed. The fault of Schema E6 is related to the limit of a created data type.

We can observe, that all the known faults could be detected by the test cases. The same faults revealed by the approach of Emer et al. were discovered with a total of 21 test cases. Emer et al. use 22 queries for testing only schema E1. Beside this, 31 faults, out 32, were discovered by using the operators proposed in this paper. The implemented operators proposed by Li and Miller [6] revealed only one fault presented on E6. This fault was detected by the operator RAR and it is the same fault not detected by the operators proposed in this work.

6. Conclusions

In this work, it was defined a testing process for XML Schema documents based on the Mutation Analysis criterion. A set of mutation operators for XML Schema was proposed to generate diverse mutant schemas. XML documents are used as test data to kill the mutants. If a mutant validates a document and the original schema does not, or if the mutant does not validate the document, but the original schema does, the mutant schema is considered dead. In this way, the mutation score can be calculated to evaluate the testing activity or to compare test sets.

The proposed operators modify the schema under test by considering two dimensions: data and structure. They are based on typical faults that can be generally present in the schemas.

To allow the application of the criterion, a tool, named XTM was implemented. This tool supports the introduced operators and other ones found in the literature. By using this tool two experiments were accomplished.

The first experiment shows that the most faults described by the operators (around 60%) were not detected by using parsers, commonly used to check XML documents. These faults when present in the schemas allow that incorrect XML documents are exchanged and this can cause a failure in the applications.

In the second experiment, the operators implemented by XTM revealed all the known faults. We observed that the operators proposed in this paper are capable of revealing a great number of faults and can be considered as complementary to the ones proposed in works from the literature.

The application of mutation testing can be an expensive process. It is directly proportional to the size and to the complexity of the schema under test. The use of a tool like XTM is fundamental, as well as to choose the operators to be applied. We also observe that some operators did not generate any mutant. We are now implementing some mechanisms in XTM to enable operators according to the characteristics of the schema being tested. This helps to decrease the number of generated mutants. Other experiments should be conducted to better evaluate these characteristics and the operators.

The proposed operators should be improved and new ones should be proposed. The operator InsertTree needs to be redefined to consider the context in which the insertion should be made. In addition, these operators can also be used in other contexts, such as the communication test of Web Services.

References

[1] DEMILLO, R. A; LIPTON, R. J. Hints on test data selection: help for practicing programmer. IEEE Computer, v. 11, n. 4, p. 34-41, April, 1978.

[2] W3C. Document Type Definition. Available: http://www.w3schools.com/dtd/default.asp. Access: February, 2005.

[3] EMER, M. C. F. P. and VERGILIO, S. R. and JINO, M. A Testing Approach for XML Schemas. In: The 29th Annual International Computer Software and Applications Conference, COMPSAC 2005 - QATWBA 2005, July, 2005.

[4] JDOM, JAVA DOM. Available: http://www.jdom.org/. Access: August, 2005.

[5] LEE, Suet Chun. OFFUTT, Jeff, Generating Test Cases for XML-based Web Component Interaction Using Mutation Analysis. In: Proceedings of the 12th International Symposium o Software Reliability Engineering, p. 200-209, Hong Kong, China, November, 2001.

[6] LI, Jian Bing, MILLER, James. Testing the Semantics of W3C XML Schema. In: he 29th Annual International Computer Software and Applications Conference, COMPSAC 2005 - QATWBA 2005, July, 2005.

[7] OFFUTT, J. and XU, W. Generating Test Cases for Web Services Using Data Perturbation. In: TAV-WEB Proceedings, September, 2004.

[8] SAX Project. Available: http://www.saxproject.org/. Acess: February, 2005.

[9] XMLSpy2005. Available: http://www.altova.com/products_ide.html. Acess: November, 2005.

[10] Stylus Studio 2006. Available: http://www.stylusstudio.com/xml_download.html. Acess: November, 2005.

[11] WONG, W.E. and MATHUR, A.P. and MALDONADO, J.C. Mutation Versus All-uses: An Empirical Evaluation of Cost, Strength and Effectiveness. In: Software Quality and Productivity – Theory, Practice, Education and Training, Hong Kong, December, 1994.

[12] W3C. Extensible Markup Language (XML) 1.0 (second edition) – W3C recommendation, October 2000. Available: http://www.w3.org/XML. Access: January, 2005.

[13] W3C. XML Schema recommendation, May, 2001. http://www.w3.org/tr/. Access: January, 2005.

[14] W3C Validator for XML Schema. Available: http://www.w3.org/2001/03/webdata/xsv. Acess: November, 2005.

[15] XU, Wuzhi, OFFUTT, Jeff, LUO, Juan. Testing Web Services by XML Perturbation. In: Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering, November, 2005.

A new heuristic for test suite generation for pair-wise testing

Nie Changhai, Xu Baowen, Shi Liang, and Wang Ziyuan

***Abstract*—Pair-wise testing is a practical and effective method to test the software with a test suite that covers all pair-wise combinations of parameter values; much research has been aimed at the test suite generation. In this paper, we present a new heuristic for the test suite generation based on solution space tree model, in which each usable test data is represented as a path and all the usable test data form a tree, called solution space tree, then the test suite generation is to search a subset of paths by using depth-first search from the tree. We have implemented this method as a tool, and the result shows that it has good merits and can be a complement of the existed methods and tools.**

***Index Terms*—software testing, combination coverage, test suite generation, software engineering**

I. Introduction

Combinatorial testing is an important method for software testing, which detects the faults caused by some factor or the interactions of some factors in the software system, it selects a minimal test suite by the need to cover all the factors and their combinations to the best degree, the selected test suite is representative and can reflect some internal rule exposed by the exhaustive testing to some extend. So this method is effective and has strong detecting ability to the software defects, especially to the high integrity software.

Combinatorial testing is also called *t*-way interaction testing or *t*-way covering design for testing, and the number of test cases required to test all *t*-way parameter combinations grows logarithmically in the number of parameters. It is pair-wise testing when t=2, pair-wise testing requires a test suite to cover all the pair-wise combinations of test parameters specified in the formal test requirements. As the trade-off between the thoroughness of test combination coverage and the availability of limited resources, such as time, human resource, cost and etc, pair-wise testing is practical and widely used, it has found many faults those are difficult to detect for the traditional testing methods [1]-[7].

This work was supported in part by the National Natural Science Foundation of China (60425206, 60373066, 60403016), Natural Science Foundation of Jiangsu Province (BK2005060), High Technology Research Project of Jiangsu Province (BG2005032).

Nie Changhai, Xu Baowen, Shi Liang, and Wang Ziyuan are with the School of Computer Science and Engineering, Southeast University, Nanjing, 210096, China. They are also with the Jiangsu Institute of Software Quality, Nanjing, 210096, China (e-mail: {changhainie, bwxu, shiliang_ada, wangziyuan}@seu.du.cn).

Much research has been done in the issue of test suite generation for pair-wise testing [8]-[16], but it is still not well solved today. In this paper, we first give a model for combinatorial testing, and then represent all the usable test data as a tree, where each path from the root to the leaf is a test data. Then select a minimal subset of all the paths from the tree with backtrack method as pair-wise test suite. We have implemented it as a tool, and the result shows that it has some good properties and merits, and it can be a complement of the existed methods and tools.

The remainder of this paper is organized as follows: section 2 describe the model for combinatorial testing and introduce some of the basic concepts. Section 3 reviews the related works. In section 4, we give a model and the heuristic algorithm for pair-wise test generation, and then discuss its properties. Section 5 compares our heuristic with the existed methods. Moreover, we suggest further possible extension of this work in section 6.

II. The Model for Combinatorial Testing and Related Concepts

Suppose software under test SUT has n parameters c_1, c_2,…, c_n, which may be configuration parameters, internal events, user input parameters, external events and etc that can influence it. Let each of parameters c_i has a_i discrete values in the finite set T_i, $a_i = |T_i|$, and let $a_1 \geq a_2 \geq \ldots \geq a_n$ where all the parameters are independent, that is, any of the parameter values is not determined by the others.

Definition 1 A n-tuple(v_1, v_2,…, v_n) ($v_1 \in T_1$, $v_2 \in T_2$,…, $v_n \in T_n$) is a **test data** of SUT.

Definition 2 Let t_1 and t_2 are two test data for SUT, if there are b same values in the b same positions of the two n-tuples, we call that the **overlap degree** of the two test data is b.

For example, (2,1,3,2,1,3) and (3,1,2,2,1,1) are two test data and their overlap degree is 3. When the overlap degree of the two test data has the property that $b \geq 2$, the combination of the b parameter values is covered by the both two test data. None of the combinations is covered by both the test data when $b \leq 1$.

Combinatorial testing is mainly used to detect the faults trigged by system parameters and their interactions with a test suite generated by combinatorial design approach. As it will need $a_1 \times a_2 \times \ldots \times a_n$ test data for the exhaustive testing to SUT and this will be impractical, we should reduce the whole usable test suite to *t*-way covering test suite based the test requirements

of SUT.

Definition 3 Suppose A is an $m\times n$ array, $A=(a_{i,j})_{m\times n}$, elements of the j-th column is values of parameter c_j in SUT, that is $a_{ij}\in T_j$, where $j = 1,2\ldots n$. If every two column of A, say column p and q, contains all the ordered pairs formed by a_p symbols in T_p and a_q symbols in T_q with same times, then A is **an orthogonal array**, denoted by OA(m, $a_1\times a_2\times\ldots\times a_n$); If it only contains all the pairs, then A is called a **pair-wise covering array**, Denoted by CA(m, $a_1\times a_2\times\ldots\times a_n$). Where every row of A is a test data and m is the number of the needed test data. When m is the least, that is, the pair-wise covering array A contains the least number of rows, A is called the **smallest pair-wise covering array**.

Definition 4 The method of generating test suite by orthogonal array is called **orthogonal experiment design**.

Definition 5 The method of generating test suite by pair-wise covering array is called **pair-wise testing** or **2-way interaction testing**.

From the definitions we can know that an orthogonal array is a kind of pair-wise covering array, the reverse is not right. Test suite needed by pair-wise testing is generally far smaller than the orthogonal experiment design needed [3]. For example, for 100 parameters with two values each, the orthogonal array requires 128 tests, while 10 test cases are sufficient to cover all pairs. In software testing, it is only necessary to cover the combinations of parameter values once and not necessary to cover them with the same number of times. So pair-wise testing is more practical, it can improve the efficiency and decrease the cost of software testing with the smaller test suite.

III. Related Work

Since we like to minimize the testing cost as much as possible, we are interested in generating the least test suite for pair-wise testing, known as the smallest pair-wise covering array. However, the problem of finding the smallest pair-wise covering array is NP-complete. There are two main pragmatic approaches towards the problem. One is the algebraic approach. Various algebraic have been proposed for finding the smallest pair-wise covering array.

The original approach is to use orthogonal arrays [1]-[4], but orthogonal arrays have a balance requirement that every pair is covered the same number of times, and this requirement make it impractical for software testing. A. W. Williams presented a construction method based on some basic blocks, and developed a new, fast, deterministic algorithm for achieving pair-wise interaction coverage [16]. Noritaka Kobayashi et al. also propose a new algebraic construction and give an upper bound on the size of test set generated. The results show that the proposed construction can generate very small 2-factor covering designs [13]. Although these algebraic constructions are very effective when all parameters have the same number of values, they cannot well deal with the case where parameters have different numbers of values.

Another approach is to use the heuristics method. D. M. Cohen et al. proposed a heuristic search-based approach, which has been implemented as a test generation system, called AETG [8]-[10]. K. C. Tai and Y. Lei proposed a new test generation strategy, called in-parameter-order (or IPO), for pair-wise testing, and they have also implemented it as a tool, called PairTest [11], [12]. M. B. Cohen also gives a way to generating test suite with simulated annealing for variable strength arrays, where pair-wise covering array is a specific case of the variable strength arrays [21]-[23]. Generally test sets generated by these approaches tend to be larger than those generated by the algebraic methods, and they cannot guarantee bounds on the size of resulting test sets.

The performance of the algebraic methods and the heuristics were studied and compared in [16]. It is found that none of the methods always outperforms the other in delivering the smallest sized test sets. Instead they can work as complementary for each other.

IV. The New Heuristic Based on Solution Space Tree Model

Next we present a new heuristic, which is based on solution space model.

A. Solution space tree model

For the software under test SUT which has n parameters and each parameter c_i has a_i values where we can let $a_1\geq a_2\geq\ldots\geq a_n$ without loss of generality, each test data can be represented as a path from root to leaf in the tree and all the usable test data forms a tree as follows: The root of the tree has a_1 child branches which represent a_1 values of parameter c_1 respectively; each root in the second level of the tree has a_2 child branches which represent a_2 values of parameter c_2 respectively; …; each root in the nth level of the tree has a_n child branches which represent a_n values of parameter c_n respectively. For example, when n=3 and $a_1=a_2=a_3=3$ to a SUT, all the usable test data forms a tree as figure 1.

Test suite generation for pair-wise covering array is to search a subset of paths from the solution space tree. For example, the 9 paths: 111, 122, 133, 212, 223, 231, 313, 321 and 332 form a test suite: {(1,1,1), (1,2,2), (1,3,3), (2,1,2), (2,2,3), (2,3,1), (3,1,3), (3,2,1), (3,3,2)}, which satisfies the requirement of pair-wise covering and is the smallest pair-wise covering array.

B. The new heuristic based on solution space tree model

The algorithm consists of four steps based on the above model.

Step 1, Testers assign a test set TS which they take care of, or think it may trigger failures. If testers don't assign then TS= $\varnothing$.

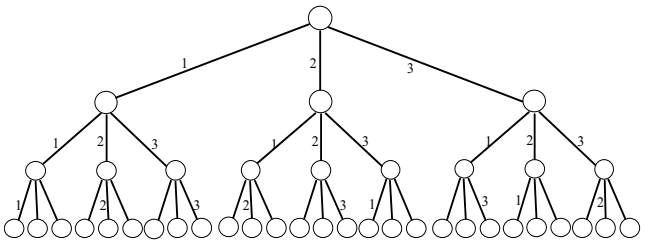


Fig. 1. The solution space tree when n=3,$a_1=a_2=a_3$=3

ALGORITHM 1

SEARCH THE SOLUTION SPACE TREE USING DEPTH-FIRST SEARCH

```
class Pairwisect{
    friend int nPairwisect(int);
private:
    bool Place(int k);
    void Backtrack(int t);
    int  n;    //the number of parameters
    int  *T;  //the number of values for each parameter
    int  *x;  //current solution
    int  sum;   //the number of test data that have been found
    int  **ct;   //covering array
};
bool Place(int k)
{
    for (int j=1;j<sum;j++){
      int overlap=0;
      for (i=1,i=k,i++)
           if (ct[j,i]=x[i]){
               overlap+=1;
               if (overlap>1) return false;
           }
      return true;
    }
}
void Pairwisect::Backtrack(int t)
{
    if (t>n){for (i=1;i<n+1;i++)ct[sum,i]=x[i]; sum++;}
    else
        for (int j=1;j<=T[t];j++){
            x[t]=j;
            if (place(t)) Backtrack(t+1);
        }
}
int nPairwisect(int)    //search the solution space tree
{
    Pairwisect X;       //initialize
    X.n=n; X.sum=0;
    int **c=new int[n*n][n];
    int *p=new int[n+1];
    for (i=1,i<=n,i++)
      {p[i]=0; for (j=1,j<=n*n,j++)c[j,i]=0;}
    X.x=p; X.ct=c;
    X.Backtrack(1);
    delete []p;
    Return X.ct;
}
```

Step 2, Search the solution space tree with backtrack method using depth-first search to find each test data *t*, which overlap degree with the existed test data in TS is no more than 1, and put *t* into TS, until the search is end. The detail procedure can be seen in algorithm 1.

Step 3, Check whether all the test data in TS can cover all the pair-wise combinations of parameter values. If it satisfies, then algorithm ends, else lists all the pairs those are not covered by TS. It will take running time of $O(n^2)$ to do this step.

Step 4, Construct test data with greedy algorithm to cover all the pairs those are not covered by TS [8]. The detail is described in algorithm 2.

ALGORITHM 2

ADDITIONAL TEST DATA GENERATION

Assume there have been r test data in TS for SUT and then we select the r+1 to cover the most new pairs those are not covered by TS. The test data is selected by the following greedy algorithm:

1. Chooses the value *v* of parameter such that the parameter value appears in the greatest number of uncovered pairs.
2. Assume that values have been selected for some j parameters where $1 \le j \le n$, then select the j+1th parameter value that appeared in the greatest number of new pairs. Until all the n parameter values are determined.
3. Output the test data into TS and delete all the pairs covered by it from the list of uncovered pairs. If the list is empty then end else go to the step 1.

C. *Properties of the algorithm*

In the algorithm model there is a one-to-one correspondence between all the paths in the solution space tree and all the usable test data for SUT. The test data generation for pair-wise testing is to search a subset of paths from the solution space tree. The algorithm has the following properties:

Proposition 1 The heuristic algorithm base on the solution space tree model can generate test suite for pair-wise testing on the basis of an assigned test data set by the testers. Such that the generated test suite not only satisfy the intention of testers, but also satisfy the requirement of pair-wise testing.

Proposition 2 For a software under test SUT, assume the test set assigned by testers TS= $\varnothing$,if $a_1=a_2=\ldots=a_n=r=p$(or p^m),$n \le p+1$(or p^m+1),where p is a prime, then the test suit generated by algorithm 1 is an orthogonal array OA(r^2, r^n).

Proof: By the construction theory of orthogonal array [1], for SUT when $a_1=a_2=\ldots=a_n=r$($r=p$ or $r=p^m$), $n \le p+1$(p^m+1), where p is a prime, there exists an orthogonal array OA(r^2,r^n). OA(r^2,r^n) is a $r^2 \times n$ matrix and every row of it is a test data, so the overlap degree between any two of the test data in OA(r^2,r^n) is no more than 1, otherwise, there must exists two test data, the overlap degree is more than 1 and is equal to 2 or more, then the two test data cover a pair of parameter value combination twice. Since there are r^2 distinct pairs between any two parameters, there should be also at least r^2 test data to cover them. So there must exist a pair uncovered by the test suite generated from OA(r^2,r^n).

Corresponding to the solution space tree, there exist r^2 paths that their overlap degree to each other is no more than 1. Algorithm 1 is used to search all the paths with the overlap degree of no more than 1 to each other. So the test suite generated by algorithm 1 is an orthogonal array OA(r^2,r^n).▌

V. COMPARISON WITH EXISTED TOOLS

The tools of AETG and PairTest can both generate pair-wise test suite effectively, but they cannot assure the smallest size of

TABLE 2 SIZES OF PAIRWISE TEST SET GENERATED BY PAIRTEST AND HSST

Number of parameters	10	20	30	40	50	60	70	80	90	100
Tests generated by PairTest	31	40	46	49	52	57	60	62	62	66
Tests generated by HSST	31	40	44	49	50	54	56	56	58	58

In table 2 every parameter is 4-value.

TABLE 1
SIZES OF PAIR-WISE TEST SET GENERATED BY PAIRTEST, AETG AND HSST

system	S1	S2	S3	S4	S5	S6
PairTest	9	19	29	29	15	37
AETG	9	15	31	28	10	31
HSST	9	20	21	30	16	35

Where:

S1 4 3-value parameters; S2 13 3-value parameters;
S3 17 parameters(12 3-value parameters, 5 4-value parameters);
S4 75 parameters(1 4-value parameters, 39 3-value parameters, 35 2-value parameters);
S5 100 2-value parameters;
S6 9 parameters (3 5-value parameters, 3 4-value parameters, 1 3-value parameters, 2 2-value parameters);

the generated test suite, especially when each parameter has the same number of values in SUT. Although the algebraic methods proposed by A. W. Williams and Noritaka Kobayashi can well deal with this case, they cannot do so when parameters have different numbers of values.

The method in this paper is very intuitive which is based on the solution space tree model and make a systematic search with backtrack. Although it is also a heuristic, it combines the merits of the heuristics and the algebraic methods. It can not only well deal with the case when parameters have different numbers of values, but also can generate the least pair-wise array when each parameter has p or p^n values and p is a prime. The method also generate pair-wise test suite by extending the test suite assigned by testers, so it has some pertinence and agility.

The heuristic based on solution space tree model is a new exploring to generate pair-wise test suite, it has made some improvement in some aspects, but it also has some disadvantages. It can work as a complement to the existed methods, when all of the methods work together, they can deliver a smaller test suite. We have implemented the heuristic algorithm as a tool, named HSST (heuristic based on solution space tree) and also implemented the existed methods; next we make some comparisons between HSST and the existed tools, mainly AETG and PairTest [8], [12]. The results of comparison show that none of the tools always outperforms the other in delivering the smallest sized test suite, and they can be complementary for each other. The details of comparison can be seen in table 1 and table 2.

The size of pair-wise test set generated by PairTest for the SUT with 30 4-value parameters is 46, while it is 44 when the test set is generated by HSST, as showed in the fourth column of table 2. From table 2 we can see the tool HSST, developed by us, can generate smaller test suite than PairTest in many cases.

VI. CONCLUSIONS

Exhaustive testing is impractical and impossible, so it is a key issue to select the minimal test suite for the effective software testing. The reduced test suite with good quality can improve the efficiency and decrease the cost. In this paper, we proposed a new heuristic based on solution space tree model and implemented it as a tool. We also implemented the algorithm of AETG and the IPO strategy of PairTest, and let them work together. These algorithms can generate good pair-wise covering array as the approximation of the least pair-wise covering array for various cases. The better approximation still needs the further research on the better algorithm.

Since there always exists a least pair-wise covering array for any SUT and the size satisfies the following inequality: $a_1 \times a_2 \leq \text{size} \leq a_1 \times a_2 \times \ldots \times a_n$, the generation of the least pair-wise covering array is to the search the least test subset from the whole set with size of $a_1 \times a_2 \times \ldots \times a_n$. To some important software with high testing cost, it is very valuable to reduce the test set. Sometimes the SUT even needs to cover the 3-way, 4-way or t-way combinations of the system test parameters. So in the future work, we will research on the general problem of the t-way covering array generation.

REFERENCES

[1] A. S. Hedayat et al. *Orthogonal Arrays: Theory and Applications*. pringer-Verlag, USA,1999.

[2] R. Mandl. "Orthogonal Latin Squares: An Application of Experimental Design to Compiler Testing," In: Communications of the ACM, Oct. 1985, 28(10): 1054-1058.

[3] I. S. Dunietz, W. K. Ehrlich, B. D. Szablak, C. L. Mallows, A. Iannino. "Applying design of experiments to software testing: experience report," Proceedings of the 19th international conference on Software engineering, Boston, Massachusetts, United States, 5(1997), 205-215.

[4] A. M. Salem. "A software testing model: Using Design of Experiments (DOE) and logistic regression," Dissertation for PhD, Florida Institute of Technology. Melbourne, Florida.Dec.2001.

[5] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Harowitz. "Model Based Testing in Practice," Proceedings of ICSE 1999, May 1999, Los Angeles, 285-294.

[6] K. Burr, W. Young. "Combinatorial Test Techniques: Table-Based Automation, Test Generation, and Test Coverage," International Conference on Software Testing, Analysis, and Review. San Diego, CA, October 26 - 28, 1998.

[7] H. Jerry. "Reducing Time to Market with Combinatorial Design Method Testing," INCOSE Colorado 2000 Conference.

[8] D. M. Cohen et al. "The AETG System: An Approach to Testing Based on Combinatorial Design," In: IEEE Trans. on Software Engineering, July 1997, 23(7): 437-444.

[9] D. M. Cohen et al. "The Combinatorial Design Approach to Automatic Test Generation," In: IEEE Software Sep. 1996: 83-87.

[10] D. M. Cohen et al. "New Techniques for Designing Qualitatively Independent Systems," In: J. Combin. Designs. 1998, 6(6): 411-416.

[11] Y. Lei and K. C. Tai. "In_Parameter_Oder: A Test Generation Strategy for Pair-wise Testing," In: Technical Report TR-2001-03. Dept. of

Computer Science, North Carolina State Univ, Raleigh, North Carolina, Mar. 2001.
[12] K. C. Tai and Y. Lei. “A Test Generation Strategy for Pair-wise Testing,” In: IEEE Trans. on Software Engineering, Jan 2002, 28(1): 109-111.
[13] N. Kobayashi, T. Tsuchiya and T. Kikuno. “A New Method for Constructing Pair-wise Covering Designs for Software Testing,” In: Information Processing Letters 81(2)2002: 85-91.
[14] A.W. Williams, R L Probert. “A Practical Strategy for Testing Pair-wise coverage of Network Interfaces,” In: Proc. 7th Internat. Symp. Software Reliability Engineer. Nov, 1997. 246-254.
[15] A.W. Williams. “Determination of Test Configurations for Pair-Wise Interaction Coverage,” In: Proceedings of the 13th International Conference on the Testing of Communicating Systems (TestCom 2000), Ottawa Canada, August 2000, pp. 59-74.
[16] A. W. Williams. “Software component interaction testing: coverage measurement and generation of configurations,” A thesis for PhD, Ottawa-Carleton Institute for Computer Science, School of Information Technology and Engineering, University of Ottawa, Canada. 2002.
[17] P. J. Schroeder. “Black-box test reduction using input-output analysis,” Dissertation for PhD, Department of Computer Science Illinois Institute of Technology, Chicago, IL.USA.
[18] P. J. Schroeder, B. Korel. “Black-box test reduction using input-output analysis,” In: ISSTA’00, Portland, Oregon, 2000:173-177.
[19] D. R. Kuhn and A.M. Gallo. “Software Fault Interactions and Implications for Software Testing,” IEEE Trans. On Software Engineering, June 2004, 30(6): 418-421.
[20] D. R. Kuhn, M. J. Reilly. “An Investigation of the Applicability of Design of Experiments to Software Testing,” 27th NASA/IEEE Software Engineering Workshop, NASA Goddard Space Flight Center, 4-6 Dec 2002.

Towards the Establishment of an Ontology of Software Testing

Ellen Francine Barbosa, Elisa Yumi Nakagawa and José Carlos Maldonado
University of São Paulo – ICMC/USP, São Carlos (SP), Brazil
{*francine, elisa, jcmaldon*}*@icmc.usp.br*

Abstract

A growing interest on the establishment of ontologies has been observed for the most different knowledge domains. This work presents OntoTest *– an ontology of software testing, which has been developed to support acquisition, organization, reuse and sharing of testing knowledge.* OntoTest *has been built with basis on ISO/IEC 12207 standard and intends to explore the different aspects involved in the testing activity. Specifically, we are interested in defining a common well-established vocabulary for testing, which can be useful to develop supporting tools as well as to increase the inter-operability among them.*

Keywords: *Ontologies, Software Testing, Reference Architectures.*

1. Introduction

Ontologies – formal explicit specifications of a shared conceptualization [5, 11] – have been applied to describe a variety of knowledge domains. In the field of software engineering, specific ontologies have been identified. Falbo et al. [2], for instance, developed an ontology of software process to support the acquisition, organization, reuse and sharing of software process knowledge. In another work, Huo et al. [7, 12] investigated the development of an ontology of testing as a support for a multi-agent software environment which tests web-based applications.

Software engineering ontologies can contribute to developmental activities in different perspectives. One of them refers to the establishment of reference architectures to ease the use and integration of tools, processes, and artifacts in SEEs (Software Engineering Environments). A reference architecture can be seen as the knowledge of a specific domain, consolidated by activities and their relations [4]. The proposal for reference architectures for domain-specific systems is not trivial, involving deep knowledge about the domain for which the reference architecture is being built. Ontologies can play an important role in this direction.

In an on-going work we are investigating the establishment of *RefASSET* (Reference Architecture for Software Engineering Tools) – a reference architecture that supports the development of environments/tools to automate the software engineering activities [10]. To implement SEEs based on *RefASSET*, it is necessary to refine *RefASSET* to a specific domain and establish a reference architecture for that domain. We have refined *RefASSET* to the testing domain in order to provide a basis to the development of testing tools and frameworks for testing tools, resulting in *RefTEST* (Reference Architecture for Software Testing Tools).

In this paper we present *OntoTest* – the ontology of software testing used in the architectural specialization of *RefASSET* to the testing domain. Based on ISO/IEC 12207, *OntoTest* intends to explore the different aspects involved in the testing activity – techniques and criteria, human and organizational resources, automated tools. Specifically, we are interested in defining a common well-established vocabulary for software testing, which can be useful to develop supporting tools as well as to increase the inter-operability among them.

The remainder of this paper is organized as follows. Section 2 provides an overview on the notations for building *OntoTest*. Section 3 presents the definition of *OntoTest* and its general structure. The main ontology and its sub-ontologies are discussed. In Section 4 we summarize our contributions and discuss the perspectives for further work.

2. Notations for Building Ontologies

Several methodologies for building ontologies can be found in the literature [2, 3, 5, 6, 11]. Despite their differences, a set of basic activities can be identified: planning, requirements specification, knowledge conceptualization (or ontology capture), formalization, integration, implementation, evaluation, documentation, and maintenance. To develop *OntoTest*, we have focused on the ontology capture and formalization. Ontology capture refers to the identification/organization of relevant concepts for the knowledge domain and their inter-relations. A conceptual model, in conjunction with a dictionary of terms, can be used to describe the domain. Ontology formalization aims to represent the conceptual model into a formal language. Concepts and relations, as well as their definitions, properties and constraints, are expressed by a set of axioms [2, 3].

To capture the testing ontology, we have adopted a graphical language – a UML's profile for expressing ontologies [8]. By adopting UML, we intend to ease the validation of the ontology by human experts. To formalize our ontology, a formal language has been used. We have written formal axioms by using the first order logic, as proposed in [2]. Two groups of axioms are distinguished. Some relationships (e.g., whole-part, sub-type) may present a stronger semantics, having an association theory related to them. Theories are composed by association properties such as atomicity, anti-symmetry and transitivity [8]. These relationships are responsible for generating the *structuring axioms*. Besides generating pre-defined structuring axioms, it is necessary to allow that specific theories associated to the domain can be composed and applied to the relations in the ontology [8]. In this case, *consolidation axioms* are established. Both structuring and consolidation axioms have been derived in the development of *OntoTest*.

3. OntoTest: An Ontology of Software Testing

A significant amount of information on software testing has been accumulated over the last years. Such diversity makes the establishment of a consensual shared understanding on testing a fundamental issued to be addressed. Huo et al. [7, 12] investigated the development of a testing ontology to mediate communication between software agents. The ontology was represented in UML, at a high level of abstraction, and in XML to codify the knowledge of testing for agents' processing of messages.

Next we present our proposal of an ontology of software testing – *OntoTest*. Most of the testing concepts considered in Huo's work [7, 12] are in agreement with our work. Additionally, we have explored aspects from: (1) ISO/IEC 12207 standard; (2) definition and evaluation of testing criteria; (3) theoretical and empirical studies involving testing; and (4) knowledge and experience in testing tools development. Due to the complexity of the testing domain, we have adopted a layered approach [2] to the development of *OntoTest* (Figure 1). In the ontology level, the Main Software Testing Ontology deals with the main concepts and relations associated to testing. In the sub-ontology level, specific concepts from the Main Software Testing Ontology (Testing Process, Testing Phase, Testing Artifact, Testing Step, Testing Procedure, and Testing Resource) are refined and treated in details. This structure makes *OntoTest* flexible to (re)use and to integrate – depending on the application, it can be used as a whole (two levels), or only its main ontology or some of its sub-ontologies can be considered.

3.1. Main Software Testing Ontology

To develop the Main Software Testing Ontology (Figure 2) we established an analogy between software process and software testing process. Similar to software process,

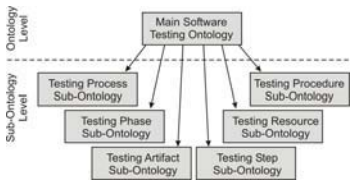


Figure 1. OntoTest Structure

which is a sequence of steps required to develop and maintain software, a testing process (`TestingProcess`) can be seen as a sequence of testing steps (`TestingStep`) required to develop and maintain the testing activity. Also, it has different testing phases (`TestingPhase`). A testing step can consume and/or produce several testing artifacts[1] (`TestingArtifact`) and can use different testing resources (`TestingResource`). Moreover, when defining a testing process, it is important to determine how the testing steps will be performed. To do so, we must establish the testing procedures (`TestingProcedure`), adopted in the accomplishment of the testing steps.

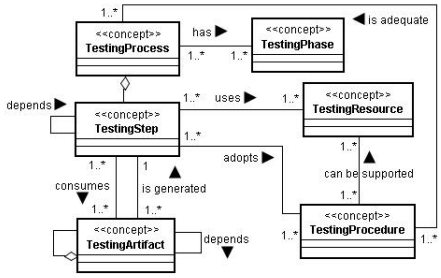


Figure 2. Main Software Testing Ontology

To formally define the main ontology, a set of axioms was established. For the sake of space, just some of them are presented here. Axiom A1 shows the *composition* relation (whole-part) between testing process and testing steps. Such relation was formalized by the predicate $TestingProcessComposition(p, s_1, \ldots, s_n)$, denoting a testing process *p* is composed by testing steps s_1, ..., s_n.

$(\forall p, s_1, \ldots, s_n)(TestingProcessComposition(p, s_1, \ldots, s_n) \rightarrow TestingProcess(p) \wedge TestingStep(s_1) \wedge \ldots \wedge TestingStep(s_n))$ (A1)

The relations *consumes* and *produces* between testing step and testing artifact were formalized by the predicates $ConsumedTestingArtifact(a, s)$, denoting a testing artifact *a* is consumed by a testing step *s*; and $ProducedTestingArtifact(a, s)$, denoting a testing artifact *a* is produced by a testing step *s*. The predicate $TestingArtifact(a, t)$ denotes *a* is a testing artifact and *a* can assume different types (*t*). The symbol $(-)$ indicates

[1] A testing step can also consume some artifacts produced by other activities of the software development process (e.g., a requirement specification document, a quality plan, and so on).

the value for this argument does not matter. Axioms A2 and A3 hold:

$(\forall a, s)(ConsumedTestingArtifact(a, s) \rightarrow TestingArtifact(a, -) \wedge TestingStep(s))$ (A2)
$(\forall a, s)(ProducedTestingArtifact(a, s) \rightarrow TestingArtifact(a, -) \wedge TestingStep(s))$ (A3)

An output artifact produced by a given testing step can also correspond to an input artifact, which will be consumed by other steps. $PreStep(s_1, s_2)$ denotes the testing step s_1 precedes the testing step s_2. Axiom A4 holds:

$(\forall s_1, s_2)(PreStep(s_1, s_2) \rightarrow (\exists a)(ProducedTestingArtifact(a, s_1) \wedge ConsumedTestingArtifact(a, s_2)))$ (A4)

For each basic concept represented in the Main Software Testing Ontology, there may be a number of subconcepts which are refined and treated in the sub-ontologies. Next, we briefly discuss each sub-ontology of *OntoTest*.

- **Testing Process**: Testing processes are defined based on a development technology and on a development paradigm, as well as on a testing procedure. Testing life cycle models are generally used as a reference in the definition of a testing process, establishing macro-steps and the dependency relation between them.

- **Testing Phase**: Testing can be divided into three incremental phases [9]. Unit testing explores each unit separately to ensure that its algorithmic aspects are correctly implemented. Integration testing aims at identifying faults related to the interfaces between units. System testing intends to assure that software and other elements of the system are adequately combined and the functionalities and the expected performance are in agreement with the specification. Besides that, data collected during the tests also play a fundamental role for maintenance [9]. Regression testing (i.e., the selective retesting of a software system that has been modified), intends to ensure that the newly modified code still complies with its specified requirements and the unmodified code has not been affected.

- **Testing Artifact**: Each testing step may involve a number of different artifacts, such as test documents, test diagrams (e.g., control-flow and def-use graphs), test cases, test requirements (e.g., nodes, edges, paths, interactions among definitions and uses of variables, program mutants), drivers and stubs, and artifacts under test (e.g., source or instrumented code).

- **Testing Resource**: Testing resources are required to the accomplishment of a testing step. A testing resource can be a human resource (as a tester or a test manager), a hardware resource, or a software resource (as a testing tool or a supporting system). Hardware and software resources are characterized in terms of a testing environment, which can be used to automate the testing procedures. Testing tools, which provide automated support for performing the tests, are a special kind of software resources. Considering ISO/IEC 12207, we classified them into primary, organizational and supporting tools. Based on our experience in testing tools development, we identified and represented in *OntoTest* a set of functional modules that a primary testing tool should provide: Test Case Management Module, Test Artifact Handling Module, Test Session Module, Test Requirement Module, Test Case Execution Module, Test Analysis and Measurement Module. Similarly, organizational and supporting testing tools were also specified.

- **Testing Procedure**: Testing procedures can be categorized in testing methods, testing guidances and testing techniques. Testing techniques, for instance, have been investigated to perform the tests in a systematic way and on a sound theoretical basis. The idea is to select subsets of the input domain, preserving the probability of uncovering faults [9]. Functional, structural, error-based and state-based are examples of testing techniques. Each primary technique establishes one or more testing criteria, which can be categorized as selection or adequacy criteria. Different testing strategies can be established depending on the testing procedures, testing steps and testing phases considered.

3.2 Testing Step Sub-Ontology

Figure 3 illustrates the Testing Step Sub-Ontology, developed in order to refine the concept of a testing step from the Main Software Testing Ontology. When performing a test, a set of essential steps (`TestingStep`) should be considered [9]: testers should be able to plan tests (`TestPlanning`), construct test cases (`TestCaseDesign`), execute the tests (`TestExecution`), and analyze and evaluate the testing results (`TestAnalysis`). Moreover, a testing step can be seen as a composition of testing activities (`TestingActivity`). According to ISO/IEC 12207, an activity can be classified as primary, organizational or supporting activity, depending on the role it plays in the development process. In this sense, we have identified as primary activities of software testing: test case generation (`TestCaseGeneration`), testing artifacts handling (`TestingArtifactHandling`), test requirements generation (`TestRequirementGeneration`), test case execution (`TestCaseExecution`), and test analysis and measurement (`TestAnalysisMeasurement`). Similarly, organizational and supporting testing activities have also been established.

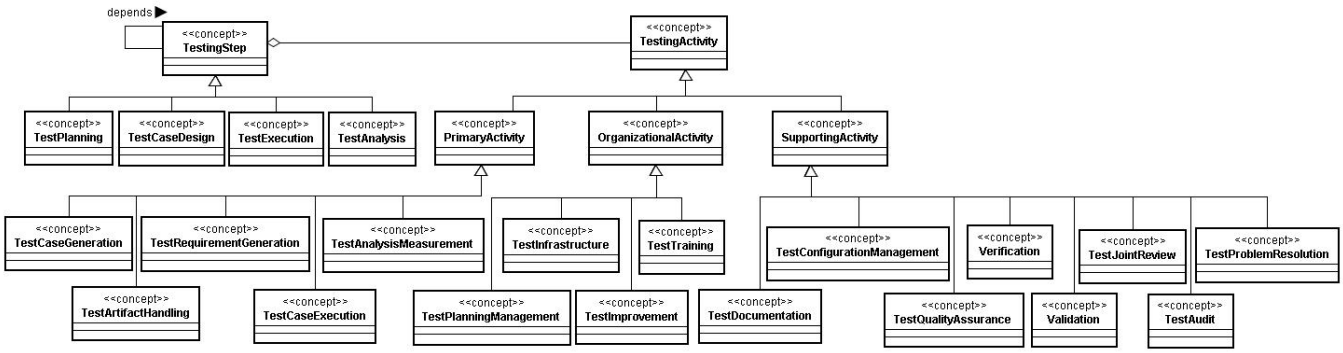


Figure 3. Testing Step Sub-Ontology

Notice that constraints between testing steps and testing activities must be established. For instance, `TestCaseDesign` involves `TestCaseGeneration`. A set of axioms was also defined to capture and write down these kinds of constraints.

4. Conclusions and Further Work

In this paper we discussed the establishment of *OntoTest*, a generic ontology of software testing, developed in order to promote acquisition, organization, reuse and sharing of testing knowledge. *OntoTest* has been built with basis on ISO/IEC 12207 and on our knowledge and experience in the software testing domain. Regarding its development, we focused on the ontology capture and on the ontology formalization activities. Graphical representations for the *OntoTest* components (main ontology and sub-ontologies) were developed and a set of axioms was derived.

OntoTest was applied in the architectural specialization of *RefASSET*, a reference architecture for SEEs, to the testing domain [10]. Such refinement resulted in *RefTEST*, a reference architecture for testing tools, that can be used as a supporting mechanism from which testing tools can be designed and implemented. As further work, we intend to keep investigating *OntoTest* in the context of reference architectures, specially regarding the development of application frameworks for testing.

Another scenario where we have explored the use of *OntoTest* is in the establishment of a learning process of software testing [1]. The availability of such process could promote the improvement of personnel formation and the technology transfer. Besides that, since the body of knowledge on testing keeps evolving, mainly due to the theoretical and empirical studies conducted, *OntoTest* could also help on the establishment of a systematic way to deal with the evolutionary aspects associated with the testing information. In other words, it could make easier both to carry out changes to the testing knowledge as well as to teach this new knowledge to software engineers. In our research line, at the very end, we intend to provide a context for "open learning materials". In this direction, the establishment of learning processes based on ontologies should ease the cooperative work to create, reuse and evolve the materials.

References

[1] E. F. Barbosa. *Uma Contribuição ao Processo de Desenvolvimento e Modelagem de Módulos Educacionais*. PhD thesis, ICMC-USP, São Carlos, SP, March 2004. (in Portuguese).

[2] R. A. Falbo, C. S. Menezes, and A. R. Rocha. A systematic approach for building ontologies. In *VI Ibero-American Conference on AI (IBERAMIA 98)*, Lisboa, Portugal, 1998.

[3] M. Fernández, A. Gómez-Pérez, and N. Juristo. Methontology: From ontological art to ontological engineering. In *Workshop on Knowledge Engineering: Spring Symposium Series (AAAI97)*, pages 33–40, Mellow Park, CA, 1997.

[4] David Garlan. Software architecture: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 91–101, New York, NY, USA, 2000. ACM Press.

[5] A. Gómez-Pérez, O. Corcho, and M. Fernandez-Lopez. *Ontological Engineering: with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer, 1st. edition, 2004.

[6] M. Grüninger and M. S. Fox. Methodology for the design and evaluation of ontologies. In *International Joint Conference on Artificial Inteligence (IJCAI95), Workshop on Basic Ontological Issues in Knowledge Sharing*, 1995.

[7] Q. Huo, H. Zhu, and S. Greenwood. A multi-agent software environment for testing web-based applications. In *27th Annual International Computer Software and Applications Conference (COMPSAC03)*, 2003.

[8] P. G. Mian and R. A. Falbo. Supporting ontologies development with ODEd. *Journal of the Brazilian Computer Society*, 9(2):57–76, November 2003.

[9] G. J. Myers, Corey Sandler, Tom Badgett, and Todd M. Thomas. *The Art of Software Testing*. John Wiley & Sons, 2nd. edition, 2004.

[10] E. Y. Nakagawa. *Uma Contribuição ao Projeto Arquitetural de Ambientes de Engenharia de Software*. PhD thesis, ICMC/USP, São Carlos, SP, may 2006. (in Portuguese).

[11] M. Uschold and M. Grüninger. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2), June 1996.

[12] H. Zhu and Q. Huo. Developing a software testing ontology in UML for a software growth environment of web-based applications. In *Software Evolution with UML and XML*. Idea Group, 2004. Hongji Yang (eds.).

Automating the Implementation of Mobile Applications and Services

Michael Jiang, Anant Athale, Zhihui Yang
Software and System Engineering Research Lab, Motorola
{michael.jiang, anant.athale, zhihui.yang}@motorola.com

Rajarshi Chatterjee and Jay Acharya
Networks Applications and Services, Motorola
{rajarshi.chatterjee, jay.acharya}@motorola.com

Abstract

This paper describes a service creation framework to automate the design of message-based mobile applications and services. In this framework, mobile applications are modeled using design flows containing rules and constraints of application domains. Mobile service components serve as the building blocks for the implementation of domain functions and executable mobile applications are generated from the workflow-based design specifications. Experiments of the service framework for design specification and automatic generation of mobile applications and services will be described.

Keywords: mobile data services, J2EE, message services, and component-based design, domain-specific modeling, automated software design.

1. Introduction

Various types of mobile data applications and services, such as text message services, location-based and presence-based services, multi-media message services, etc., have been proposed, developed, and deployed by wireless service providers and made available to varieties of mobile devices [1] [2]. The development of these applications and services is a challenging task due to the various access technologies, protocols, and legacy systems in the wireless network infrastructures. Mobile applications and services need to function in a heterogeneous networking environment and be integrated well with existing authentication, accounting, billing, and other legacy systems.

Several wireless service platforms and frameworks have been proposed to hide the complexity and heterogeneity of wireless network infrastructures to simplify the creation and deployment of mobile applications and services [3] [4]. These wireless service platforms and frameworks provide a common application programming interface (API) for the development of applications and services independent of wireless communication protocols, network devices, and network resources. By raising the level of design abstraction, such wireless service platforms and frameworks enable rapid service creation and deployment. Authors in [□] describe a programming framework that provides a generic synchronization model for mobile enterprise applications that use heterogeneous backend stores. The framework offers a uniform data access interface that applications can use for data replication and synchronization. A framework for implementing interactive multimedia messaging services is described in [6] and a programming model is introduced to support the development of these applications. An adaptive middleware infrastructure is described in [7] to simplify the development of mobile applications. The authors in [8] describe an enterprise mobile service platform that delivers end-to-end mobile solutions to large enterprises. The architecture of this enterprise mobile service platform allows developers to rapidly add new wireless enterprise applications and services independent of specific access devices and wireless networks. Authors in [9] describe an approach for analysis, design, and simulation of mobile systems. This paper presents the basic structures and operations common to a certain class of mobile computing platforms to support application design and development. Knowledge-based approaches [10] [11] are also adopted for the specification, analysis, and development of software systems.

In this paper, we describe a component-based service creation framework for the creation of message-

based mobile applications and services. The main contribution of this paper is the automatic generation of mobile applications and services from component-based design specifications. By capturing domain knowledge and expertise into the design process, the service creation framework provides application developers with design workflows and service components for the design specifications of mobile applications and services. Executable programs are generated from the design specifications and packaged for deployment on application servers integrated with mobile service platforms. We developed a prototype to demonstrate the effectiveness of this service creation framework.

The rest of the paper is organized as follows. Section 2 describes the architecture and design of the service creation framework. Section 3 describes the generation of application code from design specifications. Experiments of the service creation framework for developing mobile applications and services are described in section 4 and the conclusion of the paper is presented in Section □.

2. A component-based design framework for mobile applications

The service creation framework takes the component-based design approach to simplify the development of mobile applications by hiding the complexity of the wireless networks, communication protocols, and legacy systems. In this section, we first describe high level architecture of mobile applications and services. We proceed to describe the component-based design approach to model domain functions and infrastructure services to support specification of message-based mobile applications and services. Finally, we describe the workflow-based programming model for the design specifications of mobile applications.

2.1. High level architecture for mobile services

Mobile applications and services require the support of network infrastructure, application servers, and mobile clients. Figure 1 shows the high level end-to-end architecture of message-based mobile services. The Message Gateway provides the communication mechanism for message exchanges between Mobile Clients and Mobile App Servers hosting mobile applications and services. Mobile Clients request for message-based services through the Message Gateway by sending SMS (Short Message Service) or MMS (Multimedia Messaging Service) messages to mobile application servers. The function of the Message Dispatcher is to take messages from the message queues and dispatch to the appropriate application servers to be processed by the hosted mobile applications and services.

Mobile applications and services are often required to interact with legacy systems and access to database management systems. Billing and accounting systems are some of the typical legacy systems that require integration. The access to database systems is required by mobile applications and services to support data storage, application configuration, and transactions. In this service creation framework, service components are designed to hide the integration details of legacy systems and provide unified access to various database management systems.

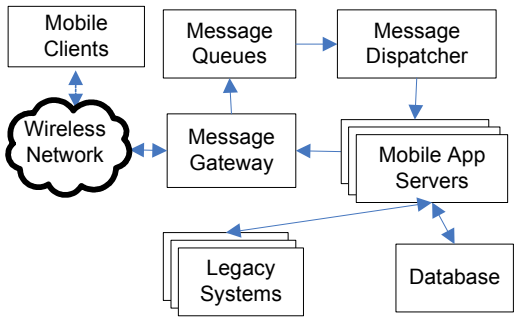


Fig. 1: Mobile service architecture for message-based services

A typical service scenario is illustrated as follows. A mobile client initiates a service request or invokes a mobile application by sending an SMS message to the message gateway over the wireless network. Upon receiving the SMS message, the message gateway places the message into a message queue. The message dispatcher takes the message out of the message queue and forwards the message to an application server based on the type of services specified in the content of the SMS message. The application server interprets the content of the message and performs the necessary functions requested by the mobile client. Information from databases or legacy systems may be accessed to support the message processing. Finally, the application server sends the processing results back to the sending mobile client through the message gateway and the result is interpreted by the mobile client.

2.2. Components for mobile applications and services

Component-based software development paradigm [12] [13] is adopted by our service framework to

provide an abstract layer of domain services for the development of mobile applications and services. Various component technologies are available for such component-based software development. COM (Component Object Model) and DCOM (Distributed COM) define the framework for components to communicate locally or in distributed environment. OMG defines the CORBA Component Model (CCM) to standardize component-based software engineering [14]. Our service framework adopts the J2EE component models including Enterprise JavaBeans (EJB) components, web components, and message-based services [1□].

mobile applications and services as shown in Figure 2. The framework adopts a tiered architecture consisting of application templates, service components, and integrated legacy systems and database management systems. At the top tier, application templates are defined to specify the design workflows for mobile applications and services. Different parts of the workflows are mapped to service components in the second tier and these service components are implemented as J2EE components and web services. Integration with legacy systems and data storage is handled by these J2EE components and web services. To hide the details and complexity of the wireless

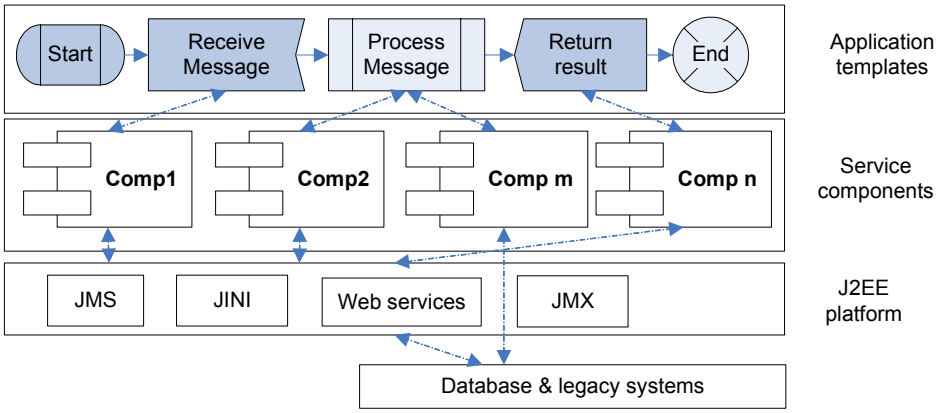


Fig. 2: Framework for mobile service creation.

The service creation framework includes the following categories of software components: message-based components, data access components, J2EE components, and general-purpose components. The message-based components are designed for handling the communication of SMS and MMS messages. Interactions with legacy systems and databases are handled by the data access components. The framework provides general-purpose components for chart generation, web page generation, and the integration of customer extensions to the framework. Text-to-speech and speech-to-text are service components specific to some domain applications.

A common programming interface is defined by the service framework in the form of Java interface definitions for various mobile service components and general-purpose components.

2.3. Modeling of mobile applications and services

The mobile service creation framework combines component-based design and a workflow-based programming model to facilitate the development of

networks and communication protocols, the wireless infrastructure is abstracted as components and services with common access interface in this component-based framework to facilitate the development of mobile applications.

The workflow-based programming model abstracts the design of a domain application as an application template consisting of a set of activities. The application template orchestrates these activities and regulates their interactions and relationships to capture the rules and constraints of a domain application. For an SMS-based query applications, for instance, the activities include receiving of the SMS message, extracting the content of the message, performing the query specified in the message, sending the query results back to the requesting mobile client, and interacting with legacy systems for the purpose of billing and accounting. The workflow also determines the dependencies and relationships of activities. The activity for recording billing and accounting information, for example, may not be performed until the service is completed. The set of activities and their properties and relationships specify the functionality of the domain application.

The function of an activity is specified by a set of properties. The properties of the activity handling message reception include the types of messages and message format denoted in the form of a regular expression. Only those messages matching the specified properties will be received and processed. Some activities may be generic to all applications. For example, the activity for packaging a J2EE application may require the specification of deployment descriptors, such as port number and URL, as its properties.

The implementation of an activity is mapped onto one or more service components in the service creation framework described earlier. The reception of SMS or MMS messages, for instance, is implemented by the corresponding message service components designed to handle message transmission on wireless networks. The specification of an activity contains sufficient information required by service components. In the cases of SMS message reception, the name of the message queue is specified for the storage and retrieval of SMS messages.

The mobile service framework allows the creation of service containers to enable extensions and customization beyond what is provided by application templates. The extension API consists of a MessageProcessor interface that contains framework callback methods to be implemented by the application and a MessageContent class that represents the message data sent to the application for interpretation. The callback methods are invoked by the framework for registered events and messages. The event information is encapsulated and sent as an instance of the MessageContent object for application processing. Service developers can implement custom logic using the APIs defined by the MessageProcessor and MessageContent classes.

3. Generation of mobile applications and services

The service creation framework provides a code generation facility to automate the generation of

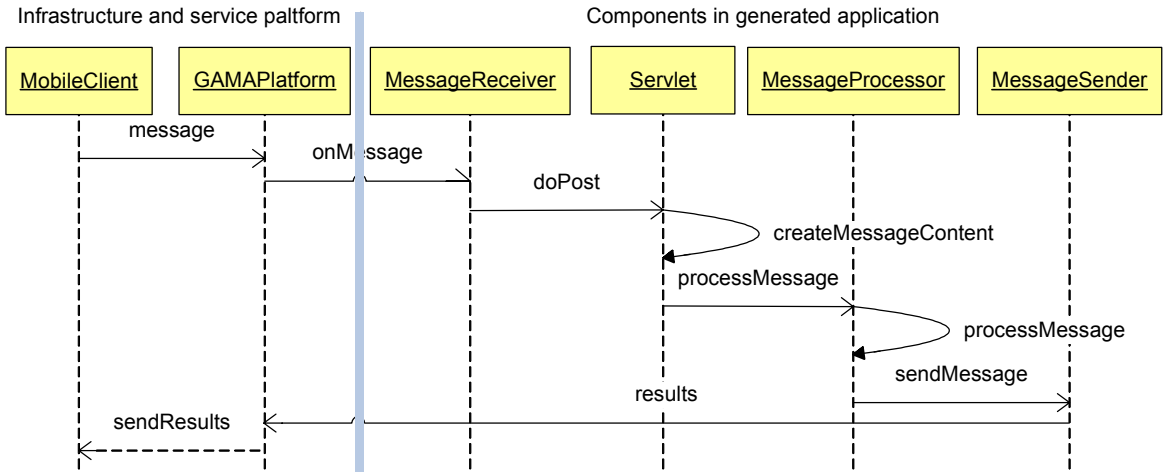


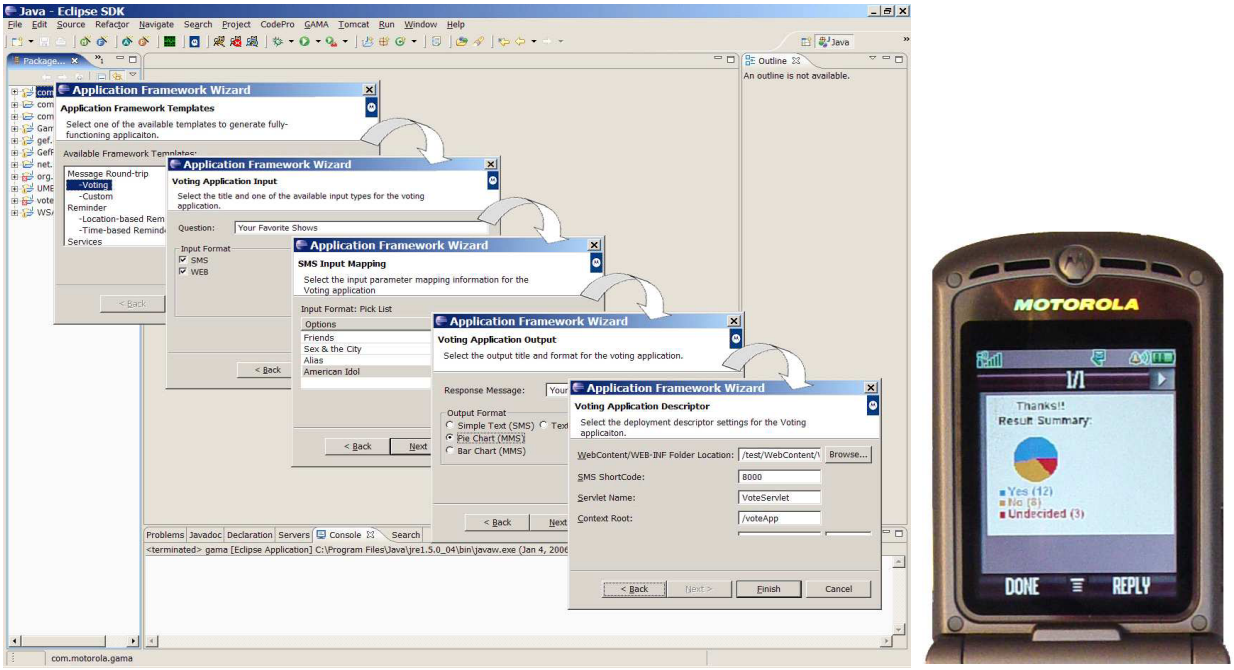
Fig. 3: Components and their interactions in generated application.

The workflow-based programming model and component-based design offer the potential to automate and simplify the development of domain applications. The service creation framework uses the domain knowledge described in application templates to guide application developers for the design specification of domain applications. Developers will focus on specifying domain functions of the design specification and implementing the domain functions using service components in stead of low-level programming. The domain rules also help prevent incorrect composition and interactions of components.

application code from the workflow-based design specifications. Currently the service creation framework generates J2EE applications. For every application template, a J2EE enterprise application project is created and the skeleton code of the J2EE application is generated. The code generation facility interprets the activities and their relationships captured by the design workflows to produce corresponding application code and J2EE deployment descriptors. The properties of receiving messages, for instances, are interpreted by the code generation facility to produce code to invoke the service component designed for receiving messages. The generated application code,

hand-written code, and runtime libraries for service components are packaged as J2EE applications to be deployed on application servers. A browser-based test client is also generated to facilitate the validation of the generated application.

applications by leveraging tools from the Eclipse Web Tools Platform (WTP) project [17] for developing J2EE applications. The packaged mobile applications and services are deployed on the Sun Java System Application Server Platform [18].



(a) Workflow design for a sample application.

(b) Application results

Fig. 4: A sample mobile application.

The components and their relationships in a generated mobile application are shown in Figure 3. These components and classes can also be extended to implement specific domain functionalities of mobile applications and services. The MessageReceiver handles the reception of SMS and MMS messages from the wireless service platform providing common services and application interfaces for interactions with the infrastructure of the wireless networks. Messages are forwarded to MessageProcessor for the processing of messages by the Servlet in response to message events. After the processing is completed, the results are sent back to mobile clients by the MessageSender through the wireless service platform. As discussed in the experiment section, we used the GAMA platform developed internally as the common wireless service platform to conduct our experiments.

The mobile service creation framework is implemented as a set of plug-ins in Eclipse [16], an extensible open source development platform and application frameworks. The applications generated by this framework are packaged and deployed as J2EE

4. Experiments

We developed several application templates specialized in message-based mobile services to demonstrate the effectiveness of this service creation framework. Software components, both common to message-based services and specific to individual services were implemented and made available in this service creation framework. We used a combination of form-driven and dialog-driven workflows to guide the developers to specify the application logic and data processing functions.

In this experiment, our component-based service creation framework leveraged the Global Applications Management Architecture (GAMA) platform developed by Motorola [3]. The GAMA platform shields our service framework from the complexity of wireless network infrastructure and communication protocols. The transportation of SMS and MMS messages and the access of legacy systems are provided

by this wireless application platform. The applications built in this experiment used GAMA application interface APIs to access infrastructure services. We collaborated with network operators to provision the wireless network to route messages to the Motorola GAMA platform for this experiment.

Figure 4 (a) illustrates the development of a sample mobile application using the service creation framework. A message-based voting application is created by instantiating it from an application template available in the service creation framework. The form-driven design workflow will guide the user to specify the properties and attributes of the voting application, presentation of results, access methods, and other information necessary to complete the design specification. The service creation framework provides components appropriate to the selected application type. The code generation facility of the service framework was used to generate the mobile application from the design specification of the voting application and the generated voting application was deployed on the Java System Application Server. Figure 4 (b) shows the results of accessing the generated voting application on a mobile client.

5. Conclusion

This paper describes a component-based and domain-specific service creation framework to automate the development of mobile data applications and services. The component-based software engineering approach facilitates software development by raising the level of design abstraction. With focus on domain-specific engineering, our mobile service creation framework offers effective guidance and constraints to automate the design and development of mobile applications and services. The framework enables application developers to focus on domain problems and solutions in stead of communication protocols and infrastructures of the wireless network. Experience with our development environment showed that the creation of message-based mobile applications and services could be automated.

The current mobile service creation framework has only been used for the specification and generation of relatively small mobile applications and services. Our on-going research will focus on enhancing the service creation framework with more formal and semantically richer modeling languages for the modeling and generation of complex mobile applications and services.

6. References

[1] N. Ravi, C. Borcea, P. Kang, and L. Iftode, “Portable Smart Messages for Ubiquitous Java-enabled Devices”, Proceedings of the First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous), 2004.

[2] B. Wei, et al, “MediaAlert—A Broadcast Video Monitoring and Alerting System for Mobile Users”, Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services, (MobiSys), 200□

[3] Motorola Global Applications Management Architecture (GAMA),http://www.motorola.com/content/0,,20378181,00.html.

[4] HP Mobile Service Delivery Platform (MSDP), http://www.hpintelco.net/pdf/solutions/telecom/mobile/DeliveryMSDP.pdf.

[□] P. Castro, F. Giraud, R. Konuru, A. Purakayastha, and D. Yeh, “A Programming Framework for Mobilizing Enterprise Applications”, Proceedings of the Sixth IEEE Workshop on Mobile Computing Systems and Applications, 2004.

[6] J. Shen, P. Sun, C. Guo, Y. Yin, and S. Song, “Delivering Mobile Enterprise Applications on iMMS Framework”, 6th International Conference on Mobile Data Management (MDM), 200□

[7] R. Cheung, “An Adaptive Middleware Infrastructure for Mobile Computing”, the 14th International World Wide Web Conference, Chiba, Japan, 200□

[8] Y. Chen, H. Huang, R. Jana, et al, “iMobile EE – An Enterprise Mobile Service Platform”, Wireless Networks 9, 283-297, 2003.

[9] P. Guo and R. Heckle, “Modeling and Simulation of Context-Aware Mobile Systems”, Proceedings of the 19th International Conference on Automated Software Engineering (ASE’04), 2004.

[10] J. J.P. Tsai and T. Weigert, “Knowledge-Based Software Development for Real-Time Distributed Systems”, World Scientific Inc., New Jersey, December 1993.

[11] J.J.P. Tsai, T. Weigert, and H. Jang, "A Hybrid Knowledge Representation as a Basis of Requirements Specification and Specification Analysis", IEEE Transactions on Software Engineering, Vol. 18, No. 12, pp. 1076-1100, December 1992.

[12] C. Szyperski, “Component Software: Beyond Object-Oriented Programming”, Addison-Wesley/ACM Press, 2002.

[13] G. Wang, and C. Fung, “Architecture Paradigms and Their Influences and Impacts on Component-Based Software Systems”, Proceedings of the 37th Hawaii International Conference on System Sciences, 2004.

[14] Object Management Group (OMG), “CORBA Component Model”, http://www.omg.org/.

[1□] Sun Microsystems, “Java 2 Platform, Enterprise Edition (J2EE)”, http://java.sun.com/j2ee/.

[16] Eclipse Foundation, http://www.eclipse.org.

[17] Eclipse Foundation, “The Eclipse Web Tools Platform (WTP) project”, http://www.eclipse.org/webtools/.

[18] Sun Microsystems, “Java System Application Server Platform”,http://www.sun.com/software/products/appsrvr_pe/index.xml.

UML Modelling Web Applications via Formal Concept Analysis

Zhuopeng Zhang, Jian Kang and Hongji Yang
Software Technology Research Laboratory
De Montfort University, Leicester, LE1 9BH, England
{zpzhang, jkang, hyang@dmu.ac.uk}

Abstract

Web applications are widely adopted today. Due to the rapid development process and the frequent maintenance, the comprehension of Web applications becomes extremely important. In addition, because Web applications always have a trivial structure, modelling existing Web applications at the design level is fundamental to the maintenance and reengineering. In this paper, we present a flexible approach to model existing Web applications with UML diagrams. Formal concept analysis techniques are adopted to abstract an object-oriented concept model from a Web application, which is the foundation of further UML diagram representations.

1. Introduction

The World Wide Web is progressively evolving toward complex applications. Static Web sites are being gradually replaced by dynamic ones, where information is stored in databases and non trivial computation is performed. Not only is the complexity of Web Applications (WA) kept increasing, but also the development and maintenance process of Web applications turn to be challenging. Web applications usually present disordered architectures, poor or non-existing documentation, and can be analysed, comprehended and modified with a considerable effort.

In order to reduce the effort required to comprehend existing Web applications and to support their maintenance and evolution, Web application reverse engineering [8] methods and tools are being proposed. Web application reverse engineering needs to tackle more issues than traditional reverse engineering, because Web applications normally combine many integrating technologies and multiple languages. In order to recover abstractions, it requires the adoption of dynamic analyses to complement the information gathered with static analysis. In Web application reverse engineering, modelling an existing Web application is essential for its understanding and evolution. The aim of the Web application model is to describe a Web application in terms of composing pages and allowed navigation links. Both dynamic and static pages are to be properly modelled.

We present an approach to modelling WAs in this paper, which is organised as follows. Section 2 and Section 3 presents some related work and background; Section 4 describes the proposed Web application reverse engineering process and the architecture of our Web application modelling system; Section 5 presents a case study. Finally, Section 6 concludes the paper with a summary of our experience to date.

2. Related Work

Unified Modelling Language (UML) has been recently adapted to the Web application domain with the definition of stereotyped elements that can be used to describe the typical components of a Web application. Conallen [4] introduced an extension and a tailorisation of UML for modelling and developing Web applications. Conallen's proposal was focused on forward engineering. Chung and Lee [2] adopted the Conallen UML extensions for Web site reverse engineering, and extracted component diagrams to facilitate Web site maintenance. Adopting the same model, Di Lucca et al. proposed an approach for reverse engineering of Web-based applications, which emphasises both static analysis and dynamic analysis of Web applications [5]. They also provided a tool, called WARE [6], to recover relevant information about some real Web applications and to be modelled by UML diagrams. Furthermore, Di Lucca et al. proposed another approach for reconstructing UML diagrams at business level of the application domain of a Web application [7]. With the emphasis on dynamic analysis, [1] presented an approach and a tool, named WANDA, to recover some extended UML documentation. Although various methods to model WAs with UML have been proposed, the results from the previous research are still not satisfying.

3. Formal Concept Analysis

Formal Concept Analysis (FCA) is a theory of data analysis which identifies conceptual structures among data

sets. In [11], the authors presented a broad overview by describing and classifying academic papers that report the application of FCA in software engineering. FCA has typically been applied in the field of software engineering to support software maintenance and object-oriented class identification tasks. [9], [10] and [12] describe some successfully applications of FCA in software engineering. A strong feature of FCA is its capability of producing graphical visualizations of the inherent structures among data. FCA is more general than graph-based methods as it can capture the same kinds of relations depicted in graphs and presents several additional advantages. These include a finer control over the granularity of the obtained modularisation and an improved discriminatory power.

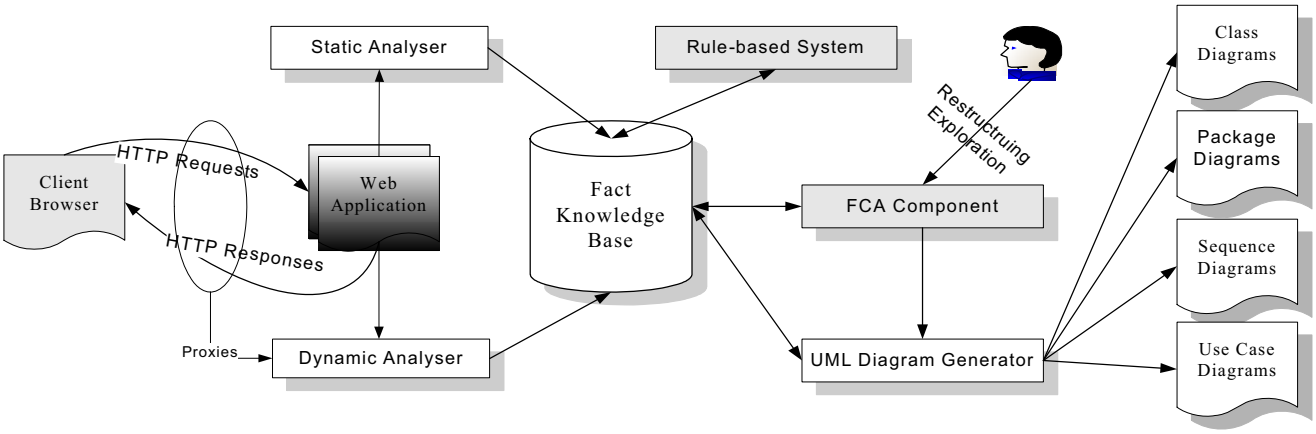


Figure 1. The Web application modelling process

4. Web Application Modelling System

Our Web application modelling system consists of five components as shown in Figure 1. These components are described in the following subsections.

4.1. Static Analyser and Dynamic Analyser

The static analyser is responsible for the first step of the process, namely the analysis of the Web application static structure. The static analyser uses a set of specialised parsers/extractors, such as HTML parser, Java script parser and SQL parser, which analyse the source code and binaries of Web applications. In this static analysis process, HTML, SQL, client-side scripting and server-side scripting are detected and separated. The static relationships between these components are recovered. These recovered static facts are stored in the fact knowledge base.

To fully understand the behaviour of a Web application, it is necessary to detect the dynamic interactions among its components. The dynamic analyser, which is implemented in proxies, monitors the application server to collect the execution traces and invocation information. From the input and output point of view, the HTTP requests, HTTP parameters, Cookies, Database access and invocations to other components are captured. This information is analysed to further understand the active pages. Dynamic analysis provides additional details for the static structure and supports the recovery of the dynamic and behavioural model of the Web applications.

4.2. Fact Knowledge Base

The fact knowledge base is responsible for storing the extracted static and dynamic information. Furthermore, the abstracted UML documentation is asserted in the fact knowledge base as well.

Information graphs [3] are adopted to represent the facts stored in the knowledge base. Information graphs are motivated by Resource Description Framework (RDF), but are simplified. An information graph consists of a collection of triples of the form (object, predicate, subject). The triples contain vertices and each vertex is either a constant or a variable which are identified by strings. Information graphs, rules and queries have formal descriptions. A linear form is a textual representation of information graphs and is used to both enter data graphs and construct query graphs.

After the facts are recovered by the static analysis and dynamic analysis, and stored in the fact knowledge base, the rule-base system is used to extend the knowledge base with new relationships and artefacts. Graph-based queries define an aspect of the code to be explored and are used to generate result sets that are visualised using concept lattices. Human users are able to compare the concept lattices with their background knowledge and formulate hypotheses and further questions. These hypotheses or questions may then be investigated either by generating new lattices, perhaps displaying new aspects of the structure of a Web application, or by navigating back the source artefacts within the Web application or its documentation.

4.3 Applying Formal Concept Analysis

The formal concept analysis component includes a context builder, a concept analyser, a lattice partitioner and a visualiser. The context builder creates a context of objects and attributes relevant to the hot spot to be specified. Then a concept lattice formed from the context together with all possible concept partitions. Finally, the lattice partitioner performs a selection of set of concepts that covers the set of relevant objects. The partition selection process is a semi-automatic interactive process, which needs the supervision of human experts. The visualiser is provided to support the semi-automatic formal concept analysis process. ToscanaJ and Elba are integrated as the visualiser currently.

In the Web application modelling process, FCA is mainly applied to abstract the object-oriented concept model by analysing facts in the fact knowledge base. In order to create the object-oriented concept model, the facts recovered from Web applications need to be mapped into object-oriented entities. The objects in the formal context are the facts in the knowledge base. The attributes in the formal context are defined according to the same user input/output operations, or database access operations, or background scenarios. The formal concept analysis process is performed by query graphs and data graphs. In the abstracted object-oriented concept model, each class may have a couple of Web pages. Parameters of sub-components of each class (e.g., forms, script blocks, database component, etc.) are considered as class attributes, while links are mapped into relations. Elementary parameters may be modelled as attributes, while other, such as database connections, may require the introduction of new classes.

4.4 UML Diagram Generator

The abstraction component works by querying the fact knowledge base and obtaining results from the FCA component. It summarises the results of the queries creating diagrammatic representation. Queries are mostly conceived to detect the building blocks of the WA and their temporal sequence of interaction. Moreover, the interaction frequency and the information exchanged between the entities are exploited to significantly enhance the UML representation providing additional, useful insights.

The UML diagram generator produces class diagrams and package diagrams depicting the Web application according to the object-oriented concept model. Using the information recovered in the previous steps of the reverse engineering process, a use case model can be reconstructed for describing the external behaviours offered by the application to the end users. In addition, sequence diagrams depicting the interactions among the Web application components and the user too, may be produced for each use case. In this step, FCA techniques can be applied to execute queries on the fact knowledge base associated with each Web application, to obtain the required information.

.....	
"notice.htm" is-a HTML-Fact.	_page in-derives _C,
"stulogin.aspx" calls "notice.htm".	_C contains "db.student",
"stulogin.aspx" is-a Ser-Script.	_page name _p,
clock.class is-a applet.	_C kind not–inner,
Clock in "stulogin.aspx".	_p scope {invoked}.
.....	
(a) An information graph	(b) A query graph

Figure 2. The information graph and query graph

5. An Example

In order to evaluate the effectiveness of the proposed Web application modelling approach and our prototype system, several controlled experiments were carried out involving some real-world Web applications. In this section, the results obtained in a case study are presented and discussed. The Web application used in this experiment implements a faculty Web site in a university that supports the academic activities. This WA includes 315 files, distributed in 23 directories, with an overall size of about 60 Mb. The static pages are implemented by 106 HTML files with htm and html extension contained in 13 directories, while 109 files with the aspx extension, contained in 8 directories, implement 109 server pages. The remaining files include data or other objects, like images, logos, pdf files, etc., to be displayed in the pages or downloaded by the users.

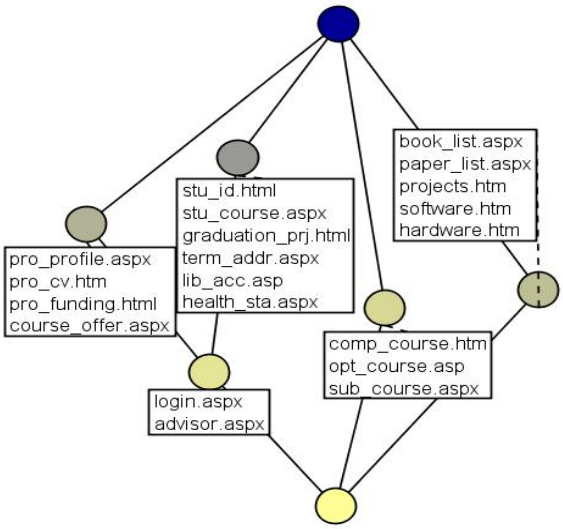


Figure 3. The concept lattice used to abstract class hierarchy

The application was not described by any documentation and just its source code was available. The

maintainers involved in the experiment, with no previous knowledge of it, were asked for using the tool for recovering the UML diagrams of the application. The static analysis of the source files was automatically performed by the tool. The information graphs were generated, and stored in the fact knowledge base. Figure 2a shows a partial information graph. Then dynamic analysis was carried out with the assistance of the maintainers. Some typical user operations were performed to facilitate the dynamic analysis process. After the dynamic analysis, more facts were asserted in the knowledge base.

Based on the information graphs stored in the knowledge base, formal concept analysis is performed by query graphs. An example of a query graph is shown in Figure 2b. Furthermore, Figure 3 shows a concept lattice, which is used to abstract the class hierarchy.

In order to obtain the use cases of the application, a candidate use case was defined and associated with each sub-graph from the class diagram. The scenarios describing the use cases had to be reconstructed too. Each scenario was described by a sequence diagram including the objects corresponding to the items in the sub-graph. An obtained use case diagram is shown in figure 4. The UML diagrams obtained were stored in the fact knowledge base to support the subsequent maintenance and evolution of the Web application.

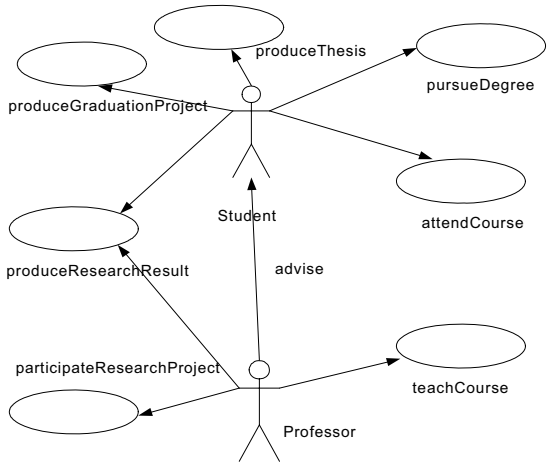


Figure 4. The generated use case diagram of the Web application

6. Conclusions

This paper presented a flexible approach to modelling existing Web applications with UML diagrams. In the approach, the information graphs and related rules are proved to be efficiently used to mine the implicit relationships among program facts via formal concept analysis. During the application of FCA, we found that concept analysis really provides a family of solution algorithms, rather than offering one fixed technique, and different attributes can be chosen for different conditions. In addition, the theory of formal concept analysis allows two or more aspects of the software structure to be combined coherently in nested diagrams. Since each concept lattice is generated from a query graph, a natural refinement ordering allows general views to be elaborated and made more specific. Thus the user is able to progress from a general view to a more specific view, or vice versa. Due to the application of FCA, our approach provides more flexibility and capability. Our experience with the prototype Web application modelling system to date is by no means mature, but the results proved the approach is powerful and promising.

References

[1] G. Antoniol, M. Di Penta and M. Zazzara, "Understanding Web Applications through Dynamic Analysis," In *Proceedings of the 12th IEEE International Workshop on Program Comprehension* (IWPC'04), IEEE CS Press, 2004, pp. 120-129.

[2] S. Chung and Y. S. Lee, "Reverse Software Engineering with UML for Web Site Maintenance," In *Proceedings of the 1st International Conference on Web Information Systems Engineering,* IEEE CS Press, 2000, vol. 2, pp. 157-161.

[3] R. Cole and T. Tilley, "Conceptual Analysis of Software Structure," In *Proceedings of the Fifteenth International Conference on Software Engineering & Knowledge Engineering* (SEKE'03), Knowledge Systems Institute, 2003.

[4] J. Conallen, *Building Web Applications with UML*, Addison-Wesley, 2002, p. 496.

[5] G. A. Di Lucca, M. Di Penta, G. Antoniol and G. Casazza, "An Approach for Reverse Engineering of Web-Based Applications," In *Proceedings of the Eighth Working Conference on Reverse Engineering* (WCRE'01), IEEE CS Press, 2001, pp. 231-240.

[6] G. A. Di Lucca, A. Fasolino, F. Pace, P. Tramontana and U. De Carlini, "WARE: A Tool for the Reverse Engineering of Web Applications," In *Proceedings of the 6th European Conference on Software Maintenance and Reengineering* (CSMR'02), IEEE CS Press, 2002, pp. 241-250.

[7] G. A. Di Lucca, A. Fasolino, P. Tramontana and U. De Carlini, "Abstracting Business Level UML Diagrams from Web Applications," In *Proceedings of 5th IEEE International Workshop on Web Site Evolution* (WSE'03), IEEE CS Press, 2003, pp. 12-19.

[8] H. Kienle and H. Müller, "Leveraging Program Analysis for Web Site Reverse Engineering," In *Proceedings of the 3rd International Workshop on Web Site Evolution* (WSE'01), IEEE CS Press, 2001, pp. 117-125.

[9] M. Siff and T. Reps, "Identifying Modules via Concept Analysis," *IEEE Transactions on Software Engineering*, vol. 25, no. 6, 1999, pp. 749-768.

[10] G. Snelting, "Software Reengineering Based on Concept Lattices," In *Proceedings of the 4th European Conference on Software Maintenance and Reengineering* (CSMR'00), IEEE CS Press, 2000, pp. 3-10.

[11] T. Tilley, R. Cole, P. Becker and P. Eklund, "A Survey of Formal Concept Analysis Support for Software Engineering Activities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 11, no. 1, 1999, pp. 187-201.

[12] P. Tonella, "Concept Analysis for Module Restructuring," *IEEE Transactions on Software Engineering*, vol. 27, no. 4, 2001, pp. 351-363.

Debugging Failures in Web Services Coordination

Wolfgang Mayer and Markus Stumptner
Advanced Computing Research Centre
University of South Australia
[mayer,mst]@cs.unisa.edu.au

Abstract

The rise of Web Services over the past years offers a new development paradigm for distributed applications: high level communication using exchange of structured XML data, using communication protocols orchestrated by workflow languages with complex control constructs. We study the use of model-based techniques that have been used for fault analysis in imperative (Java) and concurrent (VHDL) languages in a Web Service environment, with the goal of diagnosing Web service interactions specified in BPEL4WS, using an Abstract Interpretation approach.

1. Introduction

Web services are currently gaining ground as a new paradigm for distributed applications [1], using Web protocols and XML-based data formats to replace the traditional middleware layer for communication between self-contained externally invokable applications, called services. The XML encoding and the standardised interface definitions such as provided by the Web Service Definition Language (WSDL) [5] facilitates interoperability, making Web services a well suited basis for EAI and cross-company application integration. This has led to the development of service-oriented architectures, where complex applications (business processes) are assumed to be composed from interacting (Web) services.

The development of such service constellations requires specifying and programming the actual interaction patterns (referred to as choreography) between the different services, and dedicated languages have been developed for this purpose, with BPEL4WS [8] and OWL-S [9] currently the foremost representatives. The fact that these languages provide high level process description constructs while their XML-based code and data structures make them amenable to meta-data descriptions such as the various Semantic Web service proposals, invites speculation about automated composition, and repair. As depicted in [2], the goal of self-healing services requires the diagnosis of individual services (which may be written in arbitrary languages) and of diagnosing their choreography. Building on earlier work in diagnosing different software paradigms, in this paper we consider the task of diagnosing BPEL choreographies.

2. A Crash Course in BPEL4WS

The definition of BPEL4WS [8] (shortened to BPEL from here) is based on the assumption that to realise the full potential of Web Services as an integration platform, applications and business processes will need a standard process integration model to interact in a principled fashion, and that model will need to support business process executions: potentially long sequences of message exchanges within stateful interactions run by two or more parties. Thus, need was perceived for a language to specify the patterns of message exchanges and the description of process states (since Web Services, as defined by WSDL, are essentially stateless).

BPEL has been defined to be used in two ways. Either it is used as a specification language for business protocols, which describes patterns of possible message exchanges while abstracting away from specific process details. In this case, definitions are nondeterministic (for example, the specific choice between multiple offered selections could not be specified). They could also be understood as constraints on an actual execution and could be used as a machine readable specification that may be use as additional information in debugging.

The second way, which we will focus on here, is to use BPEL as an executable language that describes the actual computations and message contents to be passed on at the coordination level.

The business processes that BPEL defines are supposed to coordinate Web Services that communicate according to the specifications of WSDL [5]. WSDL services define named *portTypes* (corresponding to interfaces in normal programming languages) that specify individual *operations*. An operation is a predefined exchange of messages, which can be *one-way* (receiving),

request-response, *solicit-response* (two-way, depending on which side starts the exchange) and *notification* (just sending). A service and message name together with a specification of the actual protocol used for sending the message (e.g., SOAP) is called an *endpoint*.

BPEL builds on this structure by establishing *partner links* that send messages between service endpoints. A process also possesses *variables* that can be used to store state data and process history resulting from message exchanges. (WSDL parameters and therefore BPEL variable values are XML structures; the expression language specified to access these structures or parts of them is XPath.)

Example 1 *A standard example frequently mentioned in literature is the "Loan Approval" process (Figure 1). The process involves four processes, each represented as independent Web Service: the client (Cl), the Risk Assessor (RA), the Loan Approver (LA), and the Financial Institution (FI).*

Initially, the client Cl requests a loan of amount monetary units from FI. If the amount is above 10000, the application must be studied in detail and is forwarded to LA for this purpose. Otherwise, if the amount is low, a risk assessment is obtained through the RA service. If the risk is low, approval is automatic. High-risk cases are processed the same way as applications involving large amounts. Once the decision about a request has been made, the Cl is notified.

Provided the application was approved, the client can then withdraw money from the account. In case the amount withdrawn is less than the approved amount, the client is notified, the credit balance is updated and the process ends. Otherwise, an exception is thrown, which subsequently triggers a reply to the Cl. In addition to that, the risk assessment needs to be discarded as the client is no longer trustworthy.

In the rest of this paper, we are interested in the process description dealing with FI and consider the other processes as opaque processes.

Occurrences of individual actions in a BPEL process are referred to as *activities* The basic message passing activities are `invoke`, `receive`, and `reply`, of which the first refers to initiating a one-way or request-response message exchange. Variables are assigned values using the `assign` activity. Basic activities can be grouped by various control constructs (with the usual semantics): `while`, `sequence`, and `switch`.

The two most important constructs that express concurrency and nondeterminism are `flow` and `pick`. The first, unlike a `sequence`, allows to express explicit synchronisation arcs between the activities included in the flow. (These activities can be basic or themselves be nested.) The `pick` activity waits for a set of incoming messages; once one has been received it is chosen for processing and the others are ignored.

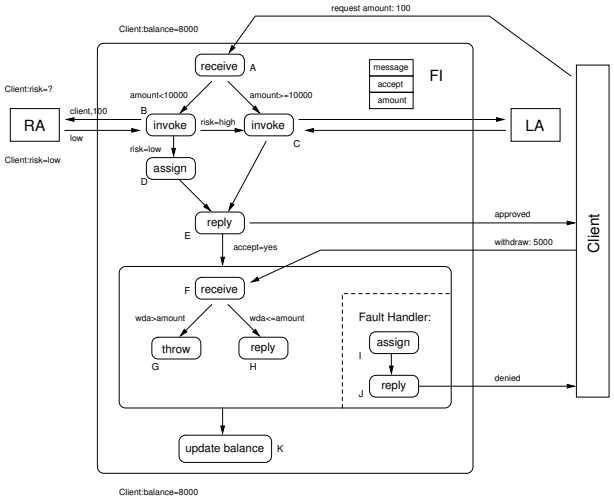


Figure 1. Loan Application Process

BPEL also permits the specification of temporal events, e.g., for defining wait periods and timeouts.

2.1. Fault handling

The core of BPEL is built around the notion of execution *scopes*. A scope contains a process description, together with a fault handler and a compensation handler. In the simplest form, a scope contains a single activity. Dependencies between scopes are modelled through *links*, which define a partial execution order. The execution of a scope can be interrupted either internally, through an uncaught exception in one of the nested scopes, or externally, when a nested scope is terminated due to an exception in an enclosing scope. When a scope *S* is terminated, all scopes enclosed within *S* are terminated immediately, possibly executing fault and compensation handlers. Fault handlers are also represented as activities in a scope, which become active as soon as an activity in the scope raises an exception. Compensation handlers are embedded within the enclosing scope's fault handler.

Since BPEL processes represent long-running business activities, classic atomic transaction models were considered inapplicable. Instead, BPEL relies on explicit reporting of faults, either through a set of system exceptions or by using the `throw` activity, and their handling by dedicated *fault handling* activities. The set of activities that are terminated by throwing a fault is defined by explicitly grouping activities in user-defined *scopes*. A scope is simply a group of activities that are considered grouped for fault handling purposes. Scopes can be nested, and fault handlers can throw faults in enclosing scopes. (Conversely, a fault in a given scope A results in the termination of all activities in A and all scopes nested in A.)

A key property for the fault handlers is that they may contain so-called *compensation handlers*. Transaction effects are not assumed to be automatically rolled back; instead it is the duty of the developers to program compensating actions that restore a correct overall system state, a classic concept from long-running transaction research. An implication of this choice is that compensation handlers are completely application dependent.

2.2. Executable vs Non-executable BPEL

BPEL extensions for executable processes include the ability to explicitly terminate the behaviour of a business process instance. It requires the use of input/output variables in message-related activities (they can be abstracted out in non-executable processes). Also added are fault definitions in case an XPath expression on the right side of an assignment selects no or more than one node; in case a variable or correlation is used before it is initialised; in case multiple `receive` actions are enabled for the same partner, link, portType, operation and correlation sets; and finally for multiple outstanding synchronous requests.

Example 2 (cont'd) *Modelling the process in Figure 1 in BPEL, the decision of whether to automatically assess an application or to can be modelled as a* `flow`*, where the choice which transition to follow is done by the transition conditions amount*$<$10000 *and amount*$\geq$10000*, respectively. The communication with RA and LA is done using synchronised* `invoke` *activities, blocking until the response has arrived. The decision whether to accept or reject the application is stored in a variable* `accept`*. The value of the variable is obtained from the response from LA, or, if the automatic assessment predicted a low-risk case, through an explicit assignment. The reply to the client is modelled using* `reply` *activities to make sure the client can associate the asynchronous reply with the initial request. In case the withdrawn amount is invalid, an error message is constructed in variable* `message` *and is subsequently forwarded to the client.*

Note that the BPEL description contains an error: if the withdrawal fails, the risk assessment is not undone. This is because the fault handler does not include the explicit `compensate` *action to trigger the rollback.*[1]

3. Modelling Web Services for Diagnosis

In this work we are primarily concerned with locating faults in business process models and Web service descriptions. In contrast to previous work [2], our goal is to locate errors *in the description* of a service coordination rather than to identify a faulty service in a groups of interacting processes. While [2] employ local reasoners to monitor the Web service execution and eliminate infeasible diagnoses, we assume a more centralised approach where a single coordination template is the focus of interest. Information obtained from a failing service execution together with descriptions of the interaction protocols of the peer services involved in the execution allows us to derive possible explanations for the misbehaviour in the specification of the local service.

In principle, the same techniques employed to debug computer programs [20, 14] may seem suitable for analysing BPEL descriptions. However, there is a fundamental difference between computer programs and BPEL "programs": while the effects of arbitrary programs can be derived from the program's structure and the semantics of the language constructs, actions in BPEL specification are usually described on a very high level which is not directly amenable for analysis or execution. Consequently, models based solely on the propagation of values between statements are not effective and result in many possible explanations.

We propose to incorporate information about the messages passed between communicating processes to eliminate explanations that imply lost messages or blocked processes. In addition, concrete values obtained from successful and failing executions of the service are exploited to derive contradictions and focus the search to relevant paths through the process.

3.1. Modelling Constraints

Typically, models created for diagnosis purposes are created statically, considering only the structure of the system and the flow between components. This approach has proved to be successful and often allows for optimising the performance of the diagnosis engine by pre-compiling parts of the model description [10, 11]. This approach has also been applied to computer programs [20, 14], but is limited to deterministic execution paths. In particular, presence of loops requires a meta-layer which dynamically modifies the model to accommodate additional iterations. More importantly, the models assume that the data flow between components can be determined statically, which makes them unsuitable for expressing concurrent executions.

Much effort has been invested in modelling different aspects of concurrent systems as automata or transition systems, applying various forms of state space reductions to keep the models small [6]. A prerequisite of many reduction techniques is that all possible transitions between locations in the system are known. Unfortunately, the presence of externally triggered termination of processes, as is present in BPEL, thus renders

[1] While BPEL provides default fault and compensation handlers that would trigger compensation actions in case the exception escapes unprocessed, the explicitly specified fault handler disables this functionality.

many standard techniques unusable for our purposes.

Prior to describing a model that can be used for debugging BPEL descriptions, we first take a look at the constraints the modelling process must adhere to:

- Business processes and Web services are not purely computational services that can be invoked arbitrarily, but also may have some effects on the real world. For example, a commercial service would charge a fee. Thus, the diagnostic process should not rely on the repeated execution of a service, but exploit the information obtained through a single (or a small set of) executions.

- The assumption that a precise description of the (correct) operation of the service would be available is somewhat unrealistic. While description languages such as BPEL and OWL-S gain popularity, generally only a partial description of message sequences and preconditions is provided; a precise specification at a level that could be used for diagnosis is usually not available (yet). Thus, the diagnosis engine should not assume the presence of specifications beyond what is provided through the execution to be debugged and a description of the expected results.

- In contrast to many programming languages and diagnostic models, BPEL processes operate on complex messages containing structured but dynamically generated data, such as XML documents. Consequently, building precise models that can not only check but also predict values becomes more difficult in the general case.

- Concurrent execution and termination of processes are essential features of BPEL and must be supported to a certain degree.

3.2. Dynamic Model

We propose a dynamic modelling approach where the model is not derived statically, but built using the information provided by observations and test cases. The process description to be diagnosed is modified to reflect the current fault assumptions of the diagnosis engine. Then, an abstract execution engine tries to find a path that is (a) feasible given the observations and (b) the modified process description and leads to a state where all observations are satisfied. The behaviour implied by the process description needs to be modelled only to the extent where it is consistent with the constraints given by the test case. A conflict is derived when a consistent model cannot be found.

Definition 1 (Debugging Problem) *A BPEL debugging problem DP is a tuple $\langle BP, TC, COMP \rangle$ where BP denotes the BPEL description to be debugged, TC denotes a set of test case descriptions, and COMP represents the set of diagnosis components in terms of elements of BP.*

BP can be seen as a template describing all possible executions, where the elements that appear in *COMP* may be substituted with "holes" representing arbitrary behaviour, corresponding to fault assumptions made by the diagnosis engine. Typically, elements of *COMP* would be activities, transition conditions, or even entire scopes. For our purpose, "arbitrary behaviour" is defined as assigning unspecified values to *visible variables*[2], continue normally, send or receive messages, or throw an exception. *TC* describes the state of the environment and the inputs provided to *BP*, together with the expected result and any other constraints that every valid execution must satisfy.

3.3. Activation Model

To simplify modelling, we abstract from the concrete syntax of BPEL and represent the process descriptions as nested graphs.

Definition 2 (Scope) *A scope S is a tuple $\langle N, F, V, F, C \rangle$ where N denotes a set of actions and nested scopes directly contained in S, $F \subseteq N \times N$ denotes the partial execution order between elements in N, V denotes a set of variables visible in S and all contained elements, F the fault handler, and C the compensation handler. For all nested scopes N, S is the* parent *of N.*

We model join conditions of activities and transition conditions as separate actions j without side-effects, apart from determining the activation of the component for which j is specified. We assume that this transformation has been applied and omit conditions from actions and links. F specifies a partial execution order in the sense that if for all links $s_i \rightarrow s \in F$, s may be considered for execution only if the status of all s_i has been determined (see below).

For each scope S, we generate a number of artificial variables in the enclosing scope that represent the current status of S: ($+$ denotes "yes", $-$ denotes "no").

- S_{active}=$+$ if the scope is ready for execution, $-$ otherwise.

- S_{done}=$+$ if S has finished executing, $-$ otherwise.

- S_{abort}=$+$ if the scope containing S forces S to abort (due to a failure in a sibling or parent of S), $-$ otherwise.

- S_{exc}=$+$ if S has thrown an exception, $-$ otherwise.

[2] Each variable is visible in the scope where it is defined and in all nested scopes.

S_{active} and S_{abort} must be defined before S is executed, while S_{done} and S_{exc} are defined only after S has completed execution. The reason for having two separate variables S_{active} and S_{abort} is that the execution engine must be able to distinguish the case where S was interrupted while executing from the case where S was never executed because of earlier termination of the enclosing scope. Simply setting $S_{active}{=}-$ would not work because that would lead to a spurious contradiction where S had already been active.

A snapshot E of the current state of execution of BP at any time can be obtained by capturing the values of all variables in all active scopes. The infeasible environment is denoted by $\bot$. Whenever a scope S is scheduled for execution, the variables defined in S are instantiated in E (with initial value $\square$, "uninitialised") and removed again after S has completed. State variables S_* are handled specially in that $S_{active}{=}S'_{active}$ where S' is the parent scope of S in case S does not have predecessors in S. The activation variables associated with each scope identify the current progress of execution. This corresponds roughly to a variable environment found in traditional programming languages. The difference is that here the program counter found in sequential execution is encoded in the S_* variables.

As mentioned previously, a conflict has been derived if no feasible execution satisfying all constraints in $BP \cup TC$ can be found:

Definition 3 (Conflict) *A set $c_1,\ldots,c_k \subseteq COMP$ is a conflict set for DP if*

$$AB_{c_1} \circ \cdots \circ AB_{c_k}(BP) \models \bot,$$

where $\bot$ denotes the infeasible execution and the AB_{c_i} denote mutation functions which modify BP to reflect the abnormality of the source expressions in BP corresponding to c_i.

In contrast to conventional execution, the presence of fault assumptions precludes us from assuming that every execution is deterministic. Instead, the execution engine must be able to follow multiple paths even if every concrete execution of BP was deterministic. For example, it may be necessary to follow multiple paths in a `flow` construct even in case the a transition expressions are complementary, simply because with some variable values unknown, the expressions do not evaluate to a unique value.

3.4. Data Model

We apply Abstract Interpretation [7] to compute approximations if precise values cannot be derived, substituting each language element with an approximation thereof, which operates over a (finite) abstract domain AD. A simple abstraction which either predicts a concrete value or does not predict any value for a BPEL variable is sufficient. For the status variables S_*, a power set is used.

The behaviour of a simple scope S containing only a normal action A reflects the effects as defined in BP and the BPEL language specification, operating on the abstract domain AD. For example, the expression $amount < 10000$ evaluates to $true$ or $false$ in a snapshot E if $E(amount)$ is a concrete value. Otherwise, the result is also undefined.

The descriptions of the BPEL activities are not detailed enough to predict new values given the input values. Instead, we rely on the values derived through the actual execution TE of the test case. However, those values are only guaranteed to be valid in executions where the involved variables V_i and peer processes P_i exhibit the same state as in TE. Otherwise, no values can be predicted. The values of V_i can be directly compared with the value in E. For every peer P_i, it is necessary to track the messages sent and received to and from P_i to ensure the results are the same as the ones obtained in TE. This can be done by introducing an additional variable MS_i for each P_i which acts as an index into the message sequence MS_i^{TE} that was obtained from TE. The value of MS_i is updated whenever a send or receive action involving P_i is executed. MS_i is incremented if the sent or received message matches the next one in MS_i^{TE}, or undefined otherwise.

While this value predictor is quite weak, it can still be effective for limiting the search space of feasible executions in case the fault assumptions are near the end of the execution or if the messages passed between BP and P_i are independent from messages to another P_k.

3.5. Complex Behaviour

The forward behaviour of an individual action A is formalised as transfer functions, taking a process snapshot E as input and computing a set of new process snapshots $\mathscr{E}$ as result, which reflect possible effects of A in E. Typically, $\mathscr{E}$ contains a single snapshot, except for actions which may trigger exceptions. In this case, $\mathscr{E}$ contains two snapshots; one reflecting the normal path, while the other snapshot corresponds to BP following the exception path. In case A is assumed abnormal, a generic snapshot containing unspecified values for the set $\mathscr{V}_A$ of BPEL variables accessible at A is returned, as well as an exception snapshot.

Consider the `assign X:=Y+1` activity (abbreviated "A"):

$$[\![A]\!](E) = \begin{cases} \{E[X \leftarrow E(Y)+1, A_{done} \leftarrow +]\} \\ \quad \text{if } E \neq \bot \wedge + \in A_{active} \wedge + \notin A_{abort} \\ \quad\quad \wedge + \notin A_{done} \wedge \neg AB(A) \\ \{E[\mathscr{V}_A \leftarrow \top], E[S_{exc} \leftarrow \top, S_{abort} \leftarrow +]\} \\ \quad \text{if } E \neq \bot \wedge AB(A) \\ \{E\} \qquad \text{otherwise} \end{cases}$$

where $E[X \leftarrow E(Y)+1]$ denotes the substitution of a new value for X in E and $\mathcal{V}_A$ denotes the set of all variables accessible at A. $\top$ denotes the unknown value. The first clause specifies the normal behaviour,the second clause specifies the abnormal behaviour. The last clause catches all cases where either A is not ready for execution, or E is infeasible.

Each scope S in BP must respect flow constraints between its control variables S_*: S can complete normally iff S was active and S was not aborted (internally or from the outside).

$$S_{active}=- \vee S_{abort}=+ \vee S_{exc}\neq- \Leftrightarrow S_{done}=-$$

$$S_{active}=+ \wedge S_{abort}=- \wedge S_{exc}=- \Leftrightarrow S_{done}=+$$

An activity may be activated if the status of all preceding activities in the partial execution order F has been determined and at least one of the activities has completed normally:

$$S_{active}=\begin{cases} S'_{active} & \text{if } \nexists_{\langle S'',S\rangle\in F} \text{ and } S' \text{ is the parent of } S \\ + & \text{if } \forall_{\langle S'',S\rangle\in F} S''_{done}\neq\square \wedge \exists_{\langle S'',S\rangle\in F} S''_{done}=+ \\ - & \text{if } \forall_{\langle S'',S\rangle\in F} S''_{done}=- \wedge \exists_{\langle S'',S\rangle\in F} \\ \square & \text{otherwise} \end{cases}$$

In practice, the values of some of the S_* variables may not be known precisely and over-approximation must be performed.

To compute the effects of a scope S consisting of multiple activities, a worklist algorithm is applied. Starting with the initial process snapshot E, all nested scopes $S' \in S$ are chosen such that $+ \in E(S_{active})$ and new snapshots $\mathscr{E}=\bigcup_{S'}[\![S']\!](E)$ are computed for all S'. This is repeated until a fixpoint is obtained.

To account for external terminate events, all snapshots after each $[\![S']\!]$ ($S' \in S$) is applied must be combined together to simulate termination of the scope at any point in the execution. This provides a safe but coarse over-approximation of the true behaviour and is refined later.

The initial forward analysis determines if each scope may be terminated either internally or externally. The model is refined subsequently to eliminate spurious paths. If an internal terminate event T, such as a `throw` activity is encountered, all $S'_{abort}=+$ where $S'_{done}\neq+$ for all scopes S' that do not strictly precede T in the partial execution order F.

Once the scope S has been analysed, S_{done} is set to $+$ normal completion snapshot, and to $-$ in the exception case. The snapshots normal completion and exception exits are then propagated to the parent of S for further processing. In case the propagated snapshot has S_{exc} set, the fault handler becomes active.

3.6. Observations

To incorporate observations about the state of processes in BP or TC, the values of observed values in snapshots corresponding to the observation location are intersected with the observed values, potentially leading to a conflict. Execution paths not satisfying these values are eliminated from the model by applying a similar procedure as described in section 3.5 in *backward direction*, intersecting the snapshots obtained thorough backward reasoning with the values derived from forward reasoning [3]. In case $\perp$ is derived, the entire path is eliminated from the model and the analysis resumes with a different branch, until no more branches can be eliminated. A conflict has been derived if there is no feasible execution from the start node in the model to the final state specified in TC. The forward and backward analysis are repeated until a fixed-point (or a specified limit of iterations) has been reached.

After the backward phase, the subsequent forward analysis may transform the model to obtain a more precise abstraction.If it is derived that no external termination event is received, combining the execution snapshots after every activity is not necessary. If the snapshot before a cycle is inconsistent with the snapshot after the cycle, the cycle must execute at least once to obtain a consistent path. This unrolling can be done to adaptively refine a model. Values propagated from observations may contradict certain concurrent execution schedules for interfering components. The corresponding join operations of snapshots can then be eliminated.

3.7. Communication Requirements

So far, only the values monitored during the execution of TC and the observations specified in TC have been considered for diagnosis.To eliminate spurious candidates, we make use of interactions recorded through the initial execution of TC and/or specified in TC. For our purposes, we assume that the messages passed between BP and a peer process P_i are known and their sequence MS_i^{TE} is *fixed and correct*. However, we allow for *additional* messages that may be allowed by the P_i's process description. For example, we may need to consider triggering compensation actions for service RA in Figure 1. While this assumption may seem strong for general processes, in the context of a single BP run it seems to be reasonable if it is assumed that the peer processes would reply with an error message in case their protocol is not followed properly. We assume that a newly developed process description can rely on the other descriptions to be correct.

We also introduce separate variables $seen_i$ to keep track of the progress of sent and received messages for each process P_i. The progress can be represented as an index (or as interval) in the message sequence for pre-

specified sequences in TC. For processes where message sequences are not specified, we assume that for every messages that is sent there must be a matching receiving process, and vice versa. Thus, no messages should be lost nor any process should be caught in a waiting state forever.

Keeping track of the messages allows to exclude paths and concurrent execution schedules that would not satisfy the observed message sequences from our analysis and prune the search space early. This is possible in particular if the interaction protocols include different method types in subsequent steps, as misaligned send or receive operations can be detected early rather than at the end of the analysis.

3.8. Diagnosing an Example

We continue our Loan Assessment example: (1) A loan application for \$100 is requested. (2) The answer is received that the application was approved. (3) The client requests to withdraw \$5000. (4) The request is denied.

Assume further that the test case also expects the risk assessment to be cancelled because of the failed withdrawal attempt. *RA* provides an optional WSDL port that invokes *RA*'s compensation handler. We know that the interfaces of *LA* and *RA* differ, i.e. the WSDL port types are different.

We observe that the loan assessor is not used. The set *COMP* contains all activities $A,\ldots,K$ in *BP*. In the following short-cut the detailed description to suit the illustration.

When the test case is run, we observe that the client receives the expected messages, but the credit assessment is not rolled back ($RA{::}risk \neq \square$). Therefore, we find that the execution does not satisfy the test description and a contradiction has been found.

Starting the diagnosis process, assume the `invoke` activity communicating with *RA* is abnormal. Therefore, after starting the analyser, the initial snapshot E_0 contains the variables of FI: $E_0(message)=\square$, $E_0(accept)=\square$, $E_0(amount)=\square$, $E(A_{active})=+$, and $E(A_{abort})=-$. Also, no messages have been sent or received to and from any peer: $E(seen_i=0)$ for $i \in \{RA, LA, Cl\}$.

A is assumed normal. The initial client request is received ($seen_{Cl}=1$) and, because the message history is the same as for the test execution, the values must be the same as in the faulty execution. Therefore, $E_1=E_0[amount \leftarrow 100]$. The (normal) transition conditions are evaluated and it is determined that $E_1(B_{active})=+$.

As there is no other active activity, B is selected for execution. As B is assumed abnormal, neither the communication events nor the variable assignments need to occur as in the original execution. We obtain *two* possible successors; one for the normal completion/fail stop case:

$$E_2=E_1[\mathscr{X} \leftarrow \top, B_{done} \leftarrow \top, B_{exc} \leftarrow -, seen_i \leftarrow \top, seen_{Cl} \leftarrow [1,4]],$$

and one for the exception case:

$$E_2'=E_1[\mathscr{X} \leftarrow \top, B_{done} \leftarrow -, B_{exc} \leftarrow +, seen_i \leftarrow \top, seen_{Cl} \leftarrow 1],$$

with $\mathscr{X}=\{message, accept, amount\}$ and $i \in \{RA, LA\}$.

The exception case E_2' is aborted because the scope does not contain a fault handler. This case contradicts the expected behaviour and E_2' is eliminated. We continue with E_2. Because the value of $E_2(risk)$ is not known, both paths are feasible, implying $B_{done}=\top$, $C_{active}=\top$ and $D_{active}=\top$.

Because the Loan Assessor is not to be used,activity C must find a different partner. As *RA* and *LA* offer incompatible interfaces, *RA* cannot be a target. Assume that a pair of client ports suits the interface description: $seen_{Cl}=[1,3]$. Continuing the propagation, D is executed and assigns the value "yes" to `accept`.

C and D could possibly be active at the same time, the result must be combined into a single result E_3 and the preceding process repeated with $E_2 \sqcup E_3$ and $E_2' \sqcup E_3$ until a fixpoint is reached. This leads to an environment where the values of *risk*, *accept*, are unknown, the $seen_{Cl}$ variable has value $[1,4]$, $seen_{RA}=[0,2]$, and activity E is activated.

As E is not abnormal, the message matches either the second or the last client message, resulting in $E_4(seen_{Cl})=[2,4]$ and the nested scope being active. As the only active operation at this time is F, we must find a message in the sequence that matches a receive operation. We obtain $seen_{Cl}=3$.

A backward pass is done and the $seen_{Cl}=3$ is propagated to E, which at this point must match the second client message. Propagating this further up to C and D, it is derived that there is no valid message left to choose for C; thus the path is infeasible and is removed. Continuing propagation with D and subsequently B, we derive that $accept=$"*yes*" and that the abnormal B must account for the two messages exchanged with *RA*. As the communication history for the two messages is the same as in the failing test run, the same result must be derived and we obtain that $RA{::}risk=low$. Again, this is a contradiction with the test specification ($RA{::}risk=?$) and, therefore, the abnormal B cannot explain the behaviour of the test run. We apply the BPEL restriction that compensation actions may only be triggered from compensation and fault handlers.

Continuing with the remaining components, we obtain four single fault diagnoses: either I or J may be faulty in that they should issue a message to *RA* invoking the compensation handler.

4. Related work

Our work complements the work in [2] which concentrates on the definition of decentralised interactions between individual diagnosers for each Web service in an cooperative orchestration, but does not examine the individual diagnosis engines or their communication strategies in detail. An earlier approach dealt with generic component-based software systems while considering the individual components as black boxes [12]. In this paper, we focus on the actual orchestration mechanisms, using BPEL4WS as an exemplary choreography language.

Considerable effort has been spent on the use of semantic service descriptions for more or less automated composition of services, e.g., [16, 17], but so far, detailed error handling has only been addressed at the level of fault handler implementation [4], or, at best, verification [13].

5. Conclusion

We considered the task of diagnosing cooperative business processes specified in BPEL4WS, building on and extending earlier work on imperative languages [14, 15], while incorporating BPEL-specific concerns such as concurrency and nondeterminism. We focus on the development of the top level process itself, assuming that pre-existing remote services accessed in the collaboration are less likely to be the source of the fault than their specific usage and the choreography itself (although the individual component services are amenable to being examined in a hierarchic diagnosis process). This work only scratches the surface, with many aspects not yet considered, such as timeouts and timed alarms, deeper analysis of data structures (XML trees that are accessed using XPath – this would be subject to earlier work on debugging in functional languages), and the vast space of possibilities opened up by the incorporation of Semantic Web service specifications in OWL-S or other formalisms. These could be incorporated in the diagnosis process in similar fashion to pre- and postconditions in Java debugging [19]. Ultimately, reconfiguration and planning could be incorporated to effect Web Service repairs.

References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services*. Springer-Verlag, 2004.

[2] L. Ardissono, L. Console, A. Goy, G. Petrone, C. Picardi, M. Segnan, and D. T. Dupre. Cooperative model-based diagnosis of web services. In *Proc. DX'05 Workshop*, 2005.

[3] F. Bourdoncle. Abstract debugging of higher-order imperative languages. In *Proc. SIGPLAN Conf. PLDI*, pages 46–55, 1993.

[4] G. Chafle, S. Chandra, P. Kankar, and V. Mann. Handling faults in decentralized orchestration of composite web services. In *Proc. ICSOC*, pages 410–423. Springer-Verlag, 2005.

[5] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. Technical report, World Wide Web Consortium, 2001.

[6] J. C. Corbett. Using shape analysis to reduce finite-state models of concurrent Java programs. Technical report, Department of Information and Computer Science, University of Hawaii, 1998.

[7] P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction of approximation of fixpoints. In *POPL'77*, pages 238–252, 1977.

[8] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business Process Execution Language for Web Services 1.1. Technical report, IBM and Microsoft, 2003.

[9] D. A. R. P. (DARPA). OWL-based Web Service Ontology (OWL-S) 1.1. Technical report, 2004.

[10] A. Darwiche. Compiling knowledge into decomposable negation normal form. In *Proc.* 16^{th} *IJCAI*, pages 284–289, 1999.

[11] P. Fröhlich and W. Nejdl. A Static Model-Based Engine for Model-Based Reasoning. In *Proc.* 15^{th} *IJCAI*, 1997.

[12] I. Grosclaude. Model-based monitoring of component-based software systems. In *Proc. DX'04 Workshop*, 2004.

[13] R. Kazhamiakin and M. Pistore. A parametric communication model for the verification of BPEL4WS compositions. In *EPEW/WS-FM*, LNCS, pages 318–332. Springer-Verlag, 2005.

[14] W. Mayer and M. Stumptner. Modeling programs with unstructured control flow for debugging. In *Proc.* 15^{th} *Australian Joint Conf. on AI*, pages 107–118. Springer-Verlag, 2002.

[15] W. Mayer and M. Stumptner. Debugging program loops using approximate modeling. In *Proc. ECAI*, 2004.

[16] S. Narayanan and S. A. McIlraith. Simulation, verification and automated composition of web services. In *Proceedings WWW 2002 Conference*, pages 77–88. ACM Press, 2002.

[17] M. Pistore, P. Traverso, P. Bertoli, and A. Marconi. Automated synthesis of composite BPEL4WS web services. In *ICWS*, pages 293–301, 2005.

[18] M. Stumptner. Using design information to identify structural software faults. In *Proc.* 14^{th} *Australian Joint Conf. on AI*, Springer LNAI 2256, pages 473–486, 2001.

[19] M. Stumptner. Web service composition. In *4th International Conference on Information Systems Technology and its Applications (ISTA'05)*, Palmerston North, NZ, May 2005. GI - Gesellschaft fuer Informatik.

[20] M. Stumptner and F. Wotawa. Using Model-Based Reasoning for Locating Faults in VHDL Designs. *Künstliche Intelligenz*, 14(4):62–67, 2000.

OWL-S Ontology Framework Extension for Dynamic Web Service Composition

Jing Dong
Computer Science Department
University of Texas at Dallas
Richardson, TX 75083, USA
jdong@utdallas.edu

Yongtao Sun
American Airline
4333 Amon Carter Blvd
Fort Worth, TX 76155, USA
Yongtao.Sun@aa.com

Sheng Yang
Computer Science Department
University of Texas at Dallas
Richardson, TX 75083, USA
syang@utdallas.edu

Abstract

Composing existing web services for enterprise applications may enable higher level of reuse. However the composition processes are mostly static and lack of support for runtime redesign. In this paper, we describe our approach to the extension of the OWL-S ontology framework for dynamic web service composition. We raise the level of abstraction and propose an abstract service layer so that web services can be composed at the abstract service level instead of the concrete level. Each abstract service is attached with an instance pool including all instances of the abstract service to facilitate fail-over and dynamic compositions.

1. Introduction

Recent advance on web service computing enables building business processes and systems through the discovery and integration of the existing services. Current web services are typically described in terms of atomic and composite web services, using languages like BPEL [5] or OWL-S [2] which provide mechanisms for web service compositions. However, the processes of web service compositions tend to be static in the sense that these processes are normally generated off-line. Any changes to the part of a process may result in the reconfiguration of the whole process. It lacks the support of the capability of fail-over and dynamical redesign. This is especially critical for real-time enterprises since the systems cannot afford to stop, reconfigure, and restart. If some web services of a composition fail or the requirements change, the system needs to be able to change locally and reconfigure on-the-fly.

Among the numerous available web services, there are many that provide similar services. Even with different implementations, most of them present a similar interface to the end user. However, there is currently little effort on abstracting these similar services into high-level common services. Although the OWL-S language provides a way to describe the hierarchical relationship between services, the recommended ontology framework is still limited to one root-level abstract service. Raising the level of abstraction and capturing similar services as a service pool are important, especially for dealing with fail-over and dynamic redesign in real-time enterprises and e-business.

In this paper, we propose an extension to the ontology framework based on OWL-S, which enables defining the composite services at the abstract service level. We provide new constructs to specify such higher level of abstraction. Our approach also includes a service instance pool that allows filtering and plugging in candidate services at runtime. In addition, we offer a planner prototype based on Java Theorem Prover (JTP) [10] that can automatically generate the composition processes on-the-fly.

The rest of this paper is structured as follows. Section 2 gives an architecture overview of our approach. Section 3 defines our extension to the OWL-S Ontology Framework. We describe a prototype planner in Section 4. The last two sections present the related work and conclusions.

2. Architecture Overview

Figure 1 presents an architecture overview of our approach that focuses on the definition of abstract services based on both existing web service instances and the user's expected goals. We define an extension to the OWL-S recommended ontology framework for this purpose. We first define an abstract service hierarchy by grouping the available concrete services into different categories based on their functional characteristics, such as input and output, and specific nonfunctional service parameters. Multiple-level inheritance hierarchy is enabled to represent the hierarchical relationships among these abstract services. A concrete service (existing OWL-S service) can be plugged into the appropriate levels. For example, a concrete flight service AA301 (a flight from American Airline which departs from New York JFK airport and arrives in Chicago ORD airport) is a type of both the abstract American Airline Flight service and abstract JFK-ORD Flight Service which is in turn a type of the abstract Flight service. Since there are thousands of web services already deployed in the Internet and new services available every day which may not be aware of the abstract service hierarchy, we also develop a backend utility program to register a concrete service into a domain service hierarchy and dynamically

update the service ontology when detecting a change. The newly introduced abstract service layer does not require the concrete service to completely conform to the definition of abstract service because our backend utility can automatically detect the equivalent relation between semantic concepts. For example, if one specific abstract Flight service has a "maximumLoad" serviceParameter and a concrete flight service has a "loadLimit" serviceParameter, our utility program considers it a match if the "sameas" relationship has been defined for the "maximumLoad" and "loadLimit" serviceParameters in the ontology. In this paper, we assume that all concrete services of the same abstract service share the same interface information, i.e., they all have the same IOPE parameters, so that we can focus on the abstract service hierarchy. We plan to address different interface mapping in the future.

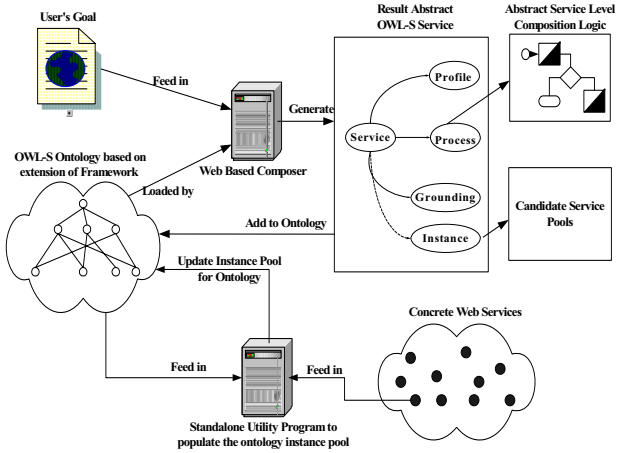


Figure 1 . Architecture Overview

If a composition of services is requested for business or enterprise applications, either an existing abstract service can be matched or a new abstract service is defined with its Profile section containing the input/output and the semantic properties. The Process and new Instance sections can be generated by our composition planner in two steps. First, based on the ontology hierarchy of all available abstract services for existing services and the definition of the user's goals including both the functional requirement and nonfunctional service parameters, we can obtain the composition process (the Process section of the abstract service that satisfies the user's goals) using our composition planner based on JTP [10]. This generated composite service matches all the input/output and flow-related semantic requirements. This can narrow down the candidate scope by filtering out all other unrelated web services. Second, other functional service parameters describing the user's goals are recorded in the Profile section of the abstract service. They are used to identify the actual candidate service pool for composite service and fill in the new Instance section if a direct concrete service candidate is found.

3. Extension to OWL-S Ontology Framework

In this section, we describe our proposed extension to the OWL-S recommended framework. This extension includes the information on the input/output, flow control, semantic property, and candidate instance pool of the abstract service in the ontology hierarchy. This new ontology framework contains the following features:

A new "Instance" section is added to the OWL-S recommended ontology framework. This new section provides the information about the candidate concrete service instances for this abstract service. These service instances can be of a standard web service, an OWL-S or a BPEL service. This Instance section is different from the Grounding section since it does not provide a binding to a physical web service. Instead, it contains a collection of references to the available candidate service instances. If the Instance section of an abstract service is not empty, there is at least one concrete service available for direct invocation. We call all these available concrete services the *instance pool* of the corresponding abstract service. At runtime, the system can pick up a candidate service from the pool and invoke it via its own binding information (like Grounding in OWL-S or WSDL in standard web service). If an abstract service does not have any candidate service instance, it may obtain its instance(s) from the Process section at runtime. For example, a traveler may want a service that allows him to fly from New York to Paris, with a stop at Moscow, called a "NewYork-Moscow-Paris Flight" service, which is not a typical connection flight available from any airline. Thus, the Instance section of the "NewYork-Moscow-Paris Flight" abstract service is empty. The Process section of "NewYork-Moscow-Paris Flight" can specify that the flow of "NewYork-Moscow-Paris Flight" is just a "NewYork-Moscow" service plus a "Moscow-Paris" service. Suppose both "NewYork-Moscow" and "Moscow-Paris" have a number of candidate services in their Instance sections. In this way, a composite concrete service for the abstract service "NewYork-Moscow-Paris Flight " can be obtained by picking one candidate service instance from the instance pool of "NewYork-Moscow" and another from that of "Moscow-Paris". The new Instance section and its relationships to other sections are illustrated in Figure 2. More details about the Instance section is presented in Section 3.2.

New constructs are added to the Process section of the recommended framework. In the Process section of the current OWL-S framework, each OWL-S service can only be an Atomic process, a Simple process, or a Composite process. No matter which type of service, it

can only contain one work flow logic. We define a new "Abstract Process" type, which has an "abstractComposedOf" attribute to specify all possible composition processes it can have. In our approach, we can define a collection of composite processes containing multiple work flow logics in the new abstract service. For example, the work flow logics of both "JFK-DFW-LAX" and "JFK-ORD-LAX" services are included in the new abstract "JFK-LAX" service. Either can be used to fulfill a flight service from NewYork JFK airport to Los Angels LAX airport. More details are presented in Section 3.3.

An abstract service does not contain the Grounding information since it is not mapped to any physical service. Instead, it gets the candidate instance from its Instance section which is like a resource pool.

In the existing OWL-S ontology framework, each service in the hierarchy is still a concrete service. With the above two framework extensions, we can define an abstract service layer, so that future service compositions can be made at the abstract level.

From software architecture and design perspective, our approach defines a service architecture that has more complex structure than the simple "subClassOf" relationship in the current OWL-S framework. With the support of abstract services, it is possible to define more complex service structures based on, e.g., inheritance, information sharing, and polymorphism. Therefore, the Profile, Process or Instance of an abstract service is subclasses of the Profile, Process or Instance of its parent service. In the following sections, we present more details of our extension.

3.1 Extension to Root Level OWL-S Framework Ontology

In order to connect the Instance section to its Service, a new predicate "implementedBy" is introduced to the root level OWL-S ontology framework. This links an Instance Class to its Service Class. The Instance Class contains the instance pool information for the abstract Service. Additionally, some new constructs (see Section 3.3) are introduced in ServiceModel which is the parent class of the Process. The new Service class with our extension is shown as follows. Figure 2 shows the visual RDF schema of the Service.owl[1], where our extensions are marked.

```
<!-- Service Implementation -->
<owl:Class rdf:ID="ServiceInstance">
<rdfs:label>ServiceInstance</rdfs:label></owl:Class><owl:Ob
jectProperty rdf:ID="implementedBy">
   <rdfs:domain rdf:resource="&service;#Service"/>
   <rdfs:range rdf:resource="&service;#ServiceInstance"/>
   <owl:inverseOf rdf:resource="&service;#implements"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="implements">
   <rdfs:domain rdf:resource="&service;#ServiceInstance"/>
   <rdfs:range rdf:resource="&service;#Service"/>
   <owl:inverseOf
rdf:resource="&service;#implementedBy"/>
</owl:ObjectProperty>
```

- **ServiceInstance**: the root class that represents the instance pool of an abstract service. Its subclass contains all reference information of the candidate instances for that particular abstract service.
- **implementedBy**: An object property extended for a Service class. The resource of this property points to a ServiceInstance.
- **implements**: An object property of a ServiceInstance class. It is an inverse property of implementedBy.

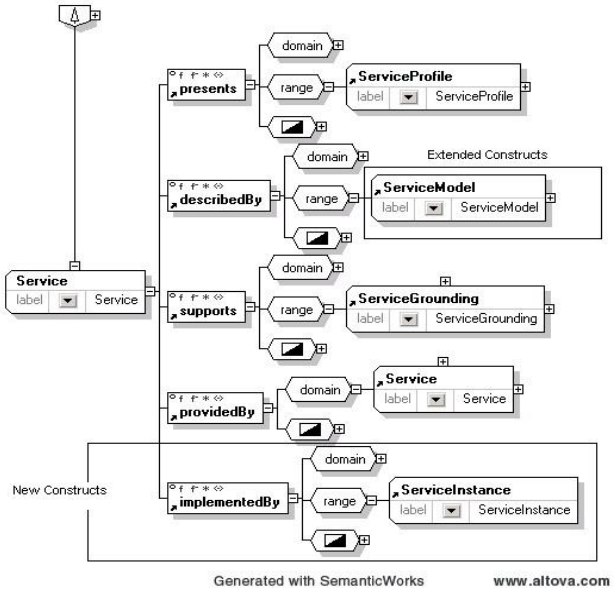


Figure 2 .Visual RDF Schema for Extended OWL-S Upper Ontology

3.2 Instance Class

As discussed previously, the Instance class has a brand new set of information introduced to describe the information of the instance pool of the available concrete service for the corresponding abstract service. It is a subclass of "ServiceInstance" shown in Figure 2. Figure 3 shows the visual RDF schema for the Instance class:

- **Instance**: A subclass of the ServiceInstance shown in Figure 2. It is the root class for all abstract service "Instance". The overall relationship to other ontology framework objects can be found in Figure 2.
- **preferenceOrder**: A data property of the Instance class. It specifies the user's preference when choosing the candidate service. It currently has four

[1] In the remainder of this paper, we only show the visual RDF schema without the corresponding RDF file to save space and for better visualization.

possible values: **Sequential, RoundRobin, Random** and **PriorityCode.**

- **ProcessCandidate**: A class that represents an implementation instance. Each ProcessCandidate points to a real web service in the Internet, such as the "AA301" service mentioned in Section 3.
- **businessOwner**: a data property of the ProcessCandidate class that specifies the owner of concrete web service.
- **priorityCode**: a data property of the ProcessCandidate class that specifies the priorityCode assigned. The value of priorityCode is greater than or equal to 0 with 0 being the highest priority. If the value of preferenceOrder is set to PriorityCode, the system takes this property value to determine which candidate will be chosen at runtime.
- **processRefID**: a data property of the ProcessCandidate class that specifies the id of a concrete web service. It is an optional reference to the physical web service.
- **serviceType**: a data property of the ProcessCandidate class. Currently three types exist: Simple, BPEL and OWL-S.
- **AccessPoint**: A class that represents the access method of a concrete web service.
- **protocolType**: a data property of the AccessPoint class. The possible value can be "HTTP", "FTP" etc.
- **accessLocation**: a data property of the AccessPoint class that specifies the access address of a concrete web service. The system can use this location to retrieve the detailed information of a concrete service.
- **processCandidate:** An object property for an Instance Class. The resource of this property points to a ProcessCandidate class.
- **accessPoint:** An object property for a ProcessCandidate class. The resource of this property points to an AccessPoint class.

3.3 New Constructs in Process.owl

For each abstract service, the composition logic may not be unique. The abstract service needs to have the capability to represent multiple composition logics, which is not possible with the current "Process" class. Therefore, we introduce a new "AbstractProcess" class in the Process section as marked in Figure 4. This class provides a collection of Process logics, like different itineraries from New York to Los Angeles flight example in Section 3. It has an "abstractComposeOf" property pointing to a ControlConstruct with different flow logics. The ControlConstruct maintains a collection of available composition solutions, which can be one process (like an AtomicProcess, a SimpleProcess or a CompositeProcess) or a combination of processes and control constructs.

- **AbstractProcess**: A class represents the possible composition logics. In current OWL-S framework, an actual Process is a subclass of the union of AtomicProcess, SimpleProcess and CompositeProcess. With the introduction of this new AbstractProcess, we are able to describe multiple composition solutions in the Process section of an abstract service.
- **abstractComposedOf:** an object property for AbstractProcess. A collection of ControlConstructs are linked to an AbstractProcess via this abstractComposedOf.
- **ControlConstruct:** relocated from the CompositeProcess of the existing OWL-S Framework which can be used to describe a composition flow for one unique solution. Moving this construct from CompositeProcess to here allows us to describe a collection of possible solutions. In the "New York to Los Angeles" Process, for example, it may use a sequence list (one subclass of ControlConstruct) to contain all possible itineraries.

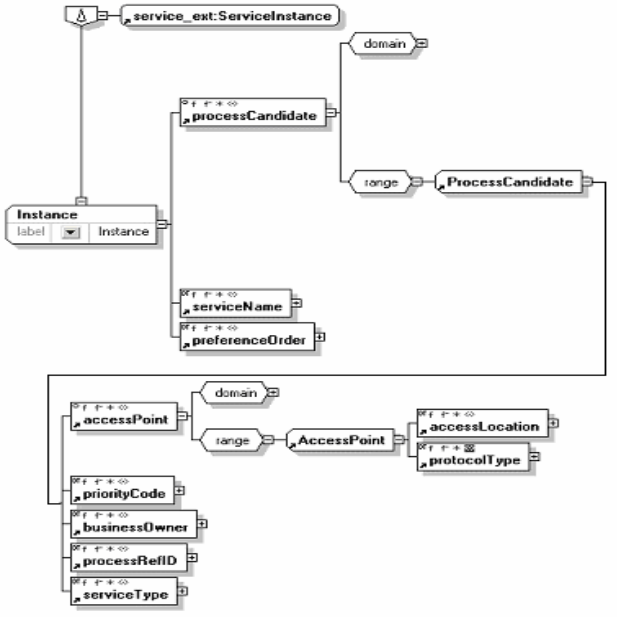


Figure 3 . Visual RDF Schema for "Instance" Class

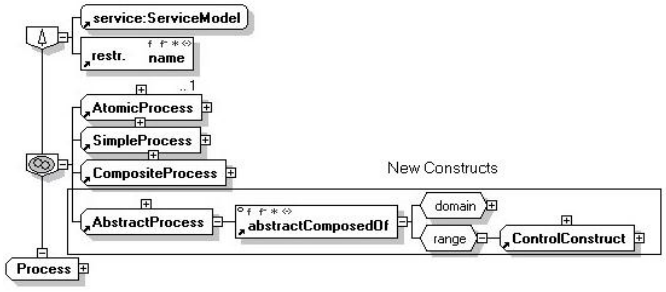


Figure 4 .Visual RDF Schema for "Process" Class

4. Planner Prototype

In order to facilitate the generation of different composite processes, we develop a web-based composer. It utilizes the J2EE web architecture and the Embedded JTP server. The Prototype contains a web interface to gather user's goals. Users can load their special ontologies while the default OWL-S ontology is automatically loaded in the planner. The customer's goals are converted into RDF entries and fed into the JTP inference engine. Consequently, the detailed inference steps are shown on the screen, and the resulting service files are generated when the user finishes the query and clicks the "Generate Result Service" button. Our prototype tool can be used for other service compositions, although we show flight service composition as an example in this paper.

One of the main differences between our extension and the original OWL-S is that we propose abstract service layers. Each specific abstract service needs to connect to its own profile, process and instance while maintaining the inheritance relationship with its parent class. To maintain the correct relationship for different parts of same service, each abstract service uses the "someValuesFrom" "allValuesFrom" and "hasValue" restrictions. Like the following example, AA-Flight_Service, the abstract flight service for American Airline, is connected to AA-Flight_Profile, instead of the Root level Profile Class.

```
<owl:Class rdf:about="#AA-Flight_Service">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:someValuesFrom>
        <owl:Class rdf:about="#AA-Flight_Profile"/>
      </owl:someValuesFrom>
      <owl:onProperty rdf:resource="http://www.daml
      .org/services/owl-s/1.1/Service.owl#presents"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="http://www.daml
      .org/services/owl-s/1.1/Service.owl#presents"/>
      <owl:allValuesFrom>
        <owl:Class rdf:about="#AA-Flight_Profile"/>
      </owl:allValuesFrom>
    </owl:Restriction>
  </rdfs:subClassOf>
  ……..
```

Similarly, we use these restrictions to specify the IOPE for the profile and process of each abstract service. For instance, the profile of "AA-Flight_Service" can be defined with a "hasValue" restriction that sets the serviceParameter property to "aaServiceParameter", which means AA-Flight_Profile has a non-functional service parameter that is set to American Airline. Likewise, "JFK-ORD-Flight_Profile" may have a "hasInput" property with the "hasValue" restriction set to airport "JFK" and a "hasOutput" property with the "hasValue" restriction set to "ORD". Our planner prototype converts the user's goals into backend JTP query string and generates the Process based on the query result. The "and" and "or" operators are used on different levels to combine each JTP query string. Currently the prototype tool specifies three levels where level 1 is the top level. Below shows an example of the functional requirement of the user's goal to get an abstract service which has the departure airport "JFK" (input) and the arrival airport "ORD" (output). The non-functional requirements of the user's goal are that the flight service belongs to American Airlines and has either BusinessCabin or CoachOnlyCabin. The following is the generated JTP queries:

```
(and
  (|http://www.daml.org/services/owl-s/1.1/Process.owl#|::|hasInput|
    ?myFlight |http://127.0.0.1:8080/Parameters.owl#|::|JFK|)
  (|http://www.daml.org/services/owl-s/1.1/Process.owl#|::|hasOutput|
    ?myFlight |http://127.0.0.1:8080/Parameters.owl#|::|ORD|)
  (|http://www.daml.org/services/owl-s/1.1/Profile.owl#|::|servicePara
    meter| ?myFlight
    |http://127.0.0.1:8080/Parameters.owl#|::|aaServiceParameter|)
  (or
    (|http://www.daml.org/services/owl-s/1.1/Process.owl#|::|servicePa
      rameter| ?myFlight |http://127.0.0.1:8080/Parameters.owl#|
      ::|businessCabin|)
    (|http://www.daml.org/services/owl-s/1.1/Process.owl#|::|servicePa
      rameter| ?myFlight |http://127.0.0.1:8080/Parameters.owl#|
      ::|coachOnlyCabin|)
  )
)
```

Users can keep refining their requirements by adding/modifying their goals via our prototype tool. The query results are shown on the screen for review. When users are satisfied with the result, the files containing the final result OWL-S service are generated and added back to the ontology.

Currently, our planner prototype can only handle sequential compositions. When the JTP queries the result for the service components that satisfy the user's goals, the prototype will populate the Process abstractComposedOf property only with the "sequence" control construct. Suppose the user requests a flight service (JFK-LAX) from New York (JFK) to Los Angeles (LAX), for instance, the "JFK-ORD" and "ORD-LAX" services are the query result. We will add support for the other control constructs in the future.

5. Related Work

A number of languages/frameworks have been developed based on the standard W3C web service language to support web service composition. Among

them, two major efforts are the BPEL4WS [5] and OWL-S [2] which define a standard for concrete composite web services. Our work is an extension of OWL-S at the abstract service level.

There are several different approaches in web service composition area. Rao et al. [9] propose architecture for web service composition using the linear logic theorem proving. Both the service profile and customer goals are translated into the propositional linear logic and fed into the Jena planner [6]. The goal realization is based on the individual concrete web service. Similarly, Traverso et al. [11] use the EaGle language and the MBP Planner to generate composition result in π-calculus which is transformed to BPEL4WS. Mandell et al. [8] provide an automatic runtime discovery, composition and execution environment by integrating the BPEL4WS and OWL-S. These approaches focus on concrete web service composition; whereas our approach concentrates on the dynamic optimization and run-time fail over capability.

A transactional approach for web service composition is proposed in [1], where the accepted termination states are defined to allow the user to specify the required failure atomicity level. In contract, our approach focuses on fail-over and dynamic composition instead of failure atomicity.

WSMO (Web Service Modeling Ontology) [3] is another effort to address the semantic web services. WSMO relies on four core components: Ontology, Web Services, Goals and Mediators. There are several differences between the OWL-S and WSMO [7], e.g., separation of provider and requester point of view in WSMO and explicitly use of mediators to link the loose coupling core components. WSMO also describes similar concepts of OWL-S. For example, the OWL-S service profile can be expressed by the combination of the WSMO goal, the WSMO Web Service capability, and the Web Service non-functional properties [7]. Based on WSMO, implementation like WSMX [4] provides an execution environment for semantic web service, which enables the service registration, client discovery and invocation. The service compositions in WSMO, however, are still achieved at the concrete service level. In contrast, our approach focuses on abstract level for fail-over and dynamic optimization.

6. Conclusions

This paper has proposed an extension to the OWL-S ontology framework for dynamic web service composition at abstract service level. Rooted from OWL-S, our approach inherits the capability of semantic clarity. New abstract service concept is extended to the OWL-S. Each abstract service has an instance pool. Our planner prototype can take the user's goals and the service ontology and feed them into the backend inference engine to generate the results that are abstract services instead of concrete services. Each of the resulting abstract services has an instance pool of all possible concrete service solutions. Our planner prototype is based on the embedded JTP, which has been developed for the demonstration purpose.

In the future, we aim to continue enhancing the OWL-S ontology framework to support more complex optimization. We also plan to explore case studies related to both the abstract level and instance level semantic constraints, and the user's goals with "must have" and "good to have" semantic constraints. Furthermore, we intend to integrate our composer prototype with an existing ontology server.

References

[1] Sami Bhiri, Claude Godart, Olivier Perrin: Reliable Web services composition using a transactional approach, in Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Service, Hong Kong, March 2005.

[2] M. Dean, D. Connolly, F. Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, and L. Stein. OWL Web Ontology Language 1.0 Reference. http://www.w3.org/TR/2002/WD-owl-ref-20020729/

[3] C. Feier, D. Roman, A. Polleres, J. Domingue, M. Stollberg and D. Fensel. Towards Intelligent Web Service Modeling Ontology (WSMO). In Proceedings of the International Conference on Intelligent Computing (ICIC) 2005, Hefei, China, August, 2005.

[4] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. WSMX - a semantic service-oriented architecture. In Proceedings of the International Conference on Web Services (ICWS 2005), Orlando, Florida (USA), July 2005.

[5] IBM. BPWS4J. http://www.alphaWorks.ibm.com/tech /bpws4j

[6] Jena - semantic web framework for java. Online: http://jena.sourceforge.net

[7] R. Lara, D. Roman, A. Pollers, and D. Fensel. A conceptual comparison of WSMO and OWL-S. European Conference on Web Services (ECOWS 2004), Erfurt, Germany, September, 2004, pages 254-269.

[8] D. Mandell and S. McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. Proceedings of the Second International Semantic Web Conference (ISWC2003), Sanibel Island, Florida, 2003.

[9] J. Rao, P. Kungas, M. Matskin. Application of linear logic to web service composition. The First International Conference on Web Services, Las Vegas, USA, June 2003. CSREA Press.

[10] Stanford KSL. JTP. http://www.ksl.stanford.edu

[11] P. Traverso, M. Pistore. Automated Composition of Semantic Web Services into Executable Processes. Technical report. # T04-06-08. Istituto Trentino di Cultura, 2004.

WebExplain: A UPML Extension to Support the Development of Explanations on the Web for Knowledge-Based Systems

Vládia Pinheiro[1], Vasco Furtado[1], Paulo Pinheiro da Silva [2], Deborah L. McGuinness[3]
[1] University of Fortaleza, Fortaleza, Ceará, Brazil
[2]University of Texas at El Paso, TX, USA
[3] Knowledge Systems Laboratory, Stanford University, Stanford, CA, USA

***Abstract*—** Knowledge-based systems (KBS) should be able to explain their results to improve the understanding and credibility of their answers by users. However, most KBS explanation components cannot be easily reused by other applications, thus increasing the effort of implementing KBSs with explanation capabilities. In this paper we present WebExplain, an extension to Unified Problem-Solving Method Description Language (UPML), a KBS development framework. WebExplain is integrated with UPML generic components and can be easily reused during the development of other problem-solving methods and KBSs. WebExplain uses the Inference Web for enabling proof and explanation interoperability between distributed applications. We exemplify our approach by describing WebExplain's use in the development of a problem-solving method and a KBS with explanation capabilities.

1. INTRODUCTION

Problem-solving methods (PSMs) describe the reasoning steps and the knowledge roles used during problem-solving processes, regardless of the problem domains. PSMs can be reused by many applications. For example, a single PSM may be used to guide a knowledge acquisition process as well as to describe the design of a knowledge-based system [3].

KBS users need to understand the manner in which solutions provided by a KBS were produced. They then need explanations that describe the PSM flow and the inference steps followed by the KBS in order to produce a result. Despite the development of explanation components by some knowledge engineering solutions (e.g., [1,2,15,17,18]), no known component is able to explain answers from general PSM implementations. Some KBS can generate explanations based on generic tasks or meta-rules, but we are unaware of any explanation approach that provides a systematic way to generate explanations for PSM implementations or that can be easily reused for the development of other KBSs, thus reducing the effort of implementing explanatory KBSs. This paper describes an extension of the Unified Problem-Solving Method Description Language (UPML) [4]—, a KBS development framework, by providing a reusable component with explanation capabilities for KBSs. This component, called WebExplain, has two main characteristics: (i) It is integrated with UPML generic components: PSM, Task and Ontologies (i.e., Method Ontology and Task Ontology). In UPML, a PSM, a Task and their ontologies can be reused in several domains in order to develop several and different KBSs. Hence, we are leveraging UPML and benefiting from the reusability of the UPML infrastructure, thereby reducing the cost of development for Explainable KBS; (ii) It generates justifications for the KBS results in the Proof Markup Language (PML) [11]. PML is the proof format of the Inference Web (IW), an infrastructure for Web explanations enabling applications to generate portable and distributed explanations for any of their answers [8]. PML has an OWL-DL encoding [10], and is thus compatible with semantic web applications in XML, RDF, and OWL. It includes content such as sources, inference rules, inference steps, and conclusions. Thus, proofs in PML can be shared with other distributed applications on the Web, besides using the IW infrastructure and tools to abstract proofs into explanations and to present them to users. KBSs are increasingly being deployed in heterogeneous environments such as the Web—for example, Web Services, and sharing information with other applications. Hence, there is a need for interoperable KBS responses and explanations, like PML.

UPML-based KBS development makes reasoning processes explicit by implementing PSM as part of the applications. Thus, the capability of accessing PSMs at execution time can be leveraged to explain PSM answers at the reasoning level—the strategic level. With the help of UPML patterns, one can enhance the quality of explanations by abstracting away task-specific reasoning steps from proofs and by keeping relevant information for response understanding. Explanations about the domain knowledge are generated from the KBS's inference engine that chains the domain rules.

We exemplify our approach by describing how to have reusability in both the development of PSM and the development of Explainable UPML-based KBS.

2. BACKGROUND KNOWLEDGE

Our approach for Explainable KBS integrates UPML and the IW frameworks.

2.1. Inference Web

Inference Web (IW) [8,9] is a framework for explaining reasoning tasks by storing, exchanging, combining, abstracting, annotating, comparing, and rendering answer justifications[1] provided by reasoners embedded in applications. IW justifications identify the KBS reasoning steps used to derive answers from input information. In addition to a language for answer justification, IW provides an infrastructure that includes: an extensible web-based registry containing details on information sources, reasoners, languages, and rewrite rules; a justification abstractor, and explanation browser. The browser is used to support navigation and presentations of answer justifications. The explainer is used to abstract machine-level justifications into human-level explanations.

PML is the IW justification specification language and includes two major components for building proof trees: inference steps and node sets. A justification can then be defined as a tree of inference steps explaining the process of deriving answers, which are the final conclusions of a justification. Node sets represent both the antecedents and conclusions of inference steps. In other words, an inference step is the application of a single inference rule over a set of antecedents (encoded as node set conclusions) and deriving a consequent (also encoded as a node set conclusion). Each inference step contains pointers to the inference rule and variable mappings used.

2.2. UPML

The UPML framework [4] supports KBS modeling from reusable components, adapters, development guidelines, a description language, and tools. Distinct KBS software components are described by the UPML architecture:

- PSM component that defines the control structure responsible for the coordination of subtasks, i.e., the definition of the subtask execution order;
- Task component that defines the problem that should be solved by the KBS. Subtasks executed by the PSM component are also task components that implement the procedure in order to solve one part of the overall problem. Normally, the subtasks are implemented through an inference engine that executes the rules of the domain's knowledge base;
- Domain model component that describes the KBS domain knowledge, such as, domain rules;
- Ontology component that provides the terminology used in other UPML components including the Method ontology, Task ontology, and Domain ontology.
- Bridge component that establishes the relationships between two distinct UPML components. For example, the bridge between a subtask and the Domain Model sends the domain's concepts and rules to be used in resolving the subtask. This component allows the Task components and PSM component to be implemented completely detached from the Domain Model, and only receive the domain's knowledge and concepts at execution time.
- Refiner component that specializes a UPML component for a specific application.

UPML provides an approach to KBS development strongly centered on the reuse of its generic components and the use of ontologies. To resolve a knowledge intensive problem, a KBS developer can identify and reuse a specific PSM for a problem. The selected PSM must be already defined and implemented using the UPML architecture's generic components: PSM, Task, and Ontology. The developer is left with the task of defining the Domain Model and the Domain Ontology. Moreover, the growing library of generic UPML components eases the development of PSM and KBS for other tasks. For example, Pinheiro, Furtado & Furtado [12] describe a set of UPML generic components implemented in Java including the abstract-and-match PSM [16] used for KBS that solve assessment problems.

3. A UPML COMPONENT FOR KBS WEB EXPLANATIONS

3.1. General Description

In this section we introduce WebExplain, a UPML Explanation Component, responsible for generating PSM and Tasks justifications in PML. WebExplain justifications support two kinds of explanations: about the structure of the reasoning process implemented by PSMs (strategic explanations); and about the execution of subtasks, which are executions of domain's rules and concepts (domain explanations).

Figure 1 shows how WebExplain interacts with the PSM and Task components. Generally, WebExplain receives the subtask that is to be executed from the PSM component and associates all the inference steps generated from that point up to this subtask. Associations between the PML node sets and subtasks in justifications will ultimately determine the PSM's order of execution. WebExplain receives the rules that have been fired from the Task component (subtasks). Rules are provided along with their conditions and actions and recorded as inference steps of each subtask. Note that the justification generation process does not interact with the Domain Model, i.e., the inference steps are received from generic UPML components. For this reason, WebExplain is domain-independent and can be reused by other applications.

WebExplain is composed of the following classes:

- The ProofGeneration class that is used for building inference steps from parameters received from the PSM and Task components. These parameters are represented as variable bindings in the justifications. This class is responsible for encoding the current state of some of the PSM variables as sentences in the justifications. PSM sentences are written in the Knowledge Interchange Format (KIF).

[1] The terms *justification* and *proof* are used interchangeably in this paper.

- The Proof class represents a step in a justification. Inference steps are recorded from ProofGeneration and are identified by a conclusion and a set of antecedents. The conclusion of an inference step can be either derived or asserted. If the conclusion is derived, the class keeps information about the inference engine and inference rule used to derive the conclusion, e.g., JEOPS and modus ponens, and information about the premises.. If the conclusion is asserted, the class keeps information about the source, e.g. a domain ontology. To identify the inference engine generating an inference step, we have created a relationship between this class and its subclasses, i.e., UPMLProof, JEOPSProof, JESSProof, etc. A single justification generated by the ProofGeneration class can represent inference steps generated by multiple inference engines such as JEOPS [5] or JESS [7], as well as inference steps generated by the PSM control structure.
- The IWHandler class is responsible for mapping justifications, which are composed of Proof objects, into node sets and for generating PML documents.

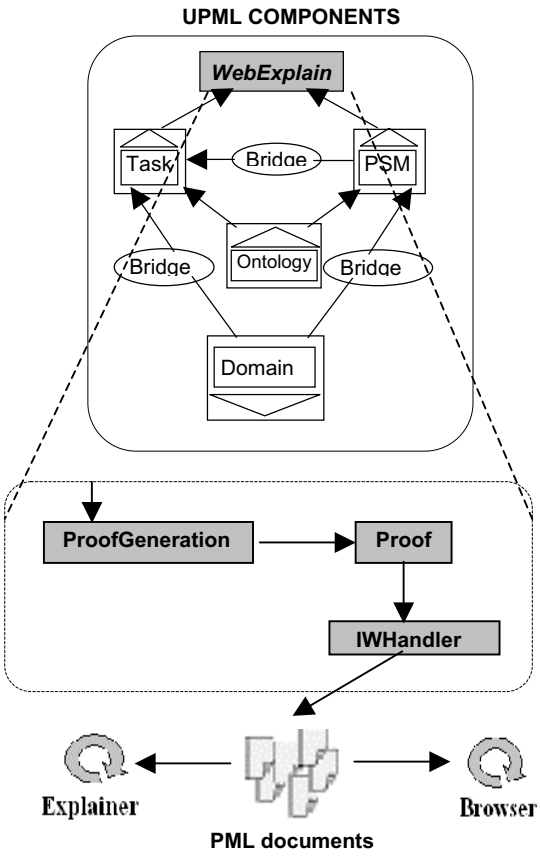


Fig. 1. The UPML original framework extended with WebExplain**.**

It is important to point out that the reasoning steps represented in PML are explored through IW services such as Explainer and Browser.

3.2. WebExplain Representation

PSM justifications can be translated into explanations since each node set in the justification is associated with the subtask wherefrom it was generated and to the node sets holding the antecedents. Moreover, the justification identifies the order in which the subtasks were executed, forming a proof tree that portrays the order defined in the PSM.

Domain explanations are generated from the inference steps associated with subtasks. For instance, a subtask can be implemented by means of the inference engine that executes rules of the domain's knowledge base or by means of an algorithm. We will call these two cases, respectively, subtask implementation A and B. Due to the generality of our approach. WebExplain should not present restrictions to KBS developers regarding the inference engines that can be used. Information referring to the conditions and actions of each rule can be retrieved from a rule ontology without the need for customizing a specific inference engine.

Figure 2 presents the classes that define the rules ontology. The Rule class defines domain rules with the name, description, rule-type, actions, and conditions slots. The actions and condition of a rule are expressions of class *Expression* defined by a name, description, expression-type, domain-variable, operator, and value slots. The domain-variable slot represents a concept of the domain manipulated by an expression. This slot is of class *Element*, which is a concept from the Domain Ontology. Instances of these classes form the domain knowledge base, whose semantic interpretation is defined by the ontology of the domain and the conditional (conditions → actions).

Tools like Protégé are widely used by the Knowledge Engineering community for creating ontologies. Moreover, tools of this category often possess plugins [14] that generate ontologies in several formats including OWL, Jess, and JEOPS. These capabilities offer two important advantages for our solution: (i) since rules are instantiated from the Rules ontology, they can be generated automatically in the format that inference engines happen to process; (ii) users of KBS or knowledge engineers can edit and evolve the domain's rules knowledge base, with no need for knowledge of inference engine languages.

The process of generation of proofs steps executed by the WebExplain is made up of the following steps:

1. It receives, from the subtask, the rule fired by inference engine (subtask implementation A). This is possible because every inference engine possesses a service, generally called Listener, that informs the sequence rules fired;
2. It executes a method of its ProofGeneration class that has access to the rules ontology to retrieve information on the conditions and action of the rule fired, as well as the classes of the domain manipulated by the rule;
3. It executes a method of its ProofGeneration class that inserts the steps of proof as objects of the Proof class based upon the conditions and actions of the rule. For example, let **p** be a wff (well-formed formula) that represents the set of the rule's conditions and let **q** be a wff that represents the set of the rule's actions. As the rule was fired, all of the rule's conditions were satisfied by facts **r,s,t**..., therefore the truth-value of **p** is true. By the rules ontology, the truth-value of (**p** → **q**) is true. Therefore, WebExplain can

insert the proof steps **q,r,s,t**..., and (**p** → **q**) as Proof objects. Proof step **q** represents a derived conclusion and is generated with information on its antecedents **r,s,t**..., e (**p** → **q**), the ModusPonens inference rule used in its deduction and the inference engine used.

4. It returns to Step 1 for the next rule fired, associating all of the proof steps to the current subtask.

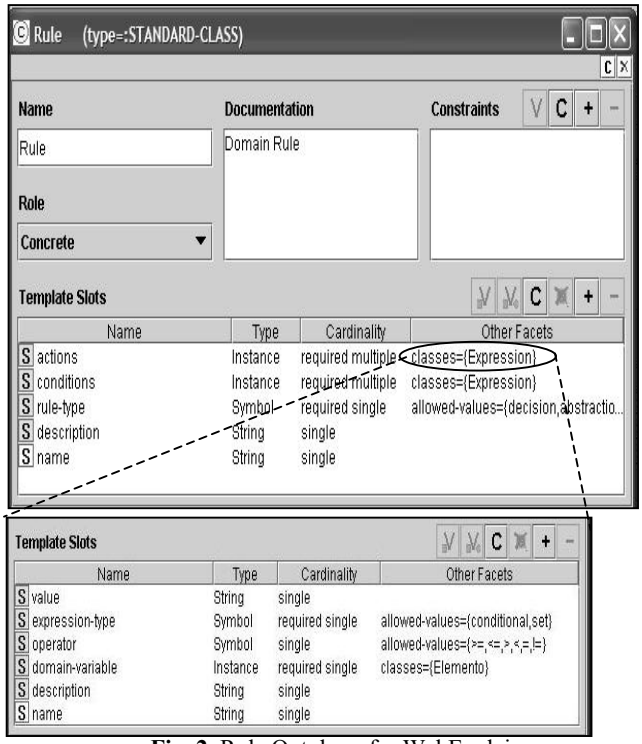


Fig. 2. Rule Ontology for WebExplain**.**

In the case where the subtask is implemented by means of an algorithm (subtask implementation B), the implementation of the algorithm must define the conditions and actions for executing each reasoning step embedded in subtask and the generation process is started from step 3, described above.

The integration between WebExplain and the generic UPML components—PSM and Task—is defined at the time of implementing the PSM, therefore the effort is performed only once, and is ready to be reused in the development of several Explainable KBS.

4. USING WEBEXPLAIN AS SUPPORT TO THE DEVELOPMENT OF EXPLAINABLE KBS

We have two kinds of reuse in our approach: the reuse of the WebExplain component in the modeling and implementation of different PSM, and the reuse of these PSM in the development of different KBS which implies, consequently, the extensibility of the WebExplain to several KBS and in its consolidation as a means of support to Explainable KBS developers.

In this section, we exemplify our approach by describing its use in the development of the abstract-and-match PSM, for assessment tasks, and in the development of an Explainable KBS—the ExpertCop System [6].

4.1. Using WebExplain in the Development of a PSM

The abstract-and-match PSM is defined conceptually in [16]. Its reasoning process defines a control structure executing the following subtasks sequentially:

1. Abstract that simplifies the case data;
2. Specify that finds criteria relevant to the case data;
3. Select that selects one criterion for evaluation;
4. Evaluate that evaluates the select criterion with respect to the case data;
5. Match that checks whether the criteria that were evaluated lead to a decision. The select, evaluate, and match subtasks are interactively executed for each criterion until a decision can be found or the criteria set is exhausted.

Basically, the AbstractMatch java class implements the PSM reasoning process through a control structure that is responsible for sequencing the subtasks. This class extends the PSMComponent class which contains generic methods to perform the mapping with the other UPML components and, among others, a method to execute the calls to subtasks—executeSubTask method. In this method, the integration with WebExplain was inserted calling a method from the ProofGeneration class, which receives the subtask that is to be executed. This integration ensures that the proof tree to be generated in PML mirrors the reasoning structure embedded in the PSM. At the end of the execution of subtasks, the AbstractMatch class invokes the publish method of the IWHandler class responsible for generating PML documents corresponding to the proof tree.

The abstract-and-match PSM subtasks are implemented as subclasses of the Java TaskComponent class and all possess a method called execute, which contains the implementation of the reasoning part destined to each one. In this method, commands must be inserted for integration with WebExplain: receive from inference engine each rule fired and call a method of the ProofGeneration class to access the rules ontology (subtask implementation A) or define the conditions and actions for executing each reasoning step (subtask implementation B); and insert the proof steps corresponding to the execution of the rule as objects of the Proof class. The Abstract, Evaluate and Match subtasks were implemented by means of an inference engine and the Specify and Select subtasks were implemented via algorithm.

The abstract-and-match PSM also define the following knowledge roles: case description, criteria to be evaluated, abstraction rules, evaluation rules, and decision rules. These knowledge roles were implemented as java classes containing generic properties and methods to receive, as parameters, the corresponding instances from classes from the domain's ontology and the rules ontology.

The development of a KBS for assessment tasks using the

UPML components in any domain can thus reuse the PSM implementation, freeing developers for implementing only domain-specific classes and for defining those domain-specific knowledge roles.

4.2. Using WebExplain for Developing an Explainable KBS

The ExpertCop System [6] is a UPML-based KBS example performing an assessment task implemented by the abstract-and-match PSM. In ExpertCop's decision-making process, a criminal cognitive agent must evaluate data gathered from a geo-simulated environment using a set of criteria to decide whether or not to commit a crime. This system is used to teach police officers about when and where crimes are likely to be committed.

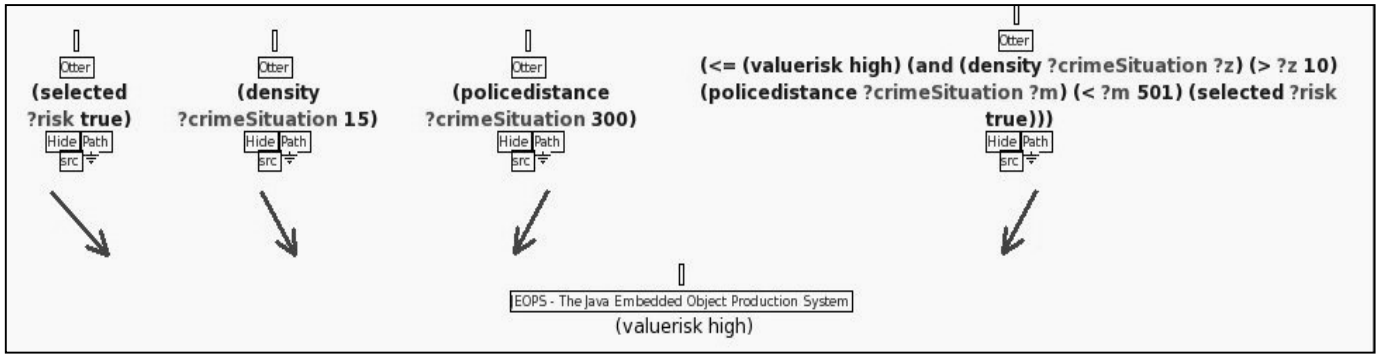


Fig. 3. Nodesets generated by the WebExplain from the subtask evaluate communication about the execution of the rule **evaluateRiskHigh**.

The developers of the ExpertCop system reused the entire implementation of the abstract-and-match PSM and its integration with WebExplain. They only had to do the domain ontology modeling and the domain rules as subclasses of knowledge roles defined by the PSM: the case description was modeled by the CrimeSituation class, the criteria to be evaluated class was modeled as CrimeCriterion, and the abstraction rules, evaluation rules, and decision rules were modeled in the rules ontology that defined rules of three types: abstraction, evaluation, and decision. These rules are processed by the JEOPS inference engine, and passed on as parameters, at execution time, to the Abstract, Evaluate, and Match subtasks.

Let us take as an example the rule evaluateRiskHigh of the evaluation type. This rule possesses the following conditions:

- CrimeSituation.density > 10;
- CrimeSituation.policeDistance < 501;
- Risk.selected = true;

and the following action:

- Risk.truthValue = high

In Figure 3, the IW browser presents a justification, in KIF, for the fact that valueRisk is high. The node sets were generated by WebExplain from the execution of the rule evaluateRiskHigh, which was chained by the subtask evaluate. The fact "valueRisk high" is encoded in a PML node set conclusion and each inference step in the node set corresponds to a fact justification. When rendering the fact justification, the browser shows that the answer, through an application of the modus ponens inference rule, was inferred from the following facts (that correspond to the rule conditions):

- (policeDistance ?crimeSituation 300)
- (density ?crimeSituation 15)
- (selected ?risk true)

and the rule itself:

- (<= (valueRisk high)
  (and (density crimeSituation ?z) (> ?z 10)
  (policeDistance ?crimeSituation ?m) (< ?m 501)
  (selected ?risk true)))

5. RELATED WORK

Explanations for KBS have appeared as a significant and independent topic of study since MYCIN [1]. NEOMYCIN [2] contributed to explanation research in KBS by using an explicit representation for problem resolution strategies and by using meta-rules in explanation planning. By using meta-rules, NEOMYCIN separated the domain ontology from MYCIN's rules. This separation allowed NEOMYCIN to be more usable as an explanation infrastructure; however it was specific to MYCIN in terms of problem solving and domain representation.

The use of Ripple Down Rules (RDR) [15] presents a new paradigm for KBS in which cases are used to explain an answer to a query. RDR provide only cases as representation for explanation which is a domain-specific representation. Moreover, RDR tools, like browsers, are specific for this knowledge representation technique. Another aspect to point out is that this approach was only used in classification tasks. Therefore, it is not trivial to extend it to other knowledge tasks such as design.

WOZ [17] is a framework for explaining component-based decision-support systems. The framework is composed of functional components that represent the reasoning process, the associated cooperative visualization

agents responsible for explanation presentation and user interaction, and application domain models such as user model, agent model, and explanation strategy. WOZ incorporates some of the major trends in software engineering including explicit models, multi-agent architectures, and visualizations. However, WOZ is not easily scalable, since a new explanation strategy must be developed for each application.

Similar to our approach is the Explanation Expert System (EES) [18] framework that provides explanations about the manners the KBS used the PSM in a certain domain. As our approach does, its Explanation Generator component provides clarifications when its explanations are not understood. The major difference is that we use a web-based infra-structure for explanation which could be useful in the development of problem solvers in general (web services, agents, etc.). Moreover, this infra-structure benefits of portability. It can be shared with other applications and opens the possibility to cooperative explanations.

6. CONCLUSION

In this paper, we address the issue that KBS need explanation components and that we are not aware of any that provide a systematic way to generate explanations of PSM implementations or that can be easily reused for the development of other applications, thus reducing the implementation effort of the explainable KBS. So, we propose an extension of the UPML framework by providing a reusable explanation component—WebExplain. WebExplain is integrated to the UPML generic components: PSM and Task. Hence, we are leveraging UPML facilities as reuse and ontologies. WebExplain provides proofs from the reasoning process embedded in the PSM and subtasks, which serve as a basis for explanations about the reasoning structure and explanations about the domain's knowledge. Another characteristic of WebExplain is the use of PML for dumping proofs, therefore enabling proof and explanation interoperability between distributed applications.

We exemplify our approach in the development of a PSM: how should the integration between WebExplain and generic UPML components be implemented. We identified three points of easy integration. This effort was performed only once and the PSM, integrated to WebExplain, can be reused, freeing Explainable KBS developers from the worry of having to implement only domain-specific classes.

Future works aim at projecting some pragmatic principles of linguistic interactions onto the semantic structures of the PML proofs in order to select the information to be conveyed to users, to simplify proof steps and to reorganize proofs, in order to improve the quality of the explanations.

REFERENCES

[1] B. Buchanan and E. Shortliffe. Rule based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project, Addison-Wesley, Reading, MA, 1984.

[2] W. Clancey. From GUIDON to NEOMYCIN and HERACLES in Twenty Short Lessons: ORN Final Report 1979-1985, AI Magazine, 7(3), pp. 40-60, 1986.

[3] D. Fensel and V.R. Benjamins. Key Issues for Automated Problem-Solving Methods Reuse. 13th European Conference on Artificial Intelligence, ECAI98, Wiley & Sons Pub., 1998.

[4] D. Fensel et al. The Unified Problem-Solving Method Development Language UPML. Knowledge and Information Systems, An International Journal, 5, 83-127, 2003.

[5] C. Figueira Filho and G. Ramalho. Jeops – The Java Embedded Object Production System. IBERAMIA-SBIA 2000. LNAI 1952, Berlin: Springer-Verlag, 2000.

[6] V. Furtado and E. Vasconcelos. A Multi-Agent System to Teach Police Allocation. Proc. of 17th Innovative Application of Artificial Intelligence (IAAI-2005), Pittsburgh, 2005.

[7] JESS. http://herzberg.ca.sandia.gov/jess, as available on March 10th, 2006.

[8] D.L. McGuinness and P. Pinheiro da Silva. Infrastructure for Web Explanations. In Proceedings of 2nd International Semantic Web Conference (ISWC2003), D. Fensel, K. Sycara and J. Mylopoulos (Eds.), LNCS 2870, Sanibel Is., FL, USA. Springer, pages 113-129, October 2003.

[9] D.L. McGuinness and P. Pinheiro da Silva. Explaining Answers from the Semantic Web: The Inference Web Approach. Journal of Web Semantics. Vol.1 No.4., pages 397-413, October 2004.

[10] D.L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation, February, 2004.

[11] P. Pinheiro da Silva, D.L. McGuinness and R. Fikes. A Proof Markup Language for Semantic Web Services. Information Systems, Volume 31, Issues 4-5, June-July 2006, Pages 381-395. Also, Stanford KSL Technical Report KSL-04-01.

[12] V. Pinheiro, E. Furtado and V. Furtado. A Unified Architecture to Develop Interactive Knowledge Based Systems. In proceedings of the 17th Brazilian Symposium of Artificial Intelligence (SBIA 2004), Bazzan, Ana L.C. e Labidi, S. (Eds), LNAI 3171, São Luís, MA, Brazil, Springer-Verlag, pp 174-183, 2004.

[13] Protégé: http://protege.stanford.edu, as available on March 10th, 2006.

[14] Protégé Plug-ins: http://protege.stanford.edu/download/plugins.html, as available on March 10th, 2006.

[15] D. Richards. User-Centred and Driven Knowledge-Based Systems for Clinical Support Using Ripple Down Rules. Proceedings of the 33rd Hawaii International Conference on System Sciences, 2000.

[16] G. Schreiber, H. Akkermans, A. Anjewierden, R. Hoog, N. Shadbolt, W. van de Velde and B. Wielinga. Knowledge Engineering and Management: The CommonKADS Methodology. The MIT Press. Cambridge, MA, 2000.

[17] R.D. Shankar, S.W. TU, M.A. Musen. A Declarative Explanation Framework That Uses A Collection Of Visualization Agents. Stanford Medical Institute, Stanford University School of Medicine, Stanford, CA, 1998.

[18] W. Swartout, C. Paris and J. Moore. Explanations in Knowledge Systems: Design for Explainable Expert Systems. IEEE Expert: Intelligent Systems and Their Applications, vol. 06, no. 3, pp. 58-64, Jun., 1991.

A Use Case Model and its Transformation to Activity Diagram

Xing-Yi Lin [a], Ching-Hui Wang [b], William C. Chu [a], and Chihhsiong Shih [a]
[a] *TungHai University, Taiwan*
[b] *National Chiao Tung University , Taiwan*
mailto:bowji@itlab.csie.thu.edu.tw, chinghui@csie.nctu.edu.tw, chu@csie.thu.edu.tw, and shihc@csie.thu.edu.tw

Abstract

Use cases play a critical role for functional requirements elicitation. In present researches, most of the use case templates employ informal definitions and text-based description. Further requirements analysis process also depends on manual labor. In this paper, we present a use case model which includes two sub-models, use case diagram model and use case template model. We also propose a concrete natural language syntax for the use case operation. In addition, the tAD-Algorithm proposed by this paper can transform the use case scenario into activity diagrams automatically.

Keyword: use case diagram, use case template, UML 2.0 activity diagram, requirement analysis

1. Introduction

Use cases have been used broadly in requirements elicitation and verification. Due to the characteristics of use cases, a use case model was proved to be an effective technique for acquiring functional requirements. Use cases can be modeled through diagrams [1] and textual specifications. Diagrams emphasize on relationships between actors and use cases. Textual descriptions depict the behavior scenario details in a use case. However, there is no UML standard for a use case specification. In order to make an effective communication with stakeholders and precise requirements analysis, it is necessary to provide a form of use case template.

Many previous researches [2][3][4][5][6][7][8][9] have proposed diverse use case templates or use case frameworks. Most of them provide information such as use case name, actors, precondition, postcondition and main flow. Nakatani et al. [2] proposed a use case framework to construct the use case metamodel. Maiden and Rodertson [3] used RESCUE process to discover how use cases and scenario were developed. In order to support scenario-based requirement engineering, Penna et al. [4] provided a use case structure to collect scenario. However, most of the use case templates are described in natural language. This increases the difficulty in requirement analysis. For this consideration, Gelperin [5] provided a precise form of use case using a Structure English grammar. Stéphane [6] proposed concrete natural language syntax for use case descriptions.

In addition, the main flow depicts the sequence of a use case scenario. It is not easy to represent the logic or the structure of a use case scenario by textual description. A picture is sometimes worth a thousand words, though there is no substitute for clean, clear prose [7]. To allow for representing the structure of functional requirements, Garcíe et. al. [8] proposed an alternative approach to use case specifications based on interaction modeling. Another considerable researches have already been done on how to transfer a use case specification document into a state machine [6][9]. However, a use case scenario describes not only behavior performed by actors but also data invoked by behaviors. In UML 2.0 standard, activity diagram can depict not only basic control flow but also data flow. In addition, activities are redesigned to use a Petri-like semantics in stead of state machines [10]. Consequently, many researches have been proposed to transform activities into Petri-net for requirements verification and validation [11][12].

In this paper, we proposed a use case model based on UML 2.0. It contains two sub-models: use case diagram model (UCD model) and use case template model (UCT model). As regards to the detail description in a use case, we extend the concrete natural language syntax proposed by Stéphane [6]. In addition, we proposed the tAD-Algorithm to automatically transform use case scenario into activity diagrams.

2. Use case model

OMG provides the concepts used for modeling use case [10]. The elements of a use case diagram include use cases, actors, and relationships. A use case is a functional requirement. An actor specifies a role that interacts with the use cases. The relationships (e.g., association, generation, inclusion, and extension) create relation between actors and use cases. However, the use case diagram of UML 2.0 (UC2) does not comprise detail description formula of a use case. We provide a use case model shown as Figure 1. The model includes two sub-models, UCD model and UCT model.

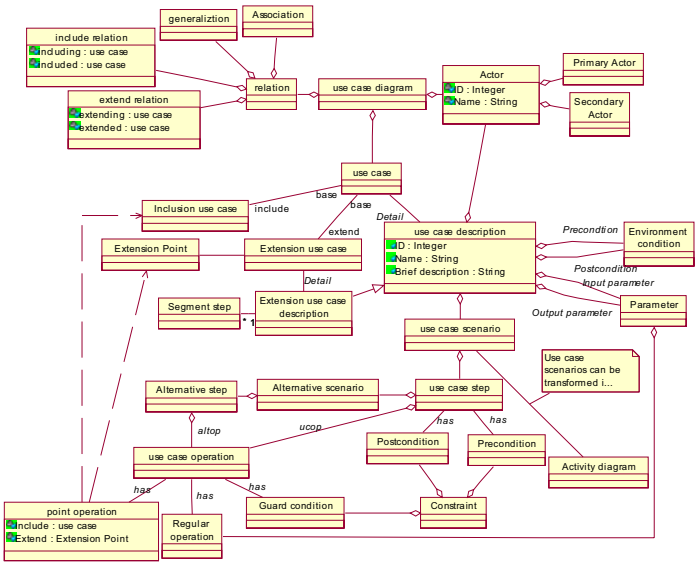


Figure 1. Use case model

2.1. UCD Model

UCD model describes elements of a use case diagram from user perspective in accordance with UML 2.0 specification. A use case diagram is composed by three kinds of element which are *actor, use case,* and *relation.* A use case may be involved with multiple actors, such as human users, external hardware, or other subjects. However, different actors interacting with the same use case could generate variant scenarios of a use case. For this consideration, it is necessary to identify the primary actor of a use case. There are four kinds of relationship between actors and use cases, such as association, generation, inclusion, and extension.

2.2. UCT model

UCT model provides a template for detail descriptions of use cases. Each use case has a unique ID and an identify name, and brief description about what the purpose of a use case. The primary actor is the main role of the use case scenario. There are some other actors who could be involved in the use case. These actors are defined as secondary actors. A precondition is the state of the system that must be present prior to a use case being performed [7]. In contrast, a postcondition is a list of possible states the system can be in immediately after a use case has finished. In previous researches, some use cases are not triggered or finished, when the environment conditions are satisfied. They are triggered by data input and finished when data output. Consequently, a use case template should also contains information about inputparameter and outputparameter.

Use case: ReturnBook
ID: 3
Brief description: The Librarian returns a book.
Primary actor: Librarian
Secondary actors: None.
Preconditions: The Librarian has logged on to the Library system
Input parameter: Null
Use case scenario: 1. Include(FindBookDetail) 2. IF the book is overdue THEN Extension Point: overdueBook Extension Point: payFine ELSE GOTO Step 3 3. The Librarian returns the book.
Postconditions: The Librarian returns the book.
Output parameter: Null
Alternative step: None

Segment 1 Use case scenario: 1. The System shows the detail of [unpaid] fine* 2. The Librarian select “Print out the fine”. 3. The System ACCEPT EVENT @print. 4. The System prints out the fine*.
Segment 2 Use case scenario: 1. The Librarian gets the payment. 2. The Librarian records the [paid] fine*. 3. The System {timestamp} on the fine* to <<transfrom>> a receipt*. 4. The Librarian select “Print out the receipt”. 5. The System ACCEPT EVENT @print. 6. The System prints out the receipt*.

Figure 3. An example of use case ReturnBook and a potion of extension use case FineProcess

<usOperation> − > <RegularOP> | <GuardOp> | <PointOp>
<RegularOp> − > <Operation> | <RegularOp> “AND” <RegularOp>
<GuardOp> − > <Branch> | <Loop> | “BREAK” | “GOTO” "Step” <id>{according to the label of a step}
<Branch> − > “IF” <RegularOp> “THEN” <usOperation> | <Branch> <elseBranch> | <Branch> <elseifBranch> <elseBranch>
<elseifBranch> − >“ ELSE IF” <RegularOp> “THEN” <usOperation>
<elseBranch> − > “ELSE” <usCoperation>
<Loop> − > “FOR” <Operation> “DO” <usOperation> | “WHILE” <Operation> “DO” <usOperation>
<PointOP> − > “Include” “(”<use_case_name>“)” | “Extension Point:” <extension_point_name>
<Operation> − > <commonOp> | <objectOp> | <signalOp> | <compareOp>
<commonOp> − > [<determinant>] <entity> <verb>{in present tense} <word>
<objectOp> − > <simpleoOp> | <actoOp> | <condtionoOp>
<signalOp> − > [<determinant>] <entity> <signal act> “@” <signal_name>
<compareOp> − > <word> <comparator> <word>
<simpleoOp> − > [<determinant>] <entity> <verb>{in present tense} <objectword>
<actoOp> − > [<determinant>] <entity> “{” <object_effect> “}” <objectword>
<conditionoOp> − > [<determinant>] <entity> “{” <object_effect> “}” <objectword> “to” “<<” <condition> “>>”
<determinant> − > “a” | “an” | “the”
<entity> − > <actor_name> | <objectword>
<condition> − > “transformation” | “selection” | “multicast” | “multireceive”
<objectword> − ><object_name> “*” | “[” <object_state> “]” <object_name> “*”
<signal_act> − >“ACCEPT EVENT” | “SEND SIGNAL” | “WAIT TIME EVENT”
<comparator> − > “<” | “>” | “=” | “<=” | “>=” | “greater than” | “less than” | “equal to” | “different to” | “greater or equal to” | “less and equal to”

Figure 4. Grammar for use case operation

Each step in use case scenario has its precondition, postcondition, and operation. Preconditions are conditions that must be true before the starting of an operation, and postconditions are conditions that will be true after the operation has finished. Use case operation describes the behavior of a step. The next section will present the detail of the concrete syntax for use case operation. Alternative scenario specifies a possible variation of a use case after a step, including exceptions, error situations, interruptions, and other less common events. Figure 3 is an example of a use case ReturnBook. The use case is filled out followed by the template provided by UCT model.

3. Concrete syntax for use case operation

We extend the concrete syntax proposed by Stéphane [6]. There are three types of operation in use case operation, *regular operation, guard condition,* and *point operation.* Regular operation describes simple behavior of an actor. Guard condition contains branching, loop, goto, and break statement. Point operation describes which use case would be included or extended. Figure 4 outlines our grammar for use case operation.

Regular operation can be distinguished into four types of operation: *common operation, object operation, signal operation,* and *compare operation.* Common operation describes a simple behavior of an *entity*. It starts with the optional *determinant* followed by an *entity*, a *verb*, and *word*. An *entity* refers an actor or an object. The *verb,* in present tense, with *word* depicts the behavior of an *entity*. Object operation describes objects or data invoked by behavior.The *object_effect* represents what behavior an *entity* does to an object and the *object_state* denotes the states of an object. Compare operation describes a condition which needs to be satisfied. Signal operation describes an event or a signal occurred. It can be distinguished into three types of signal operation: *accept event, send signal,* and *wait time event.* In addition, there are four types of guard operation which are *branch, loop, break,* and *goto*. Point operation describes which use case is included and which segment in a use case could be extended.

4. Activity diagram model

According to UML 2.0, an activity diagram is composed by activities which are networks of nodes connected by edges [13]. Consequently, the definition of an activity diagram is as follow.

Definition 1. An activity diagram AD is a tuple (E,N), where $E=\{e_1,e_2,e_3,\cdots,e_p\}$ is a finite set of activity edges, and $N=\{n_1,n_2,n_3,\cdots,n_q\}$ is a finite set of activity nodes.

According to UML 2.0, there are two categories of edge and three categories of node. We denote $eCate=\{cf,of\}$ as a set of edge category, where cf represents control flow edge, and of represents object flow edge. $nCate=\{cn,on,an\}$ is denoted as a set of node category, where cn represents control node, on represents object node, and an represents action node. Additionally, each activity has multiple attributes, denoted as the form $x._{Attr}=y$, where y is the value of

an attribute $Attr$ of an activity x. On the basis of the statements mentioned above, the definitions of edge and node are as follow.

Definition 2. An activity edge $e_j \in E$ is of the form $e_j = cate|_{AV}$, where. $cate \in eCate$, and $AV = \{a := V_a \mid (a \in eAttr) \wedge (V_a = e._a)\}$.

Let $eAttr = \{inN, outN, Condition\}$ be a set of edge attribute, where inN and $outN$ represent input node and output node, $Condition$ represents conditions on the activity edge which includes guard, selection, transformation, multicast, and multireceive condition.

There are different types of node in each node category. Let $nType(cate)$ be a set of node types in category $cate$. We assume the following notations are used.

- $nType(cn) = \{iCN, afCN, ffCN, dCN, mCN, fCN, jCN\}$ is a set of node type in a category of control node, including initial node iCN, activity final node $afCN$, flow final node $ffCN$, decision node dCN, merge node mCN, fork node fCN, and join node jCN.
- $nType(on) = \{oON, ipON, opON, apON, cBON, eON\}$, is a set of node type in a category of object node, including single object node oON, pin pON, input activity parameter $ipON$, output activity parameter $opON$, centralBuffer node $cBON$, and expansion node eON;
- $nType(an) = \{ssAN, aeAN, ateAN, caAN, cbAN, coAN\}$ is a set of node type in a category of action nodes, including send signal action node $ssAN$, accept event action node $aeAN$, and accept time event action node $ateAN$, call an activity action node $caAN$, call a behavior node $cbAN$, call an operation node $coAN$.

Definition 3. An activity node $n_j \in N$ is of the form $n_j = {}_{cate}\, type|_{AV}$, where $cate \in nCate$, $type \in nType(cate)$, and $AV = \{a := V_a \mid (a \in nAttr) \wedge (V_a = n._a)\}$.

Let $nAttr = \{ID, \Pr e, Post, in, out, Op, state, effect, multi\}$ is a set of node attribute, where

- ID denotes the identification of an activity node;
- $\Pr e$, $Post$ denotes the precondition of the source node and the postcondition of the target node;
- in, out denotes the name of input object and output object;
- Op denotes the detail description of an activity node.
- $state$ denotes the state of an object node;
- $effect$ denotes the effect an action on an object nod;
- $multi$ denotes a two bits binary code. The first bit denotes input or output, the second denotes an object is sent to multiple receivers or objects are received from multiple senders.

5. tAD-Algorithm

Figure 5 shows the tAD-Algorithm. The notations used in tAD-Algorithm are defined in previous sections. The essential elements of a use case in tAD-algorithm are defined as follow.

Definition 4. Let use case UC is of the form $UCS = (u\Pr e, inPara, ucS, outPara, uPost)$, where

- $u\Pr e = \{p_1, p_2, p_3, \cdots p_l\}$ is a finite set of precondition;
- $inPare = \{iP_1, iP_2, iP_3, \cdots iP_l\}$ is a finite set of input parameter;
- $ucS = (s_1, s_2, s_3, \cdots, s_l)$ is a steps sequence of a use case scenario;
- $outPare = \{oP_1, oP_2, oP_3, \cdots oP_l\}$ is a finite set of output parameter;
- $uPost = \{q_1, q_2, q_3, \cdots q_l\}$ is a finite set of postcondition.

TransformAD(uPre, inPara, usS, outPara, uPost){
Transform uPre into an initial node $n_0 =_{cn} iCN|_{\{\Pr e := u\Pr e\}}$;
For each $iP_j \in inPare$ {
(1) Transform each iP_j into an input activity parameter $n_j =_{on} ipON|_{\{Op := iP[j]\}}$;
(2) $e = cf|_{\{inN := n_j, outN := n_k\}}$, where $n_j ==_{cn} iCN$ or $n_j ==_{on} ipON$, k = 1 + the number of iP_j;}
AD_1 = ActionTransfer (AD, usS);
Transform uPost into an activity final flow $n_l =_{cn} afCN|_{\{Post := Post\}}$, where l = 1 + the number of nodes ;
k = 1;
while(the transformation occurs)
AD_{k+1} = RefineAD(AD_k);}

Figure 5. The tAD-Algorithm

RefineAD(AD){
For each node $n_i \in N$ {
If($n_i \cdot_{Op} \in signalOp$){
(1) $n_j =_{an} ssAN|_{\{Op:=signal\_name\}}$, if(signal_act == "SEND SIGNAL");
(2) $n_j =_{an} aeAN|_{\{Op:=signal\_name\}}$, if(signal_act == ACCEPT EVENT");
(3) $n_j =_{an} ateAN|_{\{Op:=signal\_name\}}$, if(signal_act == WAIT TIME EVENT");}
elseif(($n_i \cdot_{Op} \in simpleoOp$) or ($n_i \cdot_{Op} \in actoOp$)){
k = 1 + number of nodes of N;
(1) $n_i =_{an} cbAN|_{\{out:=n_k\}}$ and $n_k =_{on} oON|_{\{Op:=object\_name,state:=object\_state,effect=object\_effect\}}$;
(2) $e = cf|_{\{inN:=n_i,outN:=n_k\}}$;}
elseif($n_i \cdot_{Op} \in conditionoOp$){
(1) $n_i =_{on} oON|_{\{Op:=object\_name,state:=object\_state,effect:=object\_effect\}}$;
(2) $e = of|_{\{inN:=n_x,outN:=n_i,condition:=condition\}}$, where $n_{x \cdot op} == entity$;}
elseIf($n_i \cdot_{Op} \in Branch$){
$n_i =_{cn} dCN$;
For each Branch{
$e = of|_{\{inN:=n_i,outN:=n_k,condition:=RegularOp\}}$, where k = 1 + number of nodes of N;
Get the steps sequence $bucS$ from each branch;
Action(N, E, busS);}
elseif($n_i \cdot_{Op} \in Loop$){
$n_i =_{cn} dCN$;
$e = of|_{\{inN:=n_i,outN:=n_k,condition:=RegularOp\}}$, where k = 1 + number of nodes of N
Get the steps sequence $lucS$ from usOperation;
Action(N, E, lusS);
$n_k =_{cn} mCN$, where k = 1 + number of nodes of N;}
elseif($n_i \cdot_{Op}$ contains "GOTO")
Delete node n_i and link pre-node with node id by edge $e = cf|_{\{inN:=pre-node,outN:=n_{id}\}}$;
elsif($n_i \cdot_{Op}$ contains "Include")
$n_i =_{an} caAN|_{\{Op:="Call an Activity("use\_case\_name")"\}}$;
elseif($n_i \cdot_{Op}$ contains "Extension Point"){
Get the steps sequence $eucS$ from extension_point_name;
Action(N, E, eusS);}}
return AD;}

Figure 6. The refinement process of tAD-Algorithm

The tAD-Algorithm proceeds in phases. The first phase is to produce a preliminary activity diagram. The procedure *TransformAD* transforms use case precondition into an initial node, and transforms use case input parameters into input activity parameter nodes. Each use case scenario step is transformed into an action node by the procedure *ActionTransform*. Activity final flow node is transformed from use case postcondition. After the first time transformation, a preliminary activity diagram AD_1 has been created. The second phase is to refine AD_1 by the procedure *RefineAD* which is shown in Figure 6.

The Procedure *RefineAD* can transform each action node into a proper node type according to the use case operations and the definitions of activity nodes. If the operation of a node contains multiple use case operations, each operation can be transformed into an action node by the procedure *ActionTransfer*. In addition, If the k_{th} phase transformation occurs, the $(k+1)_{th}$ phase is to refine AD_k. The refinement process terminates until there is no transformation occurring. Figure 7 is an example of an activity diagram transformation from a use case ReturnBook. which is extended by use case FineProcess.

6. Conclusions

This paper proposes a use case model and an algorithm for automatic activity diagram transformation. The use case model includes two sub-models. UCD model

describes essential elements of use case diagram by user's perspective. UCT model provides a template for detail descriptions of use cases. To solve the problem of natural language, we extend the concrete natural language syntax proposed by Stéphane [6]. The tAD-Algorithm is proposed to automatically transform use case scenario into activity diagrams. According to previous researches [11][12], activity diagram and Petri-net are used for requirement verification and validation. In the future, we will further transform an activity diagram into color Petri-net or object-oriented Petri-net for the integration and consistency of divers use cases. We also plan to integrate use case with other UML 2.0 interaction diagrams for further requirements analysis.

References

[1] I. Jacobson et al. *Object-Oriented Software Engineering,* Addison-Wesley, 1992.

[2] T. Nakatani, T. Urai, S. Ohmura, and T. Tamai, "A requirements description metamodel for use cases," *the Proceedings of APSEC'2001,* 4-7 Dec., 2001, pp. 251 – 258.

[3] N. Maiden and S. Robertson, "Developing use cases and scenarios in the requirements process," the Proceedings of ICSE'2005, May 15-21, 2005, pp. 561 – 570.

[4] G. D. Penna, B. Intrigila, A. R. Laurenzi, and S. Orefice, "An XML Definition Language to Support Scenario-Based Requirments Engineering," *International Journal of Software Engineering and Knowledge Engineering*, Vol. 13, No. 3, 2003, pp. 237-256.

[5] D. Gelperin, "Precise Use Cases," *LiveSpecs Software,* 2004.

[6] Stéphane S. Somé, "Supporting use case based requirements engineering," *Information and Software Technology,* Vol. 48, No. 1, Jan. 2006, pp. 43-58.

[7] P. Kruchten, *The Rational Unified Process 2nd ed.*, Addison-Wesley, 2000.

[8] J. D. García, J. Carretero, J. M. Pérez, F. García, and R. Filgeira, "Specifying use case behavior with interaction models," *Journal of Object Technology,* Nov.-Dec. 2005, Vol. 4, No. 9.

[9] P. Fröhlich and J. Link, "Automated Test Case Generation from Dynamic Models," *Lecture Notes of Computer Science,* Vol. 1850, 2000, pp. 472-491.

[10] Object Management Group (OMG), "Unified Modeling Language: Superstructure version 2.0," *available at* http://www.omg.org, 2005.

[11] H. Störrle, "Semantics and Verification of Data Flow in UML 2.0 Activities," *Electronic Notes in Theoretical Computer Science,* Vol. 127, No. 4, 21 Apr. 2005, pp. 35-52.

[12] R. Eshuis, and R. Wieringa, "Tool support for verifying UML activity diagrams," IEEE Transactions on Software Engineering, Vol. 30, No. 7, Jul. 2004, pp.437 - 447

[13] J. Arlow and I. Neustadt, *UML 2 and the Unified Process Second Edition Practical Object-Oriented Analysis and Design,* Addison-Wesley, 2005.

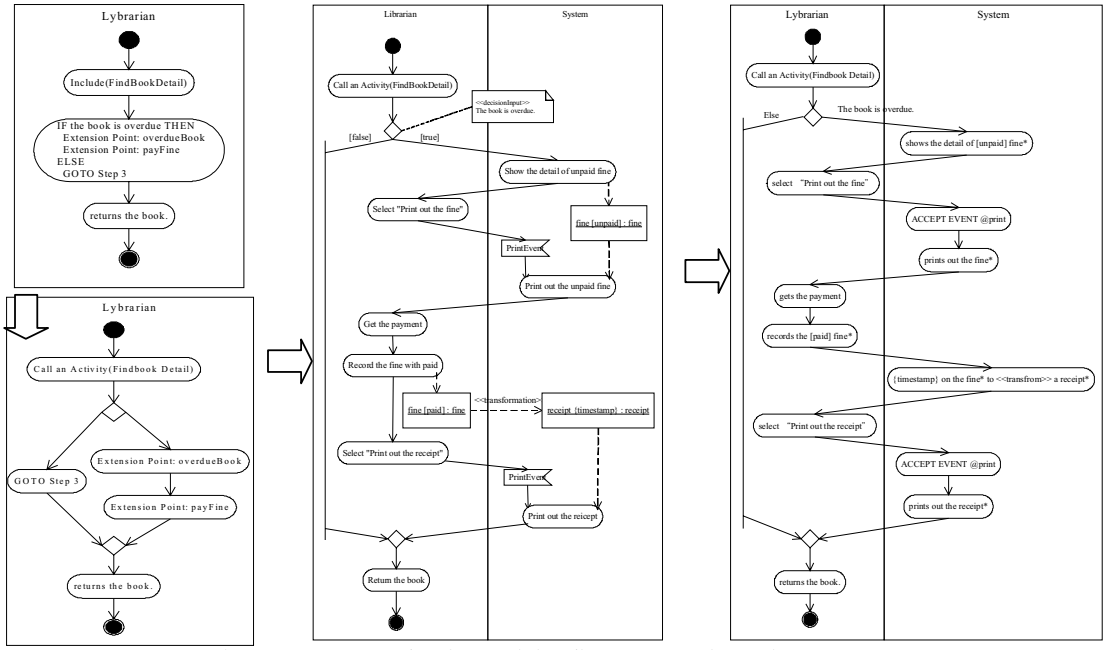


Figure 7. An example of an activity diagram transformation process

An Analysis Model of Activity Diagram in UML2.0

W.C. Piao[*a], C.H. Wang[b], William C. Chu[a], Lung-Pin Chen[a]
[a]*TungHai University, Taiwan*
[b]*National Chiao Tung University, Taiwan*
bunki@itlab.csie.thu.edu.tw, chinghui@csie.nctu.edu.tw, chu@csie.thu.edu.tw, and lbchen@thu.edu.tw

Abstract

In UML2.0, the activity diagram is reengineered based on Petri Net, it is not only the flow chart, but also has the token to represent the flow of control, object, and some data. This makes it useful in modeling and analyzing in the requirement phase. However, the activity diagram in UML2.0 is still lack of the formal semantics. Formalization of activity diagram provides the ability of system execution flow collision checking, execution flow ambiguous detection, user execution scenario generation, and so on. Moreover, formal model also supports systematic analysis mechanism. Although there are some articles discussed this problem, but all of them did not base on the last UML2.0 standard and did not provide a complete solution. This paper provides an activity diagram formalization model and uses this model to exhibit how to optimize the instance of activity diagram.

Keyword: UML2.0 activity diagram, requirement analysis, requirement verification, formalization, Object Petri Net (OPN)

1. Introduction

During the last several decades, there has been a dramatic growth of software engineering technologies. Many researches indicate that the earlier the software engineering process is introduced, the more budget is saved. This is because the cost curve of the same incorrect element which was detected in the requirement phase to testing phase is exponential. Many industries notice the importance of process improvement, and put many efforts. Being the first phase of a project, *requirement analysis* takes a very important place, the completeness and correctness of requirements will influence the following phases.

It is unreasonable to produce a product that does not meet the customer's needs. Therefore, a series of methods like VORD [1], prototype, scenario-based [2], system-based, and service-oriented approaches are developed to help constructing the requirements from different viewpoints. After constructing the draft requirement, the problem the business might happen is the change of requirements. That means actions such like modifications and interactions will impact on relative requirement documents. The result might be the inconsistency, ambiguity, and incorrectness of the requirements. RE models, DFD, and UML are tools in modeling and analyzing kinds of processes. Particularly, UML has been used in modeling the software process from requirement to design phases a long time. UML is a graphical tool easy to represent the elements relationships and understand than natural languages and formalization, respectively.

Activity diagram is one kind of languages in modeling the business processes and workflows of requirements. In the version of UML1.5, activity diagram was just a special case of state machine, and usually used as flow charts, short of using worth. However, in the UML2.0 [3], activity diagrams were enormously reengineered, the revision activity diagrams became useful to decompose requirements processes into analysis and design phases. There are signal notations, exception handlings, object flows, and parameters set up in UML2.0 activity. Moreover, the significant addition to activity diagrams is the token idea, and we can use the tokens like Petri Net tokens to trace the activity flows in UML2.0.

Requirements have to be controlled when development. The requirement documents may reduce to many sub requirements and grow up individually in the different departments in real world. In addition to the modeling processes, *verification processes* prove the correctness of the requirement and prevent from inconsistent [4, 5, 6, 7]. In the requirement analysis and verifications, it is possible to be perplexity in the duplicated documents checking, documents signature paths, new or modified elements in the requirements. For saving human resources and detecting the implicit errors, the analysis tasks should be translated into automation [5, 6, 8]. However, the revision UML is still semi-formal. Because it is a difficult and time-consuming task to check all elements in whole documents, we need to transfer the activity diagrams to formal languages and check the inconsistency based on the formal languages properties [9]. This paper translated activity diagrams in UML2.0 to

a formal language, called Object Petri Nets [10], and provided a case study of this formal analysis model to verify the inconsistence.

The rest of this article is organized as follows. Section 2 introduces the activity diagrams in UML2.0 and analysis models transformations. Section 3 summarized the analysis model and transferring definitions. Section 4 used the analysis model to verify the inconsistency of a simple example. Finally, conclusions are made in Section 5 .

2. Activity Diagrams in UML2.0

Activity diagrams are reengineered based on Petri Nets. According to the metaclasses of activities, the BasicActivities and FundamentalActivities semantics are not changed greatly. The other changes are illustrated as follows. There are three types of nodes: *action node*, *control node*, and *object node*; and two kinds of edges that is *control flows* and *object flows*. *Preconditions* and *postconditions* are adopted to restrict the input/output state of an activity. Actors also have the same constraints that called *local preconditions* and *local postconditions*. In UML2.0, *action node* has four types: *call action node*, *send signal node*, *accept event action node*, and *accept time event action node*. The usual actions we used in modeling are *call action nodes* which only invoke other activities in the past, but now can also call behaviors or operations. *Send signal* and *accept event action nodes* have the features to represent the asynchronous processes. A *send signal node* sends a signal without waiting for receipt, and an *accept event action node* receive events continuously when the activity is active. An *accept time event action node* can be taken as a kind of an *accept event action* responds to time, when the time expression of an *accept time event action node* becomes true, and it generates an event and causes the follow actions to excuse. *Control nodes* act as traffic switches including *initial nodes*, *activity final nodes*, and *flow final nodes* to start or end activities or flows; in addition, the number of such tokens is not limited to one anymore.

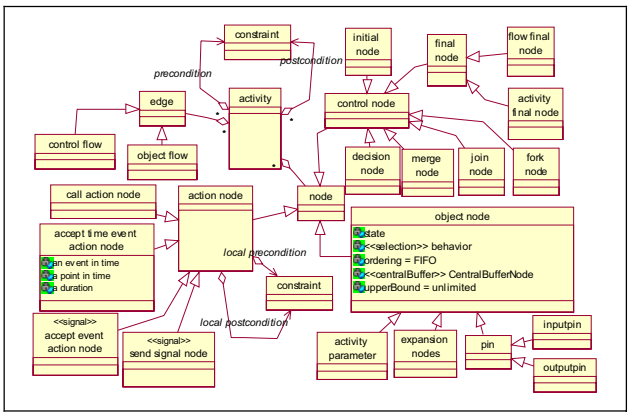


Fig 1. Activity Diagram Metamodel

Furthermore, the *guard conditions* on the edges incident to the *decision nodes* make use of modeling contingency, there is only one path can be choose, and perhaps merge in the merge node. In contrast, *fork nodes* and *join nodes* synchronize the concurrent flows, like AND gates. The most significant change of semantic must be the appearance of *object node*. *Object node* represents the objects act in the flow. It appears almost anywhere and anytime you want to indicate. An object node can be used in variety ways. A *parameter*, a *pin*, and a *expansion node* are also kinds of *object nodes* depending on where objects are flowing from and to, and depicted as similar small rectangles. These *object nodes* can complete the action behavior, but in this paper we focus on the attributes and executions of the original type of *object node* to avoid the multiple and confused representations. *Object nodes* indicate the instances of a particular classifier, specify the state of required tokens, identify the upper bound of the number of tokens, carry the control value, select token for outgoing edges, support for token selection, order and type the tokens, and act as buffers. Fig 1 shows the activity diagram metamodel.

3. Analysis model

3.1. Sorts of Petri Net Transformation

There are a few of studies in transferring activity diagram in UML 1.5. John and Sol [11] proposed a Petri Net based transformation approach to support formal modeling and analysis of complex UML statecharts. Eshuis and Wieringa [8] proposed a tool for verifying functional properties of complex workflow models specified in UML activity diagrams. Saldhana and Shatz [12] proposed a methodology to develop a Petri Net model of a system, by deriving a form of OPN called as Object Petri Net Models (OPMs) from UML Statechart diagrams and connecting using UML Collaboration diagrams. Miyamoto and Kumagai [13] presented state-of-the-art on OO-nets in their article. Lee and Park [14]

use an approach, called OPNets for modeling real-time systems based on the object-oriented formalization of high-level Petri Nets. The modeling experiences with OPNets demonstrate that the decoupling and separation of knowledge and constraints clearly enhances maintenance and reusability in real-time system modeling. Harald [15] tried to find out what kind of Petri Net the activity is, and he then divided the whole activity into four parts: control flow, data flow, exception node, and expansion node. To simplify the problems, the control flows and data flows are usually considered separately. Furthermore, the last two parts of activity is closer to more detail analysis phase even design phase, but it didn't address the problem of control/data-flow interaction [16].

A Coloured Petri Net (CPN) [17] is a high-level Petri Net used to model system specifications in graphic representations. However, it still lacks OO characteristics. There are many researches regarding to the extensions of CPNs with OO abilities [14]. The extended CPNs were formalized to model, simulate, and verify particular systems.

Object Petri Net is one of the extended CPNs with a complete integration of OO features such as inheritance, polymorphism and dynamic binding. Because activity diagrams have the object oriented concepts, OPN is useful for this paper in formalizing activity diagrams. Comparing OPN to traditional P/T nets and CPN, OPN is more suitable in formalizing OO activity diagrams, and has solution to control/data flow interaction. However, at this point there is no one-and-only recognized OPN formalism, but they have in common. They all have general idea of OO, and the same concepts which are two transitions, two semantics, and synchronization relation. This paper adopts the EOS which is the earlier OPN approach by Valk, and OPN which Farwer provides and solves the control/data flow interaction as the activity diagram semantic domain.

3.2. Object Petri Net

OPN is a nets-within-nets paradigm. It uses the nets as the tokens. An OPN is composed of two kinds of nets and a synchronization relation, the first net is the token nets called object nets (ONs), and the other is the higher level nets called system nets (SNs). The two kinds of nets could be Petri nets, coloured Petri Nets or others. OPN provides two transitions which are *autonomous* and *synchronous* transitions. An autonomous transition does not affect others in OPN and can be adopted in both the system nets and object nets (shown in Fig 2 (a)). In contrast, when synchronous transitions are fired, the tokens of relevant places move at the same time. Synchronization relations create the relationships between transitions. The relevant places directly connect to these transitions. The details are shown in Fig 2 (b). Moreover, an object net can be interpreted into two semantics, *reference semantics* and *value semantics*. Reference semantics means an object net refers to the other one that locates in different place(s), and the markings of these nets are the same. When the marking of any object net is changed, the remainder nets are also changed. On the other hand, for the value semantics, the markings of the object nets are treated as independent copies.

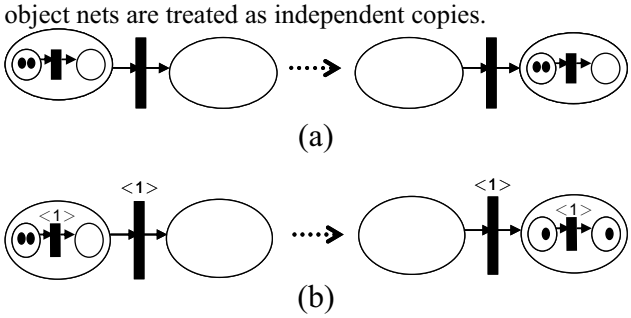


Fig 2. Synchronization relation. (a) Autonomous transition firing. (b) Synchronous transition firing.

An object node in activity diagrams could be display as object nets in Object Petri Nets which support observing object nodes' life cycle, synchronizing the token movements among the corresponding places, and controlling the attributes (*e.g.* state, upper bound, selection condition, *etc*.) of activities. Therefore, we can simplify an activity without representing the object nodes in whole diagrams, and still have rules to trace and control the object nodes. In this paper, our model only applies the OPN to a two level hierarchy, which means the constructions of SN and ONs are CPNs. This paper uses coloured Petri Nets to configure the system nets and object nets to represent the different types of object nodes on various activities and different attributes of object nodes on object flows.

Definition 3.1 (SN) A system net is a 7-tuples SN $\langle \Sigma, P, T, A, C, G, E \rangle$ where:

1. Σ set of types (coloures) with reflexive and transitive subtype relation,
2. P set of places,
3. T set of transitions,
4. A set of arcs, such that $P \cap T = P \cap A = T \cap A = \varnothing$,
5. C total function $P \rightarrow \Sigma$, set of colour classes,
6. G guard function, from T into Expressions, such that $Type(G(t))=Bool \wedge Type(Var(G(t))) \subseteq \Sigma$, $\forall t \in T$,
7. E arc function, from A into Expressions, such that $Type(E(a))=C(p)_{MS} \wedge Type(Var(G(t))) \subseteq \Sigma$, $\forall a \in A$, where p is the place,

Definition 3.2 (ON) An object net ON $\langle \Sigma_o, P_o, T_o, A_o, C_o, G_o, E_o \rangle$ is a CPN net where

1. Σ_o set of types (coloures) with reflexive and transitive subtype relation,
2. P_o set of places,
3. T_o set of transitions,
4. A_o set of arcs, such that $P_o \cap T_o = P_o \cap A_o = T_o \cap A_o = \varnothing$,
5. C_o total function $P_o \rightarrow \Sigma_o$, set of colour classes,
6. G_o guard function, from T into Expressions, such that $Type(G_o(t))=Bool \wedge Type(Var(G_o(t))) \subseteq \Sigma_o$, $\forall t \in T_o$,
7. E_o arc function, from A into Expressions, such that $Type(E_o(a))=C_o(p)_{MS} \wedge Type(Var(G_o(t))) \subseteq \Sigma_o$, $\forall a \in A_o$, where p is the place,

Definition 3.3 (OPN) An Object Petri Net is a 3-tuple OPN $\langle$ SN, ONs, ρ $\rangle$ with

1. SN system net,
2. ONs object nets,
3. ρ $T \times T_o$ is the synchronization relation, is given by labels $<>$ at t and e iff $t \rho e$,

3.3. Activity Diagrams Model

We define the activity A $\langle$ $\langle$ Nodes $\rangle$, $\langle$ Edges $\rangle$ $\rangle$. Nodes are defined as follow:

1. AN action nodes include call action node, accept event node, send signal node and accept time event node,
2. ON object nodes include object nodes, signal object nodes,

Control nodes we separate them to initial nodes, transition nodes,

3. IN initial nodes include first nodes, activity final nodes, and flow final nodes,
4. TN transition nodes include fork and join,
5. BN branch nodes include decision and merge nodes,

Edges can separate to action flows and data flows,

6. AE action flows,
7. OE object flows,
8. PC precondition and postconditon of and activity, and local preconditions and local postconditons of actions,

3.4. Transfer activity diagrams to OPN

Definition 3.4 (Mapping to SN)

The mapping from an activity diagram to SN= $\langle \Sigma, P, T, A, C, G, E \rangle$ is described as follows.

1. Σ set of types (coloures) with reflexive and transitive subtype relation,
2. P $ON \cup IN \cup BN \cup PC \cup \{p \mid p \in AE, \{ \bullet p, p \bullet \} \subseteq AN \cup TN\}$,
3. T $AN \cup TN \cup \{t \mid t \in AE, \{ \bullet t, t \bullet \} \subseteq BN \cup IN\}$,
4. A $\subseteq \{ (p \times t) \cup (t \times p) \mid \langle t, p \rangle \in AE \wedge t \in T, p \in P \}$,
5. C $P \rightarrow \Sigma$,
6. G guard conditions of decision nodes of BN,
7. E arc labeling function $F \rightarrow V$.

In the set P defined in Definition 3.4, preconditions, postconditions, local preconditions, and local postconditions are thought to be the states of activities, not an execution transition, so the PC are in the consideration of P. The set C defined in Definition 3.4 is considered as variant nodes, so the corresponding function translates P into Σ to define the total node to coloures classes. The colour classes can recognize the all action node types, and object nodes. Guard conditions in activity diagrams only occur in the decision nodes, although G defined the whole BN, but the merge nodes which share the same notation with decision nodes do not have the guard conditions attributes. In the system net transformation, C function coloures the types of nodes as the tokens of places.

Definition 3.5 (Mapping to ON) The mapping of object net is similar to system net where changed the semantic domain to object nodes.

1. Σ_o set of types (coloures) with reflexive and transitive subtype relation,
2. P_o $ON \cup IN \cup BN \cup PC \cup \{p \mid p \in AE, \{ \bullet p, p \bullet \} \subseteq AN \cup TN\}$,
3. T_o $AN \cup TN \cup \{t \mid t \in AE, \{ \bullet t, t \bullet \} \subseteq BN \cup IN\}$,
4. A_o $\subseteq \{ (p \times t) \cup (t \times p) \mid \langle t, p \rangle \in OE \wedge t \in T_o, p \in P_o \}$,
5. C_o $P_o \rightarrow \Sigma_o$,
6. G_o guard conditions of decision nodes of BN,
7. E_o arc labeling function $F \rightarrow V$.

In the object net transformation as defined in Definition 3.5, the step of constructing C_o coloures the types of object nodes as the tokens of places.

4. Verification

This article takes the regular order process in a company as a simple example to express the requirement analysis verification. The activity diagram (Fig 3.) shows the process of an order, this order process can start by the initial node or the object node. The "Receive Order" action will receive the control flow token after the process

started, and then the flow will be downstream to a decision node, the order will be accepted and the control flow token goes to the main flow or the order be rejected and the token goes to the alternative flow. In the main flow, the order will be filled before the main process fork to two concurrency flows, the token will be duplicated to two tokens to each flows. One sub flow will perform "Ship Order" action while the other flow acts " Send Invoice" action and produce an invoice as the "Invoice" object node for "Make Payment" action, finally the token will be downstream to the "Accept Payment". Whatever one of the two sub flow tokens reach the join node, the token has to wait the other token, and after the join node, the two tokens become to one and the flow goes downstream. The token only pass by the merge node to the next action "Close Order", and finally the token goes to the flow final node. The flow final node will terminate the token and end the whole process.

Fig 3. (b) derives the object flow from activity diagram presented as Fig 3. (a), in order to focus on the object flow state translation. Once an order has been requested, it will be accepted or rejected after evaluation. The accepted order will be filled and shipped, but all orders will be closed at last whatever it was accepted or rejected in the beginning.

After the translation, the corresponding OPNs of the two activity diagrams are showed in Fig 4. This purpose takes the reference semantics of OPNs, the referenced object net will be taken as the reference object net instance, the other object nets which reference to the instance will share the tokens. The semantics is convenience of modeling the shared data systems, the object net instances execute like the central buffer nodes of object nodes.

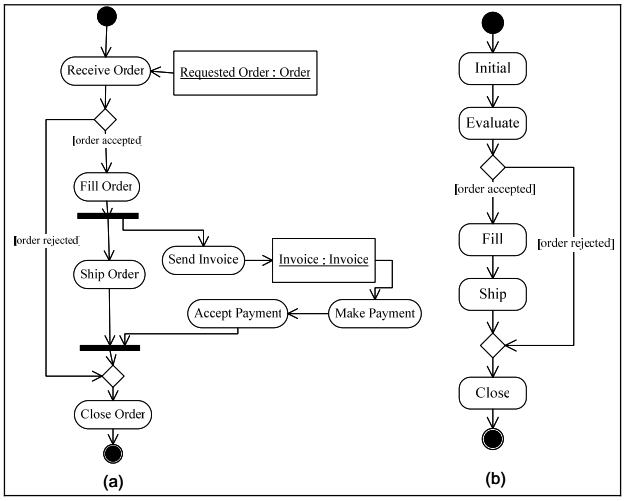


Fig 3. The activity diagrams of Order processes.

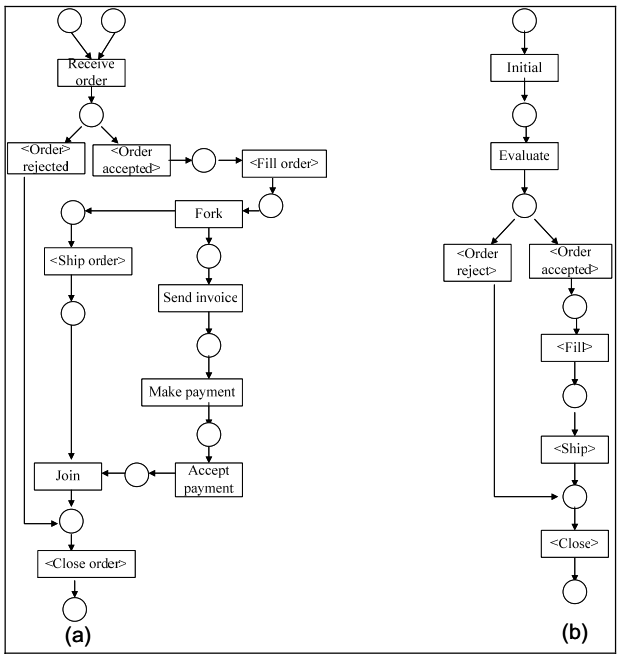


Fig 4. Object Petri Nets. (a) System net. (b) Object net.

The synchronized relation confirms the situation of the separated flows, take an example, the synchronized transition of the system net can not fire until the corresponding synchronized transition of the object net is enabled. Moreover, after "decision" place, if the object net token enables "order rejected" transition, but the system net token enables "order accepted" transition, such that the synchronization relation transition pair will never be fired. The proof drawn below shows the inconsistency relation.

SN is inconsistency to ON where $\{(t1\ \rho\ t2) \wedge (t2 \neq t3) \wedge (\bullet t2 = \bullet t3) \wedge (t1 \text{ and } t3 \text{ are fired}) \mid t1 \in T, \{t2, t3\} \in T_o\}$ or $\{(t1\ \rho\ t2) \wedge (t2 \neq t3) \wedge (\bullet t2 = \bullet t3) \wedge (t1 \text{ and } t3 \text{ are fired}) \mid t1 \in T_o, \{t2, t3\} \in T\}$

5. Conclusion

This study discusses the significant changes of activity diagram in UML2.0, and investigates what kind of Petri Net is suitable for formalizing the activity diagrams. Since the new activity diagram supports the object-oriented processes specified in the reversion of UML, this paper suggests formalizing the activity diagrams by using the OPN (Object Petri Net) which is an extension of the Coloured Petri Net. This implies that the OPN not only has the object-oriented properties but also has the analysis methods supported by CPN. Additional advantage of using OPN is the variety of existing tools such as RENEW[19].

The article derives the activity diagrams into a closure set and translates into OPN. The formalized activity

diagrams can use the analysis methods of Object Petri Net to help finding the implicit fault of requirements, providing concurrence, and solving control/data flow problem. This article discusses a case study about business order processing, and defines one kind of inconsistency by using the OPN properties.

Acknowledgement

The authors would like to thank the National Science Council of the Republic of China for financially supporting this research under Contract No. NSC 94-2213-E-029-003.

References

[1] Gerald Kotonya and Ian Sommerville, "Requirements engineering with viewpoints," *Software Engineering Journal,* vol.11, pp. 5-18, Jan. 1996.

[2] G.D. Penna, B. Intrigila, A.R. Laurenzi and S. Orefice"An XML environment for scenario based requirements engineering," *J. Systems and Software,* 2005.

[3] Object Management Group (OMG), "Unified Modeling Language: Superstructure version 2.0", version 2.0, available at http://www.omg.org, 2005.

[4] C. Heitmeyer, R. Jeffords and B. Labaw, "Automated Consistency Checking of Requirements Specificatons", *ACM Transactions on Software Engineering and Methodology,* vol. 5, no. 3, pp. 231-261, Jul. 1996.

[5] A.P. Felty and K.S. Namjoshi, "Feature specification and automated conflict detection," *ACM Transactions on Software Engineering and Methodology*, vol. 12, no. 1, pp. 3–27, Jan. 2003.

[6] C. Nentwich, W. Emmerich, A. Finkelstein and E. Ellmer, "Flexible consistency checking," *ACM Transactions on Software Engineering and Methodology*, vol. 12, no. 1, pp. 28–63, Jan. 2003.

[7] L.C. Briand, Y. Labiche, L. O'Sullivan and M. Sowka, "Automated impact analysis of UML models," *J. Systems and Software,* 2005.

[8] R. Eshuis and R. Wieringa, "Tool Support for Verifying UML Activity Diagrams," *IEEE Trans. Software Eng.*, vol. 30, no.7, Jul. 2004.

[9] J. Bhattacharyya, A.R. Chaudhuri and S. Bhattacharya, "A formal approach towards systems modeling and verification," *TENCON 2003. Conference on Convergent Technologies for Asia-Pacific Region,* vol. 1, pp. 178-182, Oct. 2003.

[10]http://www.llpn.com/OPNs.html

[11] Z. Hu and S. M. Shatz, "A Transformation Approach for Modeling and Analysis of Complex UML Statecharts: A Case Study ," *Proceedings of the International Workshop on the Applications of UML/MDA to Software Systems (UMSS05)*, Las Vegas, NV, June 2005.

[12] J.A. Saldhana and S.M. Shatz, "UML Diagrams to Object Petri Net Models: An Approach for Modeling and Analysis," *Proceedings of the Int. Conference on Software Engineering and Knowledge Engineering (SEKE)*, Chicago, pp. 103-110, Jul. 2000.

[13] T. Miyamoto and S. Kumagai, "A Survey of Object-Oriented Petri Nets and Analysis Methods," *Proceedings of the IEICE Trans. Fundamentals,* vol. E88-A, no. 11, Nov. 2005.

[14] Y.K. Lee and S.J. Park, "OPNets: An Object-Oriented High-Level Petri Net Model for Real-Time System Modeling," *J. Systems and Software,* no. 20, pp.69-86, 1993.

[15] H. Storrle, "Semantics and Verification of Data Flow in UML2.0 Activities," *Proceedings of Electronic Notes in Theoretical Computer Science*, pp. 35–52, 2005.

[16] D. Dearman, A. Cox and M. Fisher, "Adding control-flow to a visual data-flow representation," *Proceedings of the 13th International Workshop on Program Comprehension*, pp. 297-306, 2005.

[17] K. Jensen, Coloured Petri Nets, Vol 1: Basic Concepts, Springer-Verlag 1992.

[18] W.J. Lee, S.D. Cha and Y.R. Kwon, "Integration and analysis of use cases using modular Petri nets in requirements engineering," *IEEE Trans. Software Eng.,* vol. 24, no. 12, pp. 1115-1130, Dec. 1998.

[19] O.kummer, F.Wienberg, and M. Duvigneau. RENEW-The reference net workshop http:// www.renew.de.

Translation of UML Models to Object Coloured Petri Nets with a view to Analysis

Asghar Bokhari and Skip Poehlman
McMaster University
Computing and Software Department
1280, Main Street West, Hamilton, Ontario, Canada
bokhari@mcmaster.ca, poehlman@mcmaster.ca

Abstract

Higher level Petri Nets that employ object oriented concepts have recently drawn the attention of researchers for use in formalizing UML in order to carryout dynamic analysis and simulation for complex software systems. This paper presents a method for constructing Petri Net models from UML models based on Object Coloured Petri Nets (OCPN). This method results in a Petri net model consisting of class nets matching the standard practice followed by most programming languages. Such a model can be used to check behavioural properties of a system at the design stage.

1 Introduction

Unified Modeling Language (UML) is widely used by software developers for object oriented analysis and design; however, it is based on semi-formal semantics and lacks analysis capabilities whereas Software Engineering (SE) practices require analysis and validation at an early stage in the software development process. The popularity of UML stems from its simplicity and graphical syntax and therefore researchers have concentrated on improving its semantics to provide the software analyst with a dual approach of using UML to create different models that may be transformed into formal models for verification/validation. The development of approaches for analysis of UML models is a significant step for developers who routinely employ UML to create models for their systems.

Petri nets are based on sound formalism and their graphic representation makes them very attractive as analysis tools. They can prove to be a much desired analysis tool for a large class of UML modelers, if they are easily constructed from UML state diagrams. More recently, transformation of UML state diagrams to high level Petri nets that use Object Oriented (OO) concepts, has been proposed [12, 13, 1, 14, 5]; however, analysis tools for such Petri Nets, though in different stages of development, are not readily available [10]. On the other hand, well-developed and tested tools (both commercial as well as research) are available for colour Petri Nets (CPN) [4].

In this paper, we propose a technique to transform UML state diagrams to Object Coloured Petri Nets (OCPN) [9] that can be implemented using one of the popular tools for coloured Petri Nets like Design/CPN or CPN Tools [4]. The rest of this paper is organized as follows. Section 2 is a brief review of the related work. Section 3 discusses details of the algorithm to convert UML state diagrams into equivalent OCPNs, Section 4 discusses how our technique is used for verification of behavioural properties of a system and Section 5 concludes the paper.

2 Motivation and related work

Earlier attempts to provide a formal foundation for UML concentrated on translating UML models to mathematical models using formal languages like Z, HOL and PVS [11]. In [8], the authors use vUML to translate UML state charts into PROMELA, which is the input language to the model checker SPIN. Latella et. al. [7] propose a conversion of the hierarchical representation of UML state diagrams to extended hierarchical automaton (EHA) as an intermediate step and then translate it to PROMELA. More recently it has been recognized that formal methods must be adapted such that developers without deep mathematical knowledge may also use them [11]. This has generated much interest in the translation of UML models to Petri Nets. Most of the proposals suggest algorithms based on a set of rules that can be followed step by step to translate UML models into Petri Nets. We propose a set of axioms based on strong intuitive motivations, to transform a UML state diagram to Object Coloured Petri Nets (OCPNs). These axioms lead to intermediate results in the form of a state transition system rep-

resented by a table that is identical to the table representing the state transition system obtained by [7] using extended heirarchical automaton as explained in [3].

3 Transforming UML State Diagrams to OCPNs

We note that an OCPN for an object-oriented system can be created by first constructing an object net for each object represented by a UML state diagram. These object nets are then transformed into class nets by adding facilities to create tokens corresponding to the requests for new objects and to delete/consume tokens corresponding to the requests for deleting the objects. The class nets are connected together through communication channels to create a Petri net model for the system.

3.1 Overview of the Approach

a. We assume that a state diagram exists for every class in the system and transform each state diagram to a corresponding OCPN by following the algorithm given in Section 3.3. We call the net, so obtained, an object model.

b. Each object model is converted into a class net by adding necessary facilities.

c. Class nets are then integrated into an OCPN representing the system, by two sets of communication channels for asynchronous messages as proposed in [6].

3.2 Definitions and Algorithm

3.2.1 Definitions

Object Model (OM)

An Object Model is a tuple <BM, MP> where BM is a CPN that models the lifetime behavior of an object as specified by the corresponding UML state diagram and MP = { *msg_in, msg_inv, msg_out, msg_rec* } is a set of four places. The place *msg_in* contains all tokens representing messages requesting some service of the current object. The place *msg_inv* contains all tokens representing messages sent by the current object requesting some service. The place *msg_out* contains all tokens representing messages sent by the current object as an acknowledgement or a return value for a message received earlier from another object. The place *msg_rec* contains all tokens representing messages sent to the current object by another object as an acknowledgement or a return value for a message sent earlier by the current object.

Class Net (CN)

A Class Net is an enhanced version of OM that meets the formal definition of a class net given in [9]. It is obtained from the OM by adding:

1. Four transitions ($T_{in}, T_{inv}, T_{out}, T_{rec}$) and four places ($P_{in}, P_{inv}, P_{out}, P_{rec}$) to handle communication between objects.
2. Two transitions *Create and Delete* to handle creation and deletion of tokens representing objects of the class
3. A finite set of instance fusion sets, if necessary, for the class attributes.

We use the terms *input place, output place, input transition, output transition, input arc, output arc* and *a set of global fusion places* as defined in [6].

3.3 Algorithm

Precondition: A UML state diagram is available for each class in the UML class diagram.

Begin 1. For each state diagram

Begin Using Design/CPN

(a) Convert the state diagram into an object model using Algorithm 1

(b) Transform the object model to a class net using Algorithm 2

End

2. Connect the class nets by a set of global fusion places defined as

$$Channel_1 = \Big(\bigcup_{i=1}^{n} P_{ini}\Big) \cup \Big(\bigcup_{i=1}^{n} P_{invi}\Big)$$

$$Channel_2 = \Big(\bigcup_{i=1}^{n} P_{outi}\Big) \cup \Big(\bigcup_{i=1}^{n} P_{reci}\Big)$$

Where n is the total number of class nets.

End

3.3.1 Algorithm 1

Begin For each state diagram

1. If there are any composite states, "flatten" them to simple states.
2. Examine the state diagram and assign a unique identifier to each state transition.
3. Create the following tables for all state transitions (the two tables can be combined if desired):

Table 1. State Diagram Table A

State Transition#	Input State	Output State	Transition Events	Transition Guards

Table 2. State Diagram Table B

State Transition#	Input State Exit Action(s)	Output State Entry Action(s)	Transition Action(s)

4. Start with a new CPN page and create four places named *msg_in, msg_inv, msg_out and msg_rec* and position them at suitable locations on the page, for example, close to the top, bottom, left and right of the page.
5. For each state transition
   (a) Create a CPN transition and transform the guard conditions to CPN guards.
   (b) Create a CPN input place for the input state and a CPN output place for the output state. If a state already exists, do not duplicate it.
   (c) **If** there is a state transition event
   then If this is a local event create a CPN input place for it
   ElseIf the event is a message received from another object then create an input arc from *msg_in* place to this transition and another from a place containing a token of the object receiving the message.
   ElseIf the event represents a returned value from a previous action then draw an input arc from the *msg_rec* place and create an output place for the value returned.
   EndIf
   EndIf
   (d) **If** there is an action for the state transition or an entry action for the output state or an exit action for the input state
   then If it is a local action create an output CPN place for it.
   ElseIf this action is a message (M) for another object draw an output arc to *msg_inv* place from the transition. Create CPN input places for all parameters and draw arcs from these places to the transition. Specify initial markings for these places to represent the parameter values.
   ElseIf an action results in sending of a return value or acknowledgement to another object then draw an output arc to *msg_out* place.
   EndIf
   EndIf

End

3.3.2 Algorithm 2

Begin
1. Create four CPN transitions named T_{in}, T_{inv}, T_{out} and T_{rec} and four CPN places named P_{in}, P_{inv}, P_{out} and P_{rec}.
2. Draw an input arc to T_{inv} and T_{out} from *msg_inv and msg_out* places of the object net respectively.
3. Draw output arcs from T_{inv} to P_{inv} and T_{out} to P_{out}.
4. Draw an output arc from T_{in} and T_{rec} to *msg_in and msg_rec* places of the object net respectively.
5. Draw input arcs to T_{in} from P_{in} and to T_{rec} from P_{rec}.
6. Add facilities to create and destroy objects in the class by adding a *create* and a *delete* transition with input arcs from the *msg_in* place and in case of the create transition, an output arc to a place (AllIDS) that contains all object ids for the class and in case of the delete transition an input arc from this place.
7. Draw an input-output arc between T_{in} and AllIDS to ensure that a message is passed on to an object only if it exists.
8. A create or delete message is handled by the class and it places a token representing the newly created object in the initial place of the net representing the class. In case of deletion, the token is consumed by the transition implementing this message.

End

4 Verification of Behavioural Properties

One of the aproaches for formal verification of a software system consists of proving that an abstract model of the system has a set of desireable properties such as boundedness that ensures the system will reach some stationary behaviour eventually or deadlock-freeness that verifies that global deadlock never happens or home state that verifies the system is able to re-initialize itself or liveness that means

the system never looses its capacities. We have been working on a service oriented architecture for distributed electronic suport systems in which services are implemented as software agents [2]. We successfully transformed the UML model of this EPSS to an OPCN model using the approach presented in section 3. The resulting model was simulated using Design/CPN and an occurence graph was produced for verification of behavioural properties such as boundedness, liveness, deadlockfreeness and fairness etc. A partial report generated by Design/CPN is shown in table 3. As part of our future work we intend to examine the scalability aspects of the EPSS architecture using the OCPN model.

Table 3. Desirable Properties of EPSS

Occurrence Graph Nodes: 139 Arcs: 141 Secs: 0 Status: Full
Scc Graph Nodes: 1 Arcs: 0 Secs: 0
Home Properties Home Markings: All
Liveness Properties Dead Markings: None

5 Conclusion

We have presented an algorithm to transform the UML model of a software system to an OCPN model which consists of a class net corresponding to each class in the UML model. Different class nets are connected through communication channels that are used for message passing between objects. A class net creates an object in the form of a token when required. This token is consumed/deleted by the class net when the object is no longer needed. The algorithm provides a methodology for software engineering of complex systems in which the industry-standard practice of design and analysis using UML models is enhanced by verification of behavioural properties at design stage. The graphical nature of Petri Net models means a software developer without a deep mathematical understanding of Petri nets can use this technique for dynamic analysis of UML models.

6 Acknowledgement

The authors wish to acknowledge the many helpful discussions with Dr. Ridha Khedri of the Computing and Software Department at McMaster University, which aided greatly in clarifying the issues presented in this paper.

References

[1] L. Baresi. Some preliminary hints on formalizing UML with object Petri nets. In *Integrated Design and Process Technology, IDPT-2002*, June 2002.

[2] A. Bokhari and S. Poehlman. Design of an agile performance support system. In *Proceedings of the 42nd International Performance Improvement Conference and Exposition, Tempa, Florida*. International Society for Performance Improvement (ISPI), April 2004.

[3] A. Bokhari and S. Poehlman. Formalization of uml statecharts: Aproaches for handling composite states. Technical Report CAS-05-07-SP, McMaster University, Computing and Software Department, 2005.

[4] A. U. Denmark. Design/cpn and cpn tools. http://www.daimi.au.dk/CPnets/.

[5] Z. Hu and S. M. Shatz. Mapping UML diagrams to a Petri net notation for system simulation. In *Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE), pp 213-219*, Banff, Canada, June 2004.

[6] K. Jensen. *Coloured Petri Nets Basic Concepts, Analysis Methods and Practical Use*, volume 1. Springer Verlag, Aarhus University, Denamrk, 2 edition, 1997.

[7] D. Latella, I. Majzik2, and M. Massink1. Automatic verifcation of a behavioural subset of uml statechart diagrams using the spin model-checker1. *Formal Aspects of Computing*, 11:pp 637–664, 1999.

[8] J. Lilius and I. P. Paltor. Formalizing UML state machines for model checking. In *Robert France and Burnhard Rumpe (Eds.):UML'99, Lecture Notes in Computer Science (LNCS 1723), pp 430-444*, Springer Verlag Berlin Heidelberg, 1999.

[9] C. Maier and D. Moldt. Object coloured Petri nets - a formal technique for object oriented modelling. In *Agha et al.(Eds.):Concurrent OOP and PN, LNCS 2001, pp 406-427*, Berlin Heidelberg, 2001. Springer Verlag 2001.

[10] S. Philippi. Seamless object-oriented software development on a formal base. In *Proceedings of the Workshop on Software Engineering and Petri Nets, 21st International Conference on Application and Theory of Petri Nets*, June 2000.

[11] O. Rysavy. A survey on approaches to formal representation of UML. Technical report, Brno University of Technology, Czech Republic, 2003.

[12] J. A. Saldana and S. M. Shatz. UML diagrams to object Petri net models: An approach for modelling and analysis. In *Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE), pp 102-110*, July 2000.

[13] J. A. Saldana and S. M. Shatz. Formalization of object behavior and interactions from UML models. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, 11 No. 6:643–673, 2001.

[14] Y. Zhao and Y. Fan. Towards formal verifi cation of UML diagrams based on graph transformation. In *Proceedings of the IEEE International Conference on E-Commerce Technology for Dynamic E-Business (CEC-East'04)*, 2004.

Translating UML Diagrams Into Maude Formal Specifications: A Systematic Approach

Farid Mokhati[(1)], Mourad Badri[(2)] & Patrice Gagnon[(2)]

[(1)]Department of Computer Science , University of Oum-El-Bouaghi, Algeria
[(2)]Department of Mathematics and Computer Science, University of Quebec at Trois-Rivieres, Canada
E-mail: Mokhati@yahoo.fr, Mourad.Badri@uqtr.ca, Patrice.Gagnon@uqtr.ca

Abstract

In this paper, we present a formal framework supporting the translation of UML diagrams into a formal specification based on the Maude language. Our approach considers both static and dynamic aspects of object-oriented systems. We consider, in particular, UML class, state and collaboration diagrams. The formal and object-oriented language Maude, based on rewriting logic, supports formal specification and programming of concurrent systems. The major motivations of this work are: (1) binding together the UML notation and the Maude language, and exploiting their respective advantages, (2) further the precision of the description of object-oriented systems and, finally, (3) provide a healthy description preserving the coherence and supporting their verification and validation process. The generated Maude specifications are validated by simulation. The approach is illustrated using a concrete example.

Key Words: *Object-Oriented Systems, UML, Static Aspects, Dynamic Aspects, Formal Specification, Maude, Translation, Verification and Validation.*

1. INTRODUCTION

UML (*Unified Modeling Language*) is a language for specifying, visualizing and constructing the artifacts of software systems [12]. UML allows describing various aspects of complex systems. However, UML models can present some ambiguities and inconsistencies and be the source of diverse problems as mentioned in [13, 27, 3, 28, 7]. It suffers from a lack of formal semantics [2, 23, 28]. This weakness can lead to inconsistencies within the developed models. Using formal methods in the development of complex systems presents notable advantages [13, 7, 28, 18], like a more simple design without ambiguities, as well as a more complete documentation [4, 28].

In this paper, we present a formal framework supporting the translation of UML diagrams into a formal specification based on the Maude language. We are interested in both static and dynamic aspects of object-oriented systems. We consider, in particular, UML class, state and collaboration diagrams. Our approach is based on the coupling principle of semi formal notations (UML) and formal notations (Maude). The developed translation process preserves the semantics of UML diagrams. The adopted approach is structured in three major steps. The first step consists of describing both static and dynamic aspects of an object-oriented system using UML class diagram (static aspect), state (individual behavior) and collaboration diagrams (collective behavior). The second step corresponds to an inter-diagrams validation process. The third step consists on automatically generating a Maude description from the considered UML diagrams. The ultimate goal of our work is to support the formal verification of UML models based on Maude's model checking techniques. In this paper, we focus on the translation process of UML models into Maude specifications. Our approach is generic.

The remainder of the paper is organized as follows. In section 2, we give a brief overview of related work. Section 3 briefly presents the UML diagrams we use. Section 4 gives an overview of rewriting logic and Maude. We present, in section 5, the translation process we propose. Section 6 illustrates the translation process using a concrete case study. Finally, we give a conclusion and future work directions in section 7.

2. RELATED WORK

Funes et al. [10] have formalized UML class diagram using the formal specification language RSL (Raise Specification Language). Using the same language, Meng et al. [25] presented a formalization for state diagrams. Furthermore, Favre [9] has proposed a translation process, for class diagrams and packages in the NEREUS language, based on the MDA (Model Driven Architecture) methodology. The obtained NEREUS specification is then transformed into an object-oriented code (Eiffel language). Joao et al. [1] proposed a generation process to obtain Object-Z specifications from UML collaboration diagrams. In the same context, other proposals [24, 14] have considered other UML diagrams. On the other hand, Paige and Brooke presented in [21] a pragmatic approach integrating the object-oriented methodology BON (an alternative to UML) and the Object-Z language. Their approach was implemented using the BON-

CASE tool [22]. This tool supports formal specifications through pre-conditions, post-conditions and class invariants, whether for reasoning or for formal analysis [22]. The majority of these papers have focused on translating to a formal specification only one aspect of object-oriented systems, whether static or dynamic.

In the same context, other approaches have used jointly class diagrams to describe static aspects of object-oriented systems and state diagrams to describe their dynamic aspects (individual behavior of objects). Among those approaches, we can cite the U2B tool [26]. U2B is a script file for *Rational Rose* allowing converting to the B language the current Rational Rose model composed of class and state diagrams. However, the collective behavior is not considered. Furthermore, Ledang et al. have developed the ArgoUML+B tool [20]. Of course, those approaches have considerably forwarded the domain by integrating the static and dynamic aspects of object-oriented systems and their translation into formal specifications. However, the dynamic aspects presented in those papers only consider the individual behavior of objects.

We present, in this paper, a more global approach allowing generating a Maude formal specification integrating both static and dynamic aspects (individual and collective) of object-oriented systems. We use UML class diagram to represent static aspects of an object-oriented system, and state and collaboration diagrams (respectively individual and collective behavior) to represent its dynamic aspects. The formal and object-oriented language Maude supports formal specification and programming of concurrent systems. Aside from the semantics of concurrency (intra and inter object) that it offers, Maude is a multi paradigm language. Furthermore, the Maude language is supported by a tool, which allows the validation of the generated formal descriptions.

3. UML DIAGRAMS

UML class diagrams express the static structure of a system, in terms of classes and relationships between classes. Classes are essentially organized through aggregation, inheritance or association relationships [19]. UML state diagrams [19] describe, using finite state machines, the life cycle of objects. Even while there exist different types of events defined by UML, we focus only on the events of the "Call" type. UML Collaboration diagrams [11, 19] describe how a set of objects collaborate to accomplish a specific task. They emphasize the dynamic interactions between those objects (message exchanges) as well as their synchronization.

4. REWRITING LOGIC AND MAUDE

Rewriting logic, having a sound and complete semantic, was introduced by Meseguer [17]. It allows describing concurrent systems [18, 15, 8, 6]. This logic unifies all the formal models that express concurrency [16]. The rewriting rules are of the form $R : [t] \rightarrow [t']if C$, which indicates that, according to rule R, term t is transformed into t' if a certain condition C is verified.

Maude is a specification and programming language based on rewriting logic [17, 5, 6, 15]. Three types of modules are defined in Maude. *Functional* modules allow defining data types and their functions. The *system* modules allow defining the dynamic behavior of a system. This type of module augments the functional modules by introducing rewriting rules. Finally, the *object-oriented* modules, which can be reduced to system modules, offer a more appropriate syntax to describe the basic entities of the object paradigm.

5. TRANSLATION PROCESS

The adopted translation process consists of systematically deriving a Maude formal specification from an analysis of the UML class, state and collaboration diagrams. Figure 1 shows the steps of the translation process.

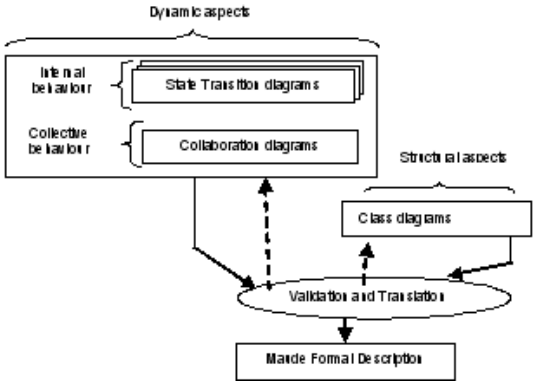


Figure 1. Methodology of the approach

The diagrams go through a first verification step to make sure, for example, that each messages sent to a destination object in the collaboration diagram exists in the state diagram and that it is accessible.

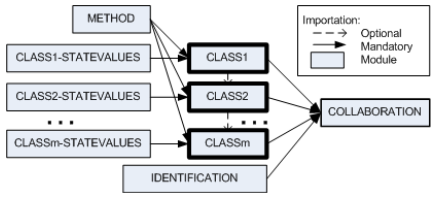


Figure 2. Generated modules

During the translation process of the considered UML diagrams, several Maude modules are generated. Figure 2 shows those modules. Please note that modules in bold are object-oriented modules. The functional module *METHOD* (figure 3) contains all the types used to describe a method. Types *Parameter* and *ParameterList* are generic. They describe the type of parameters a method uses. Furthermore,

ResultType and *Void* describe the type of the result returned by the method. *ResultType* is generic, and *Void* is a particular case of *ResultType*. The operation (_,_) is a constructor for the parameter list of a function.

```
fmod METHOD is
sorts  ParamaterList ResultType Paramater \void .
subsort Paramater < ParamaterList .
subsort \void < ResultType .
op EmptyParamaterList :-> ParamaterList .
op _,_ : Paramater ParamaterList -> ParamaterList .
endfm
```

Figure 3. The *METHOD* module

We associate to each state diagram a functional module for which the name is the concatenation of the class' name and the string *'STATEVALUES'*. The functional module *IDENTIFICATION* is generated to describe the identification mechanism of the objects of the collaboration diagram. For each class of the class diagram, we associate an object-oriented module bearing the same name as the class, while adopting a generic form for the classes (figure 4). In the case where one of such a class is in relation to other classes in the class diagram, the module associated to it must import all the other modules associated to those classes. The class is declared in a module with a state attribute called *State* and for which its type is declared in the corresponding functional module. In the case of an aggregation class, an identification list of all the aggregated classes must also be present.

```
class ClassName | State : ClassNameStateValues [, ComposantList] .
```

Figure 4. Generic form adopted for the classes

In this module (see figure 5), we define the methods of a class by using the specified form.

```
op FunctionName : ParamaterList -> ResultType .
```

Figure 5. Form adopted for the methods

The object-oriented module *COLLABORATION* is the principal one generated by our approach. It imports all the other modules. In this module, we extend all the other object-oriented modules by describing the behavior of the different objects implicated in the collaboration diagram with the help of rewriting rules. Each message exchanged between two objects of the collaboration diagram is translated in the form of a *ComingMsg* shown in figure 6.

```
msg ComingMsg : ResultType Receiver -> Msg .
```

Figure 6. Form adopted for the messages

We precise two things in this message. On the first hand, we identify the destination object (Receiver) and, on the other hand, the result type of the operation to be executed. In fact, each sending of a message in the collaboration diagram corresponds to a Call Event, launching a transition in the state diagram of the destination object. This transition is described in this module whether by an unconditional rewriting rule in the case where the sending of the message is not linked to a condition, or by a conditional rewriting rule otherwise. To implement the concept of Synchronization Point [11, 19] of the messages sent within a collaboration diagram, we introduce a new message called *IsAccomplished* (see figure 7). The rewriting rule that implements transition corresponding to the sending of a message on which depends other messages must generate a number of *IsAccomplished* messages equal to the number of messages to be sent. This message is also used in the case where the sending of an asynchronous message (such as the case of message *Start* in figure 9) depends on the sending of another message (the case of message *Initialize* of figure 9).

```
msg IsAccomplished : ResultType Receiver -> Msg .
```

Figure 7. Form of the synchronization message

The *IsAccomplished* message does not represent a message sent between two objects in the collaboration diagram. It must be interpreted as an indication that the sending of the message is terminated.

6. CASE STUDY: THE ELEVATOR

6.1. Presentation

This section illustrates the application of our approach on a concrete example taken from [19]. This example was simplified for our study. It consists of part of an elevator system. Figure 8 shows the class diagram of that system. The collaboration diagram of figure 9 describes one of the functions implemented by the system, namely the procedure done by a user at a given moment to use the elevator after it was started properly. Figure 10 shows respectively the state diagrams for classes *Door*, *SignalLight*, *Cabin* and *Elevator*.

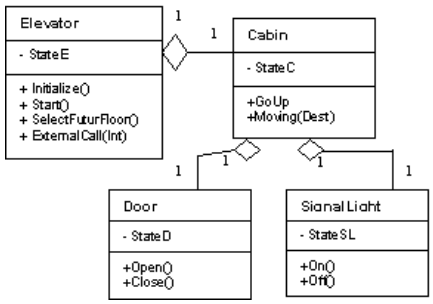


Figure 8. Class diagram of the system

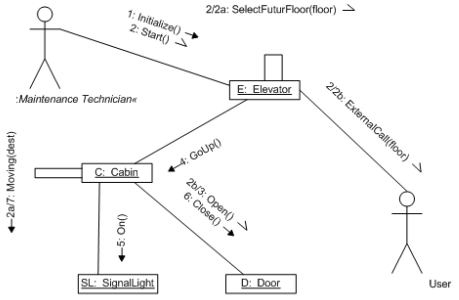


Figure 9. Representation of the behaviour of an elevator cabin

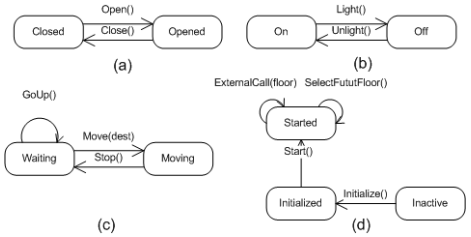


Figure 10. State Transition diagrams for classes Door (a), SignalLight (b), Cabin (c) and Elevator (d)

6.2. Application of the translation process

By applying the translation process we proposed, we obtained the modules described in what follows. We have four functional modules: *ELEVATOR-STATEVALUES*, *CABIN-STATEVALUES*, *SIGNALLIGHT-STATEVALUES* and *DOOR-STATEVALUES*.

Those last modules contain respectively the states of the different classes: *Elevator*, *Cabin*, *SignalLight* and *Door*. For reasons of space limitation, we only give the code for one of them, namely *ELEVATOR-STATEVALUES* (see figure 11).

```
(fmod ELEVATOR-STATEVALUES is
sort ElevatorStateValues .
ops Inactive Initialized Started :-> ElevatorStateValues .
endfm)
```

Figure 11. Module *ELEVATOR-STATEVALUES*

A module *IDENTIFICATION* (see figure 12) imports the predefined *CONFIGURATION* module. This module contains the definition of types *Eoid*, *Coid*, *Doid*, *SLoid* which describe the identification mechanism of the objects *E*, *C*, *P* and *SL*, instance of classes *Elevator*, *Cabin*, *Door* and *SignalLight* respectively.

We have four object-oriented modules: *ELEVATOR*, *CABIN*, *SIGNALLIGHT* and *DOOR*. In each module, we define the class with a *State* attribute describing the current state

```
(fmod IDENTIFICATION is
including CONFIGURATION .
sort Eoid Coid Doid SLoid Receiver .
subsort Eoid Coid Doid SLoid < Oid .
subsort Receiver < Eoid Coid Doid SLoid .
endfm)
```

Figure 12. Module *IDENTIFICATION*

of the object, and a list of composing objects in case of aggregate classes, as well as the different methods of the class. We give the code for only one of those modules, namely *CABIN* (see figure 13). An object-oriented module *COLLABORATION* constitutes the principal module generated by our approach. It imports modules *IDENTIFICATION*, *ELEVATOR*, *CABIN*, *SIGNALLIGHT* and *DOOR*. For space reasons, we give only part of that module (figure 14).

```
(omod CABIN is
protecting DOOR .
protecting SIGNALLIGHT .
protecting CABIN-STATE-VALUES .
sort Cabin Target .
subsort Cabin < Oid .
subsort Target < Parameter .
class Cabin | State : CabinStateValues, idDoor : Oid, idSL : Oid .
ops UP DOWN : -> Target .
op GoUp : ParameterList -> Void .
op Stop : ParameterList -> Void .
op isMoving : Target -> Void .
endom)
```

Figure 13. Module *CABIN*

Rewriting rule *'E1'* of figure 14 describes the reception of message *Initialize* by object *E*. After its execution, the rule generates a message *IsAccomplished* that will be used to allow asynchronous message *Start* to be sent. The execution of the second rule, namely *'E2'*, needs, aside from the *IsAccomplished* message generated by the first rule, the arrival of message *Start*. We then can conclude that such a rule generates two *IsAccomplished* messages, to allow two other messages, namely *SelectFuturFloor* and *ExternalCall*, to be sent (see figure 9).

```
(omod COLLABORATION is
protecting IDENTIFICATION .
extending ELEVATOR .
extending CABIN .
extending DOOR .
extending SIGNALLIGHT .
msg ComingMsg : ResultType Receiver -> Msg .
msg IsAccomplished : ResultType Receiver -> Msg .

var E : Eoid . var C : Coid . var D : Doid . var SL : SLoid .

*****************Comportement de l'Ascenseur*****************
rl [E1]: ComingMsg(Initialize(EmptyParametersList), E)
        < E : Elevator | State : Inactive, idCabin : C >
      =>
        < E : Elevator | State : Initialized, idCabin : C >
        IsAccomplished(Initialize(EmptyParametersList), E) .

rl [E2]: IsAccomplished(Initialise(EmptyParametersList), E)
        ComingMsg(Start(EmptyParametersList) E )
        < E : Elevator | State : Initialised, idCabin : C >
      =>
        < E : Elevator | State : Started, idCabin : C >
        IsAccomplished(Start(EmptyParametersList), E)
        IsAccomplished(Start(EmptyParametersList), E) .
...
endom)
```

Figure 14. Module *COLLABORATION*

6.3. Validation of the generated description

Rewriting logic is very flexible when it comes to simulation of a specification. It notably offers the possibility of selecting the initial configuration. By using the description of the system, we can validate part of the system without compromising the rest. To that end, we study two essential cases: the case where the elevator receives a message to start, after its initialization by a maintenance technician, and the case where this same elevator receives an external call by user while being in the started state.

For the first case, we propose the following initial configuration (figure 15):

```
ComingMsg(Initialise(EmptyParamaterList), E)
ComingMsg(Start(EmptyParamaterList), E)
< E : Elevator | State : Inactive , IdCabin : C > .
```

Figure 15. Initial configuration

We define an initial configuration comprising an object *E* instantiated from class *Elevator* in its initial state (*Inactive*) and two messages. The first is to initialize the elevator, and the second to start it. The infinite rewriting (with no limitation on the number of steps) returns the result shown in figure 16.

```
IsAccomplished(Start(EmptyParamaterList), E)
IsAccomplished(Initialize(EmptyParamaterList), E)
< E : Elevator | State : Started, IdCabin : C > .
```

Figure 16. Result of the unlimited rewriting of the initial configuration of figure 15

After the reception of an initialization message, object *E* goes from its *Inactive* state to *Initialized* (figure 10.d). In such a state, this same object receives a message to start and transits to its *Started* state. The resulting configuration in figure 15 contains two messages that are generated to allow messages *SelectFuturFloor* and *ExternalCall* to be executed (see figure 9). For the second case, we propose the initial configuration shown in figure 17. This configuration is in fact an extension of the one in figure 15. It contains the two additional messages *ExternalCall* and *SelectFuturFloor*.

```
ComingMsg(Initialize(EmptyParamaterList), E)
ComingMsg(Start(EmptyParamaterList), E)
< E : Elevator | State : Inactive , IdCabin : C >
ComingMsg(ExternalCall(4), E)
ComingMsg(SelectFuturFloor(EmptyParamaterList), E)
```

Figure 17. Initial configuration

After messages *Initialize* and *Start* are consumed, the elevator returns to its *Started* state. Two *IsAccmplished* messages are then generated to unlock the execution of messages *Moving* and *Open* (see figure 9). The result is shown in figure 18.

```
< E : Elevator | State : Started, IdCabin : C >
IsAccomplished(ExternalCall(4), E)
IsAccomplished(SelectFuturFloor(EmptyParamaterList), E) .
```

Figure 18. Result of the unlimited rewriting of the initial configuration of figure 17

6.4. Implementation

We implemented the collaboration diagram of figure 9 with our approach. We implemented an initial configuration that describes a coherent sequence of events in a real elevator system (see figure 9). We made two sets of simulation, the first being limited to 5 steps, the second unlimited. In this configuration, all objects are in their respective initial state. The results of these rewritings are shown in figure 19.

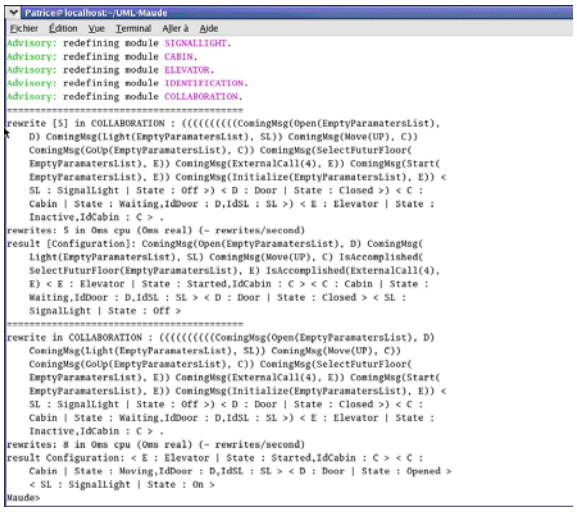


Figure 19. Results of the limited initial configuration, then unlimited

The first result describes an intermediate configuration of the system (after 5 rewriting steps). The elevator is then in its *Started* state and the other objects are still in the initial one. This expresses that the rewriting rules corresponding to changes in those states have not yet been executed. This intermediate configuration also shows that other messages not yet consumed, belonging to this same configuration. The second result represents a final configuration of the system after an unlimited rewriting of the initial configuration described previously. This final configuration represents a coherent state for such a system.

7. CONCLUSIONS AND FUTURE WORK

The translation of UML diagrams in formal languages was the subject of numerous works. Several tools supporting the translation process have been developed. However,

the majority of those approaches did not consider the collective behavior of objects. In this paper, we proposed a generic approach that allows translating static aspects (described by the UML class diagram) and dynamic aspects (described by UML collaboration diagrams along with UML state diagrams) of object-oriented systems into a Maude formal specification. Such a specification integrates both static and dynamic aspects of the described system. The Maude language is supported by a tool, which allowed us to validate the generated code by simulation. As future work, we plan to extend our approach to verify the generated descriptions by using Maude's incorporated model checker. Maude proposes an integrated model checking engine in its environment. This verification tool uses linear propositional temporal logic (LTL) to specify the properties to be tested within the model.

References

[1] J. Araujo and A. Moreira. Specyfying the behavior of UML collaborations using object-z. 2000.

[2] E. Astesiano. UML as heterogeneous multiview notation. strategie for a formal foundation.

[3] M. Barnett, R. DeLine, M. Fahndrich, K. Rustan, M. Leino, and W. Schulte. Verification of object-oriented programs with invariants, 2003.

[4] J. Bowen. Formal specification and documentation using z: A case study approach. 2003.

[5] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and José F. Quesada. Maude: Specification and programming in rewriting logic. *Theoretical Computer Science*, 2001.

[6] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Mesenguer, and Carolyn Talcott. *Maude Manual (Version 2.1.1)*, April 2005.

[7] María del Mar Gallardo, Pedro Merino, and Ernesto Pimentel. Debugging UML designs with model checking. *Journal of Object Technology*, Vol. 1(No. 2):pages 101–117, 2002.

[8] S. Eker, J. Meseguer, and A. Sridharanarayanan. The maude ltl model checker, 2002.

[9] L. Favre. Foundations for mda-based forward engineering. *Journal of Object Technology*, Vol. 4(No. 1):pages 129–153, 2005.

[10] A. Funes and C. George. Formal foundations in rsl for uml class diagrams. technical report 253. Technical report, UNU/IIST, May 2002.

[11] J. Rumbaugh et I. Jacobson G. Booch. The Unified Modeling Language User Guide. 1998.

[12] Object Modeling Group. *Unified Modeling Language Specification, Version 1.4*. September 2001.

[13] A. Moreira J.M. Bruel, J. Lilius and B. Robert. Defining precise semantics for uml. In *ECOOP'2000 Workshop Reader*, number 1964. Springer-Verlag, November 2000.

[14] Ian MacColl and David A. Carrington. Specifying interactive systems in object-z and csp. In Keijiro Araki, Andy Galloway, and Kenji Taguchi, editors, *IFM*, pages 335–352. Springer, 1999.

[15] Theodore McCombs. Maude 2.0 primer (version 1.0), August 2003.

[16] José Meseguer. Rewriting as a unified model of concurrency. *SIGPLAN OOPS Mess.*, 2(2):86–88, 1991.

[17] José Meseguer. A logical theory of concurrent objects and its realization in the maude language. pages 314–390, 1993.

[18] Jos'e Meseguer. Software specification and verification in rewriting logic. 2003.

[19] P.A. Muller and N. Gaertner. *Modélisation objet avec UML*. Deuxième edition edition, 2000.

[20] ArgoUML Tigris Organisation. *ArgoUML User Manual*, 2002.

[21] Richard F. Paige and Phillip J. Brooke. Integrating bon and object-z. *Journal of Object Technology*, 3(3):121–141, 2004.

[22] Richard F. Paige, Liliya Kaminskaya, Jonathan S. Ostroff, and Jason Lancaric. Bon-case: An extensible case tool for formal specification and reasoning. *Journal of Object Technology*, 1(3):77–96, 2002.

[23] G. Reggio and R. Wieringa. Thirty one problems in the semantics of uml 1.3 dynamics. In *Conference on Object Oriented programming, Systems, Languages and Applications (OOPSLA'99) - Workshop*.

[24] Dong J. S. Formal specification and design techniques, 2000. Lecture Notes.

[25] Z. Naixiao S. Meng and B. K. Aichernig. The formal foundations in rsl for uml statechart diagrams. technical report 299. Technical report.

[26] C. Snook and M. Butler. *UML-B Specification for Proven Embedded Systems Design*. 2004.

[27] H. Malgouyres et G. Motet S.V. Jean-Pierre. Identification de règles de cohérence d'uml 2.0, journée see.

[28] Toufik Taibi and David Ngo Chek Ling. Formal specification of design patterns - a balanced approach. *Journal of Object Technology*, 2(4):127–140, 2003.

Specifying Consistency Constraints for Modelling Languages

Lijun Shan
Department of Computer Science
National University of Defence Technology
Changsha, 410073, China
Email: lijunshancn@yahoo.com

Hong Zhu
Department of Computing
Oxford Brookes University
Oxford OX33 1HX, UK
Email: hzhu@brookes.ac.uk

Abstract. As graphic modelling languages play an increasingly important role in software development, completeness and consistency have become essential quality attributes of software models. This paper presents a framework for the formal definition of the abstract syntax and type systems of modelling languages that facilitates formal specification and automatic checking of consistency and completeness constraints on graphic models. The approach is illustrated by the specification of CAMLE modelling language and its consistency and completeness constraints. An empirical study of the effectiveness of the framework is reported, which shows that about 85% of errors generated by mutation operators can be detected by automatic consistency checking.

1. Introduction

Modelling languages are playing an increasingly important role in software development as model-driven software development methodology is gaining wide acceptance. Typical modelling languages include UML for object oriented software development [1], Yourdon notation and SSADM for structured analysis and design, CAMLE modelling language [2] for the emerging agent-oriented software development, etc. Well-defined visual notations for modelling software systems balance well between readability and preciseness due to their semi-formal nature. As a means of separation of concerns, the multiple-views principle has been widely adopted in modern modelling languages. By representing different aspects of a system in different views and/or at different levels of abstractions, it provides a powerful vehicle for dealing with the complexity of information systems. However, as pointed out in [3, 4], maintaining consistency between views and completeness of the models is crucial, but difficult. It is highly desirable to automatically check models' consistency and completeness; yet, graphic models must be well-formed to be processed and transformed. Unfortunately, these tasks are by no means trivial. For example, UML [1] does not systematically and explicitly define consistency and completeness constraints, though OCL [5] provides a language facility for specifying constraints on the instances of models. Many research efforts addressing the consistency problems of UML have been reported, c.f. [6, 7, 8, 9, 10]. Although graphic modelling languages such as UML are widely accepted and new modelling languages are being developed, how to define the syntax and semantics of graphic modelling languages is still an open problem.

In [11], we proposed a framework for the definition of the abstract syntax and type systems of graphic modelling languages so that well-formedness, consistency and completeness of graphic models can be formally specified and automatically checked. To demonstrate the effectiveness of the approach, this paper exemplifies the framework with the CAMLE language by defining its abstract syntax and type system and specifying its consistency and completeness constraints. An experiment with CAMLE consistency checking tool is also reported in this paper. It shows that consistency constraints specified and implemented in our approach can achieve an error-detecting rate around 85%.

The remainder of the paper is organised as follows. Section 2 reviews the framework proposed in [11]. Section 3 presents the definition of CAMLE language. Section 4 reports the experiment with the effectiveness of consistency and completeness checking. Section 5 concludes the paper with a discussion of related work and further works.

2. Overview of the framework

2.1 Type systems and well-formedness

Let's first formally define a few basic concepts of modelling languages; see [11] for more details.

A modelling language $\mathcal{ML}$ is called a *multiple-views modelling language*, if it defines a finite set $\boldsymbol{T}\neq\emptyset$ of *types of diagrams*. Each type $T\in\boldsymbol{T}$ of diagrams provides a set of graphical notations to represent a view of the system. A model M in $\mathcal{ML}$ consists of a set $\boldsymbol{D}\neq\emptyset$ of diagrams. Each diagram $D\in\boldsymbol{D}$ has one and only one type T_D in $\boldsymbol{T}$. We write $Type(D)$ to denote the type T_D of diagram D. The subset of diagrams of a type T in a model M is called the *T-view of the model M* (or simply the *T-sub-model* or *T-model*), written $M.T$. Formally, $M.T=\{D|D\in M, Type(D)=T\}$.

$\mathcal{ML}$ is said to be *graphically typed*, iff for each type $T\in\boldsymbol{T}$, $\mathcal{ML}$ defines a finite set N_T of node types and a finite set E_T of relation types. For each type $te \in E_T$ of relations, a relation e of type te in a diagram D of type T can only be specified on certain type(s) of nodes or relations in D.

$\mathcal{ML}$ is *annotationally typed*, iff for each type T of diagrams, the following conditions hold.

(a) For each diagram type T, $\mathcal{ML}$ defines a finite number of fields $f_{T,i}$, i=1, …n_T, for the annotations that can be associated to a diagram of type T. For each field $f_{T,i}$ $\mathcal{ML}$ defines a data type $FT_{T,i}$ of the values that can be assigned to field $f_{T,i}$.
(b) For each node or relation type t of diagrams of type T, $\mathcal{ML}$ defines a finite set of fields $f_{t,i}$, i=1, …, n_t, for the annotations that can be associated to the nodes or relations of type t. For each field $f_{t,i}$, $\mathcal{ML}$ defines a data type $d_{t,i}$ of the values that can be assigned to field $f_{t,i}$.

A modelling language $\mathcal{ML}$ is *typed*, iff it is both graphically and annotationally typed.

In a typed $\mathcal{ML}$, a diagram D of type T is *graphically well-formed*, iff each node n is associated to one and only one node type tn, and each relation e of type te on nodes n_1, …, n_k must satisfy the type requirements of te. A diagram D of type T is *annotationally well-formed*, iff the values assigned to the annotation fields of the diagrams, the nodes and the relations of the diagrams are all compatible to the data types. A model M is *well-formed* if all diagrams of M are both graphically and annotationally well-formed.

To define type systems and abstract syntax of modelling languages, a graphical extension of BNF (called GEBNF) was proposed [11], which is summarized in Table 1.

Table 1. GEBNF Notation

Notation	Meaning	Example and explanation
<X>	X is the name of a type of entities	<*Class Diagram*>: the type of entities called class diagrams.
X ::= Y	X is defined as Y	<*Model*> *::=* <*Diagram*>* : a model is defined as consisting of a number of diagrams.
X*	Repetition of X (include null)	<*Diagram*>*: the entity consists of a number *N* of diagrams, where $N \geq 0$.
X+	Repetition of X (exclude null)	<*Diagram*>+: the entity consists of a number *N* of diagrams, where $N \geq 1$.
X | Y	Choice of X and Y	<*Actor node*>|<*Use case node*> means that the entity is either an actor node or a use case node.
X , Y	X and Y	<*Actor node*>, <*Use case node*>: an entity that consists of an actor node and a use case node.
[X]	X is optional	[<*Actor*>]: Actor is optional.
X Y	Order pairs consists of X and Y	<Actor node> <Use case node>: an element that consists of an order pair of an actor node and a use case node.
/X/	An annotation field named X	/*Use case name*/: the annotation field called use case name.
X: Y	The type of X is Y.	/*Use case name*/*: Text* : the type of the annotation use case name is text.
(X)	Parenthesis	It is used to change the preferences of the expression.
'abc'	The literal of a string	'*extends*': the literal value of the string 'extends'.
Text[!*F*]	Predefined type *Text* with syntax specified by F in BNF	*'Text'*: a text in any format; '*Text ! <object name> ':' <class name>* ' : the text that consists of an object name and a class name separated by a colon.

2.2 Consistency and completeness constraints

Generally speaking, a *consistency constraint C* is a predicate defined on models such that $C(M) = true$ means that the model is consistent with respect to the constraint; otherwise, the model is inconsistent and hence, not sound. Informally, a consistency constraint restricts how models should be constructed so that certain types of conflictions in the information specified by the model can be prevented and detected. A *completeness constraint* restricts the construction of the models so that certain types of errors due to the lack of information can be prevented and detected. A violation of a consistency constraint implies that there is an error in the model due to confliction between different parts of the model. Therefore, no system can satisfy the specification of the model. In contrast, a violation of a completeness constraint implies that a certain piece of information is missing. Therefore, there will be a system that the users do not want satisfying the specification.

There are several kinds of constraints that can be defined on modelling languages.

A. Intra-diagram vs *Inter-diagram constraints.* A constraint *C* is *intra-diagram*, if it is defined on a diagram of a specific type *T*. It is *inter-diagram*, if it is defined on two or more diagrams.

B. Inter-model vs *Intra-model constraints.* A constraint *C* is *inter-model*, if it is defined on diagrams of more than one type; otherwise, it is *intra-model*.

For hierarchical modelling languages, constraints can also be classified into vertical and horizontal constraints, and global and local constraints.

C. Vertical vs. Horizontal constraints. A constraint *C* is *horizontal* if it is defined between diagrams of the same abstraction level. A *vertical* constraint *C* is defined between diagrams that have refinement relationships between them.

E. Local vs Global constraints. A constraint *C* is *global* on a particular type of diagrams, if it is defined on the whole set of diagrams of the type. Otherwise, it is *local constraint*.

Given a definition in GEBNF, a first order language can be derived as follows for the formal definition of consistency and completeness constraints. Let φ and ρ be n-ary operator and relation, respectively.

- *Expressions* are formed by finite applications of the following constructions.
  - *Variables* and *constants* are expressions;
  - $\varphi(e_1, e_2, \ldots, e_n)$ is an expression, if $e_1, e_2, \ldots, e_n$ are;
  - $e.f$ is an expression, if e is and f is a field;
  - $e.t$ is an expression, whose value is the set of the elements of type t in e, if e is an expression and t is a type;
  - *Type(e)* is an expression, if e is. *Type(e)* is e's type.
- *Statements* are formed by finite application of the following constructions.
  - $\rho(e_1, e_2, \ldots, e_n)$ is a statement, if $e_1, e_2, \ldots, e_n$ are expressions; in particular, $e_1 = e_2$, $e_1 \in e_2$ are statements.
  - $\neg\rho$, $\rho_1 \Rightarrow \rho_2$, $\rho_1 \Leftrightarrow \rho_2$, $\rho_1 \wedge \rho_2$, and $\rho_1 \vee \rho_2$ are statements, if ρ, ρ_1 and ρ_2 are statement;
  - $\forall X \in E.S$ and $\exists X \in E.S$ are statements, if X is a free variable in statement S.

3. Definition of CAMLE Language

As shown in the following GEBNF formula, a CAMLE model consists of three sub-models.

<CAMLE model>::=

<Caste model>, <Collaboration model>, <Behaviour model>

The following subsections present the abstract syntax of the sub-models and some examples of constraints. The definitions of the syntax of various text types are omitted for the sake of space.

3.1 Caste model

From agent-oriented view of information systems, an organization consists of a collection of agents. The agents stand in certain relationships by being a member of certain groups and playing certain roles, i.e. in certain castes. The caste model describes the castes in the system and the structural relationships between them. This organizational structure is captured in a caste diagram. Fig. 1 is an example.

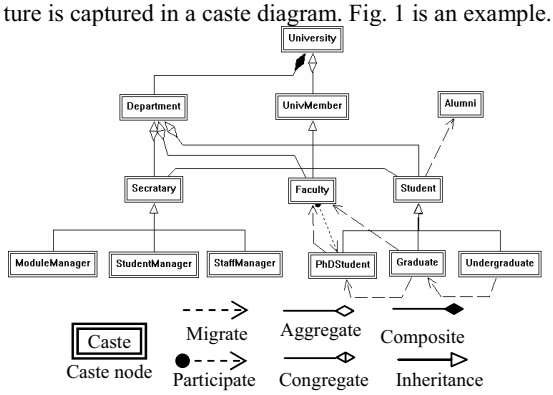


Fig 1. Caste diagram: Example and notation

In GEBNF, caste model is defined as follows.

<Caste model>::= <Caste diagram>
<Caste diagram>::= /Title/:Text ! 'main', <Caste node>+,
<Inheritance relation>*, <Migration relation>*,
<Participation relation>*, <Aggregation relation>*,
<Congregation relation>*, <Composition relation>*
<Caste node>::= /Name/: <Caste Name>
<Inheritance relation>::= <Caste node> <Caste node>
<Migration relation>::= <Caste node> <Caste node>
<Participation relation>::= <Caste node> <Caste node>
<Aggregation relation>::= <Caste node> <Caste node>
<Congregation relation>::= <Caste node> <Caste node>
<Composition relation>::= <Caste node> <Caste node>

The following is an example of intra-diagram consistency constraints on caste diagram.

In a caste diagram, each node has a unique name, i.e.
$\forall D \in M.$<*Caste diagram*>$.\forall X, Y \in D.$<*Caste node*>$.$
$(X./Name/ = Y./Name/ \Rightarrow X=Y)$

3.2 Collaboration models

Collaboration models describe the dynamic structure of a system from the perspective of communication. As shown in Fig 2, there are two types of nodes in a collaboration diagram. An agent node represents a specific agent, while a caste node represents any agent in a caste. An interaction edge from node *A* to *B* indicates that *A*'s visible actions are observed by agent *B*. The actions are annotated on the links.

The definition of collaboration model follows.

<Collaboration model>::= <Collaboration diagram>+
<Collaboration diagram>::=
/Title/: Text ! [(<Agent Name> | <Caste Name>) '*']
('main'| <Scenario description>),
<Agent node>*, <Caste node>)*, <Interaction>*,
[<Environment boundary>]
<Agent node>::= /Name/: Text !<Agent Name>
<Environment boundary>::= <Caste node>*, <Agent node>*
<Interaction>:: = /Action List/: Text ! <Actions>,
(<Agent node>|<Caste node>)(<Agent node>|<Caste node>)

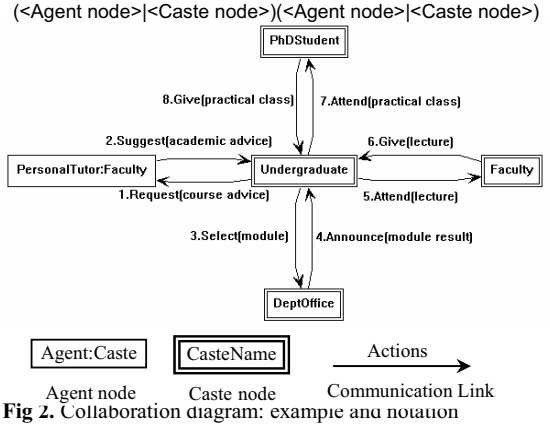


Fig 2. Collaboration diagram: example and notation

An example of intra-diagram consistency constraint on collaboration diagrams is given below.

Each caste or agent node must have a unique name.
$\forall D \in M.$<*Collaboration diagram*>$.(\forall X, Y \in D.($<*Caste node*> $\cup$ <*Agent node*>$).\ (X./Name/ = Y./Name/ \Rightarrow X=Y)\ )$

A caste is a compound caste if it is composed of a number of other castes; otherwise, it is atomic. Each compound caste has a collaboration model and a behaviour model, while each atomic caste only has a behaviour model. Thus, a collaboration model may contain a hierarchy of sub-models on various abstraction levels. When an agent in a system is decomposed into a set of components, a collaboration model is constructed for the compound agent to specify the interactions between its components.

Collaboration model on each abstraction level may contain a general diagram and a set of specific diagrams. A general diagram serves as a declaration of what castes and their instance agents are involved in collaborations, while the specific diagrams define the details of the collaboration protocols in various scenarios. Constraints on the collaboration diagrams at the same abstraction level are horizontal constraints. The following is such an example.

Every agent node in general diagram G must appear in at least one specific diagram in the same collaboration model.
$\forall n \in G.$<*Agent node*>$.(\exists D \in \boldsymbol{S}\ .(\ n \in D.$ <*Agent node*>$)$
where $\boldsymbol{S}$ *is the set of specific diagrams of the same level, CName(n) denotes CasteName part of node n,* $\exists n \in \boldsymbol{S}.$<*X*> *is an abbreviation of* $\exists D \in \boldsymbol{S}.\exists n \in D.$<*X*>*.*

The following is an example of vertical constraints. It is imposed on the models at different levels.

The set of agents and castes in C's environment described in M must be equal to the set of agents and castes in M_C*'s environment description.*
$n \in M_C.$<*Environment boundary*>
$\Leftrightarrow \exists \alpha \in G.$<*Interaction*>$.(n=Begin(\alpha) \wedge C=End(\alpha))$
where G is the general diagram in M.

3.3 Behaviour model

Behaviour model of a system consists of two types of diagrams: behaviour diagrams and scenario diagrams. A behaviour diagram contains a set of behaviour rules to specify the caste's behaviour in certain scenarios. There are six different kinds of arrows that connect different kinds of nodes in behaviour diagrams. A scenario diagram describes a typical situation in the operation of the system. Scenario diagrams are referred to in behaviour diagrams. Fig 3 shows an example of behaviour diagram.

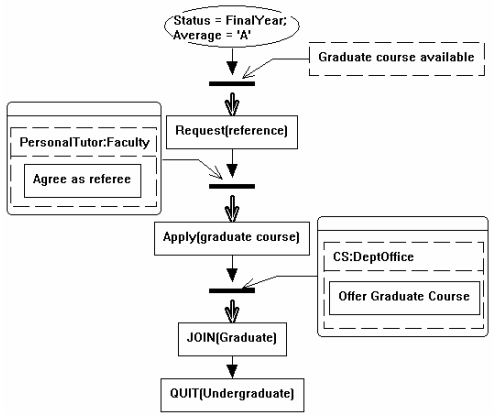


Fig 3. Example of behaviour diagram

The following is the abstract syntax of behaviour models in GEBNF, where details of various types of the links are omitted for the sake of space.

<Behaviour model>::= <Behaviour diagram>+,<Scenario diagram>*
<Behaviour diagram>::=
    /Title/: **Text** ! (<Agent Name> | <Caste Name>),
    <Logic connector>*, <Scenario node>*, <Activity node>+,
    <Action link>+, <Condition node>*,<Transition node>+,
    <Logic link>*, <Temporal relation>*
<Activity node> ::= [</Time stamp/: **Text** ! <Time stamp>],
    [/Repetition/: **Text** ! <Repetition expression>],
    (<Action node> | <State node>)
<Action node> ::= /Action/: **Text** ! <Act>
<State node> ::= /State/: **Text** ! <Predicate>
<Scenario node>::= (/Scenario name/: **Text** ! <Scenario name>)
    | <Scenario diagram>
<Scenario diagram>::= [/Scenario name/: **Text** ! <Scenario name>],
    <Logic connector>*, <Scenario node>*, < Activity node>+,
    <Logic link>*, <Temporal relation>*, <Swim lane>*
<Swim lane> ::=
    /Actor/: **Text** ! <Actor specification>, < Activity node>*

The following is a constraint on behaviour models.

*Scenarios referred to in behaviour diagrams by scenario reference nodes must be defined in scenario diagrams. Let **N** be the set of scenario nodes in the behaviour model, **S** be the set of scenario diagrams.*
$\forall n \in \boldsymbol{N}.\ (n./Scenario\ name/ = s_c \Rightarrow$
$\exists n' \in \boldsymbol{S}.\langle Scenario\ diagram\rangle\ (n'./Scenario\ name/ = s_c))$

3.4 Inter-model Consistency

There are totally 9 inter-model constraints defined for CAMLE and implemented in the automatic checking tool. This subsection gives some examples of constrains between different types of models.

(A) Between collaboration and caste models

Let ***CM*** and ***AM*** be the collaboration and caste model of a system, respectively.

Castes specified in the collaboration model must be defined in the caste model. Formally,
$\forall n \in \boldsymbol{CM}.(\langle Agent\ node\rangle \cup \langle Caste\ node\rangle).$
$\exists n' \in \boldsymbol{AM}.\langle Caste\ node\rangle.(CName(n) = n'./Name/)$

Let x be a caste in the system, M_x be the collaboration model for x. For models M_A and M_B in ***CM***, we say that M_B is an *immediate refinement* of model M_A and write $M_B \lhd M_A$, if B is the component caste of A. The following is an example of global constraints.

*The hierarchical structure of the **CM** must be consistent with the whole-part relations between castes defined in caste diagram. Formally,*
$\forall M_A, M_B \in \boldsymbol{CM}.$
$(M_B \lhd M_A \Leftrightarrow \langle B, A\rangle \in \boldsymbol{AM}.\langle Aggregation\ relation\rangle)$

(B) Between behaviour and caste model

A behaviour diagram defines the behaviour of a caste and the caste must be in the caste model.
$\forall D \in M.\langle Behaviour\ diagram\rangle.\exists n \in M.\langle Caste\ node\rangle.$
$(D./Title/ = n./Name/).$

In a behaviour diagram D_B for caste B, the description of scenarios may refer to the agents in the environment of B. Let $ReferredAgents(D_B)$ be the set of agents referred to in scenarios in D_B.

Every referred agent in a behaviour diagram must have its caste defined in the caste model. Formally,
$\forall D \in M.\langle Behaviour\ diagram\rangle.\forall a \in ReferredAgents(D).$
$\exists n \in \boldsymbol{AM}.\langle Caste\ node\rangle.(CName(a) = n./Name/).$

(C) Between collaboration and behaviour model

An action of a caste C described in a scenario Sc is called a referred action of C in Sc. We write $ReferredActions(C, Sc)$ to denote the set of referred actions of caste C in scenario Sc.

Every referred action in a scenario used in a behaviour diagram must be a visible action of the caste.
$(Type(Sc) = \langle Scenario\ node\rangle) \Rightarrow$
$\forall\ n \in ReferredActions(C, Sc).(n \in VisibleActions(C)).$

Table 2 summarises the total numbers of consistency and completeness constraints on CAMLE.

Table 2. Summary of CAMLE's Constraints

		Horizontal Consistency	Vertical Consistency	
			Local	Global
Intra-model	Intra-diagram	17	–	–
	Inter-diagram	9	4	–
Inter-model		5	4	1

4. Effectiveness of Consistency Check

The well-formedness, consistency and completeness constraints have been implemented as automated checking tools as an integral part of the CAMLE modelling environment [2]. Once invoked, the tool checks the model and reports the diagnostic information about the inconsistency

or incompleteness, if any. The violation of a constraint is reported as an error and a warning. There are totally 21 types of errors and 15 types of warning messages.

A number of case studies have been conducted to model systems including Amalthaea [12, 13], online auction [14], United Nations' Security Council, etc. In our experiences, the automatically checking consistency and completeness was helpful in detecting errors during model construction.

We have also conducted a systematic evaluation of the effectiveness of CAMLE's consistency checker using data mutation analysis to measure the checker's error detecting ability. Data mutation analysis as a software testing method was introduced in [15]. It was designed for testing software systems that have input data of highly complicated structures, such as diagrammatic models. The process of applying data mutation analysis method to our modelling tool consists of the following steps. The first step develops a number of models that passes the consistency checking. These models are called the *seeds*. The second step derives a set of *mutants* from each seed by systematically applying a set of data mutation operators. Each mutant is obtained by one application of one mutation operator on the seed so that it is slightly different from the original model. The mutation operators are design in such a way that it will in most cases make a consistent model inconsistent. Therefore, in most cases, a mutant contains one artificially inserted defect. In our experiment, totally 24 types of mutation operators are designed for CAMLE models. The Amalthaea, the online auction and the United Nations' Security Council models are taken as the seeds, from which totally 7152 mutants are generated. In the third step, the consistency checker is executed to check the consistency of the mutants. According to the output of the checker, the mutants are classified into *dead* or *alive*. A mutant is *dead* if the checker's output on it is different from that on the seed. Otherwise, it is *alive*. In other words, a mutant is dead if and only if the checker detected that it is inconsistent. The effectiveness of the consistency checker can therefore be measured by the percentage of mutants that are killed. A mutation analysis tool was implemented to automatically generate mutants as test cases, check the consistency of all mutants, and calculate statistics. Table 3 shows the results of our experiment. The dead mutant scores in the three suites are around 85%.

Table 3. Results of the Effectiveness Study

Seed	#Mutant	#Dead	#Alive	%Dead
Amalthaea	3065	2692	373	87.83%
Auction	3095	2579	516	83.33%
UNSC	992	821	171	82.76%
Total	7152	6092	1060	85.18%

In general, a mutant may remain alive for two possible reasons. First, it is still consistent and complete even though it is different from the seed. Second, the checker is incapable to detect the inconsistency or incompleteness. In the experiments, the outputs of the checker on each mutant are manually analysed to see if the results are correct. Therefore the effectiveness measurement not only tests the implementation of the consistency checker, but also evaluates the design of the consistency rules.

5. Conclusion

In this paper, we present the definition of the abstract syntax and type system of CAMLE modelling language in the graphically extended BNF. The consistency and completeness constraints for CAMLE are formally defined in the first order logic language derived from the abstract syntax and type system. These constraints are implemented in an automatic checking tool. An evaluation of the consistency constraints in detecting errors demonstrated that the approach is valid and of high error detecting ability.

In recent years, the consistency of multiple-viewed models has been an active research topic. The existing works fall into two approaches: the transformation approach, and the meta-logic/meta-programming approach. The first approach translates diagrammatic models into a formal notation such as B [16], Promela and LTL [17], CSP [18], first-order logic [19], etc. and then applies model checking or automated proof tools; see [20] for a survey. Such methods take advantages of existing formal techniques and tools, and are capable of reasoning deeply about the semantics of the models. The second approach applies formalisms at meta-model level by explicitly defining consistency constraints on the modelling languages. The consistency constraints are expressed with some formalism such as conceptual graphs [21], attributed EBNF [22], OCL [23], description logic [24], graph-grammars [25], etc. Tools have been developed to enable checking models' consistency against the constraints as CASE tools such as OCL Environment [23] and Xlinkit [26, 27], or as plug-ins of existing modelling environments, such as MCC as a plug-in of Poseidon for UML [24] and as a plug-in of Fujaba tool suit [25]. In comparison with the transformation approach, the meta-level approach is more practically applicable. The detected inconsistencies can be directly located in the original models, thus provide more instructive information on how to revise the model. The success of this approach replies on a set of well defined and explicitly specified consistency constraints, which is still an open problem.

Our work belongs to the second approach. It is based on our previous work on modelling tools for structured methods [28]. The main contributions of the work reported in this paper are three-fold. First, a framework of first order language that is capable of specifying constraints on multiple view modelling languages is proposed based on a theory of the structure and type systems of modern modelling languages and the taxonomy of constraints. Second, it demonstrates the practical applicability of the framework by specifying an adequate set of the constraints on a non-trivial modelling language. Third, it provides empirical evidence that the approach is effective and efficient to detect errors in models.

In comparison with existing works, especially those based on OCL, our language is more expressive. It is capa-

ble of specifying constraints across the boundaries between diagrams. Existing OCL-based works e.g. [23] are mainly checking on the well-formedness constraints specified in UML 2.0 documentation, which represent restrictions on the uses of individual elements [1], thus belong to intra-model constraints. They do not address inter-diagram consistency of models. A complete set of consistency and completeness constraints in OCL for UML has not been reported in the literature as far as we know. OCL was originally designed for writing constraints on instances of a system as a part of a model. Meta-level constraints written in OCL tend to be complex and lengthy as shown in [23]. It is also questionable if OCL is capable of specifying inter-diagram and inter-model constraints. The existing tools such as Dresden OCL Toolkit, Kent OCL library, OSLO [29] facilitate the use of OCL rather than the consistency and completeness problems. It is yet to be proved that they are capable of handling complicated constraints.

In [30], an empirical study was reported on uses of real industrial examples to investigate the occurrence frequency of various types of inconsistency in modelling. In this paper, we reported the case study on the effectiveness of consistency checking through artificially produced inconsistent models. These works have different purposes, but the methods are complementary in the research on the consistency problem.

We are currently further investigating how to formally specify UML and define its consistency and completeness constraints.

Acknowledgement

The work reported in this paper is partly funded by The Ministry of Science and Technology of China in the High-Technology R&D Programme (863 Programme) under the grants 2002AA116070 and 2005AA113130. The authors would like to thank Dr. Ian Bayley of Oxford Brookes University for his comments on the earlier versions of the paper and valuable discussions on the related topics.

References

[1] OMG, *Unified Modelling Language: Superstructure*, Version 2.0, formal/05-07-04.

[2] Zhu H and Shan L. Caste-Centric Modelling of Multi-Agent Systems: The CAMLE Modelling Language and Automated Tools. In *Model-driven Software Development*, Beydeda S and Gruhn V (eds), Springer, 2005, pp57-89.

[3] Finklestein A, Gabbay D, Hunter A, Kramer J, Nuseibeh B. Inconsistency handling in multi-perspective specifications, In *IEEE TSE*, Vol. 20, No. 8, 1994, pp569-578.

[4] Hunter A, Nuseibeh B. Managing inconsistent specifications: reasoning, analysis and action, In *ACM TOSEM*, Vol. 7, No. 4, October 1998, pp335-367.

[5] OMG, *OCL 2.0 Specification*, Version 2.0 ptc/2005-06-06

[6] Kuzniarz L, Reggio G, Sourrouille J L & Huzar Z (eds). *Workshop on Consistency Problems in UML-based Software Development* at UML'02.

[7] Kuzniarz L, Huzar Z, Reggio G, Sourrouille J L, Staron M (eds). *Workshop on Consistency Problems in UML-based Software Development II* at UML'03.

[8] Huzar Z, Kuzniarz L, Reggio G, Sourrouille J L (eds). *Third Workshop on Consistency Problems in UML-based Software Development* at UML'04.

[9] Paige R F, Ostroff J S, and Brooke P J. Check the Consistency of Collaboration and Class Diagrams using PVS. In *Proc. of 4th Workshop on Rigorous Object-Oriented Methods*, London, British Computer Society, 2002.

[10] Astesiano E & Reggio G. An Attempt at Analysing the Consistency Problems in the UML from a Classical Algebraic Viewpoint. *Recent Trends in Algebraic Development Techniques, Selected Papers of the 15th Int. Workshop WADT'02*, LNCS, Springer Verlag, 2003.

[11] Zhu H and Shan L. Well-Formedness, Consistency and Completeness of Graphic Models. In *Proceedings of the 9th International Modelling and Simulation* (UKSim'06). pp47-54.

[12] Zhu H. Formal Specification of Evolutionary Software Agents. *Proc. ICFEM'2002*, Springer LNCS 2495, pp249-261.

[13] Shan L and Zhu H. Modelling and specification of scenarios and agent behaviour. In *Proc. of IEEE/WIC conference on Intelligent Agent Technology* (IAT'03), IEEE CS, pp32-38.

[14] Zhu H and Shan L. Agent-Oriented Modelling and Specification of Web Services. In *Proc. of* WORDS'05, pp152-159.

[15] Shan L and Zhu H. Testing Software Modelling Tools Using Data Mutation. Accepted by AST'06 at ICSE 2006.

[16] Marcano R and Levy N. Using B formal specifications for analysis and verification of UML/OCL models. In [6]

[17] Inverardi P, Muccini H, Pelliccione P. Automated check of architectural models consistency using SPIN. In *Proc. of ASE'01,* pp346-356

[18] Küster JM, Stehr J. Towards Explicit Behavioral Consistency Concepts in the UML. In *Proceedings of the 2nd International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools*, 2003

[19] Kaneiwa K and Satoh K. Consistency Checking Algorithms for Restricted UML Class Diagrams. In *Proceedings of Foundations of Information and Knowledge Systems: 4th International Symposium, FoIKS 2006* (FoIKS2006), pp219–239. LNCS 3861, Springer

[20] Sourroulle J L, Caplat G., A Pragmatic View about Consistency Checking of UML Models. In [7]

[21] Sunetnanta T T and Finkelsteing A. Automated Consistency Checking for Multiperspective Software Specifications. In *Workshop on Advanced Separation of Concerns at ICSE 2001*.

[22] Xia Y and Glinz M. Rigorous EBNF-based Definition for a Graphic Modeling Language. In *Proceedings of APSEC 2003*, IEEE Computer Society Press.

[23] Chiorean D, Pasca M, Cârcu A, Botiza C, Moldovan S. Ensuring UML Models Consistency Using the OCL Environment. In *Electr. Notes Theor. Comput. Sci. 102*: 99-110 (2004)

[24] Simmonds J M. Bastarrica C. A Tool for Automatic UML Model Consistency Checking. In *Proc. of ASE'05.*

[25] Wagner R, Giese H and Nickel U A. A Plug-In for Flexible and Incremental Consistency Management. In [7]

[26] Gryce C, Finkelstein A, and Nentwich C. Lightweight Checking for UML Based Software Development. In [6]

[27] Nentwich C, Emmerich W, & Finkelstein A. Flexible Consistency Check, In *ACM TOSEM* 12(1), pp28-63, 2003.

[28] Xu, J. and Zhu, H., Requirements analysis and specification as a problem of software automation -- Some researches on requirements analysis, in *Proc. SEKE'96*, pp457~464.

[29] Accessible at http://www-st.inf.tu-dresden.de/ocl/

[30] Lange1 C, Chaudron M R V, Muskens J, Somers L J, Dortmans H M. An Empirical Investigation in Quantifying Inconsistency and Incompleteness of UML Designs. In [6].

WebLang : A Language for Modeling and Implementing Web Applications

Olivier Buchwalder, Claude Petitpierre

Networking Laboratory
Swiss Federal Institute of Technology in Lausanne
CH-1015 Lausanne EPFL, Switzerland
E-mail: {olivier.buchwalder, claude.petitpierre}@epfl.ch

Abstract

Nowadays Web applications are being developed by the thousand, most of the time successfully. However, too many development projects exceed deadlines and budget, or become unreliable. The large number of existing technologies and the difficulty involved in the practical application of current development methodologies are partially responsible for this situation. Even, the main existing approaches, such as MDA, based on graphical models, do not fully define the architecture and the business behavior. In this paper, we describe a DSL language, WebLang, that abstracts the application components with a useful model, and remains sufficiently close to the technology to reduce the enormous gap that exists between UML models and implementation. The WebLang solution provides a compiler and an editing tool, which produces easily usable prototypes. Thus, the validation of an architecture is possible early in the development process, avoiding certain implementation and integration problems.

1. INTRODUCTION

Web applications and distributed systems are currently booming, and the developer community is very active in this domain. Application servers and specialized frameworks provide the essential components for Web application development.

Three-tier architecture has been adopted by many application and framework designers. However, despite the fact that this global architecture is known and adopted by many companies, there are too many development projects that exceed deadlines and budget, or become unreliable. This problem is not exclusive to the domain of Web applications, but the large number of existing technologies and specialized frameworks, developed for this domain, highlight the weakness of the available development methodologies.

Currently, there is no universal development process with a designing language that is commonly used and mastered by the majority of the developers. The available methods based on the graphic UML notation bring rules and structure, but they don't respond entirely to the practical needs of the companies [19, 7]. Frequently, the latter elaborate homemade methods, specialized frameworks or code generation tools, to fully master their business.

However, the development methods mainly converge to a consensus about the use of models to abstract the system details, and to provide simpler views of the system structure and business behavior in the early development stage. The general approach where models are used as key elements during the whole development process is called Model Driven Development (MDD [1, 2]). Recently, new development methodologies or approaches, based on this trend, turned up, such as the Model Driven Architecture (MDA [13]), Executable UML [16] or WebML[4]. The main differences between the existing modeling approaches are the level of abstraction of the models and the nature of the notation language. Moreover, the methods require powerful CASE tools for the edition of the models, and for their transformation to executable code.

The UML modeling notation is an important actor of software engineering; most approaches are based on UML, subsets of UML, or UML 2.0 extensions for a specific domain. UML provides a graphical notation and includes specific representations for describing the architecture or the behavior concerns in an abstract way. The recent UML 2.0 version provides user-defined extensions through the use of tagged value, stereotypes and constraints. They enable to insert new notations or terminologies, while keeping the basis of UML generic. In practice, the compatibility between different extended UML diagrams is not guaranteed anymore, and these extensions lead to the apparition of incompatible UML dialects [18]. Moreover, UML is a complex language and the extension mechanism is also quite com-

plex. It is therefore not easy to understand how they will work in practice and how they will be manipulated and interpreted by tools to generate the code [8, 15, 17, 20].

The early implementation of Executable UML, such as Nucleus BridgePoint, uses a subset of UML plus certain rules to link the elements together. This extreme application of the MDD approach formalizes requirements and use cases, into a set of verifiable diagrams that can be executed, and expresses the specification of business behavior using action semantics, action languages and OCL constraints tags.

Several research efforts addressed the development of UML extensions for the domain of Web applications, such as the Conallen's approach [5], which defines UML extended diagrams for describing the client and server concerns of a Web application. Recently, the OMG released MDA, a standardized MDD approach, which is practically based on the UML extension mechanism. The MDA is a new way of developing applications and writing specifications, based on a platform-independent model (PIM) of the application and on automatic transformation into platform-specific models (PSMs) and into code. Several MDA tools exist, such as ArcStyler and OptimalJ, each of which introduces UML extensions for defining applications for the J2EE platform. However it's nontrivial to support and evolve correct PSMs for platforms such as J2EE or .NET. These platforms contain thousand of APIs and many of them are poorly documented. The application layer of middleware or additional libraries increase the difficulty to catch comprehensive models on which application can be built [20].

Beside UML extensions, more specific notations appeared for defining Web applications, such as Web Modeling Language (WebML), which was built on several previous Web design language proposals, including HDM[10], RMM, OOHDM, and Araneus. WebML provides a high level graphical notation and orthogonal models for designing structure, composition and presentation of a Web site; this approach is supported by the CASE tool WebRatio. WebML uses its own restricted notation and fails to express advanced composition and navigational constructs [11]. However, some research efforts have addressed the interaction between UML and WebML for defining behavior [14].

In this paper, we present WebLang, a Domain-Specific Language (DSL), which provides an adapted notation to define the architecture of J2EE Web applications. A DSL is a language designed to be useful for a specific domain, in contrast to general-purpose language (GPL), such as Java. WebLang brings abstraction for defining the structure and the business behavior of a Web application, and is intentionally closer to the target technologies than a general-purpose modeling language, such as UML. Indeed, in our opinion, the platform independent model is a theoretical good idea, but is practically difficult to apply for the wide area of existing technologies, and to maintain during the whole development process of an application. Our solution is designed to be used by architects and developers, who need to design well-defined architectures, and to generate rapidly testable prototypes on a specific platform.

The paper is organized as follows: in Section 2, the WebLang approach is presented. Section 3 shows the language specification, and finally Section 4 presents an example of a realistic application defined with WebLang.

2. WEBLANG APPROACH

WebLang is a Domain-Specific Language that makes it possible to define Web applications. The main motivation behind WebLang is to abstract the application components with a useful model, but to remain sufficiently close to the technology to reduce the enormous gap that exists between a PIM model and implementation. The WebLang approach expects to provide a simple and realistic method for designing the architecture of a Web application and a usable tool for generating a testable prototype.

The WebLang development process follows the MDD approach and brings a language model as key element of the development. The language syntax is oriented towards being naturally editable for a human in comparison with XML, which is more adapted to the machine. A WebLang application is defined by the assembling of several components that can specify each structural properties, business logic, and interconnections with other components.

The WebLang tool checks and compiles the model, and then generates the application in one atomic action. This approach is easier to implement than the incremental generation of application fragments. Furthermore, it provides the developers with a well-defined environment, where the whole application is defined with a unique and centralized model. All the generated files are standard and can be freely modified by the developer. The tool is integrated in the IBM Eclipse IDE, and is currently available for the J2EE JBoss platform, but the approach is extendable to other servers or technologies, by extending the templates or implementing new adapted modules.

2.1. Language Structure

The use of our own human-usable language allows us to freely specify the properties inherent to the technology without being limited by existing standards, such as the OMG Human-Usable Textual Specification (HUTN [12]), which is based on the OMG standards. We call a WebLang component a module, and a valid WebLang architecture is

defined by a set of module instances. The following grammar presents the syntax of a typical active component of a WebLang application.

```
<module_type> <moduleName> {
  <destination> <path>;
  ( <specialized_feature>; )*
  ( <field_type> <field_name>; )*
  ( <method_declaration> )*
}
```

The global syntax looks like well-known GPLs, such as Java or C, while the inner specification of a WebLang module is based on the target nature, in abstracting and simplifying the technology details.

2.2. Introduction of Business Logic

One of the benefits of using a text language as input model is the possibility to introduce the business logic directly and naturally into the WebLang source code. Currently, the language is dedicated to the J2EE platform; therefore a full Java 1.5 parser has been integrated into the tool by using JavaCC. The workflow, the Web-flow (3.2.3) or the business behavior can be inserted directly into the module declarations using the target platform language as shown in Section 3.1.

When the language syntax is parsed, the Java zones are checked and processed by the tool before being introduced into the generated files. During the parsing phase, certain relevant tokens or groups of tokens are recognized and stored. Then, when the module instance is processed, this Java-related information can be easily handled. Compared with Wizard-based and graphic MDA tools, this approach is simpler and offers greater flexibility. An early implementation of some .Net modules has shown that a C# parser is not essential, but brings facilities for implementing the tool, especially for checking the syntax.

2.3. Model Nature

The WebLang language is clearly a Platform Specific Model, which abstracts the implementation details of the target platform, but specifies precisely the relevant features of the target technologies. This model specification is text-based, and is defined in a constant and centralized way, contrary to several UML graphic models.

The designer can create WebLang models in which the full application prototype is defined without ambiguities. For instance, with WebLang, the architect decides at an early stage whether a service must be handled by a Servlet (3.2.3) or by a Session Bean (3.2.2), and the definition of the corresponding business behavior is easily and precisely defined in both situations. In our opinion, many implementation and integration problems result from models that are too abstract, incomplete or not fully mastered by the developer. When the components of the application are fully defined and when a usable prototype can be tested initially, the development process becomes less risky.

WebLang does not provide a platform independent model, such as the attractive MDA PIM. The latter provides facilities for switching between different technologies at any moment of the development. However, we think that currently the definition of a Web application with a PIM is too difficult to master and is not efficient enough. A PIM does not allow the specification of a significant and crucial part of most applications. For instance, the PIM cannot define the simplest application that every developer begins to write, *Hello World*, because there are no I/O libraries defined in the OMG standards [8].

On the other hand, the PSM and the WebLang approach also provide facilities for switching between different technologies, because the model synthesizes both the application structure and the business behavior; the conversion between PSMs is trivial for most cases. Moreover, when the PSM conversion would require a deep reengineering, the corresponding PIM would certainly be not trivial to specify. The definition of a PIM requires the additional intellectual work of an architect or a developer, and there is no assurance that a defined PIM would be reusable in the context of a new platform.

The WebLang model is defined with a unique input structure; the whole information concerning architecture and behavior is centralized in the same model. On the contrary, UML diagrams are not linked together by nature; each of them defines a fragment of structure or behavior of the application. This fragmented graphic approach is interesting for documentation or discussion on a specific part of the system, but it implies difficulties for developers and architects, who must deal with a set of diagrams, between which the interconnections are not trivial.

3. WEBLANG LANGUAGE

3.1. A Simple Example

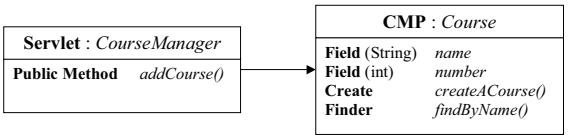


Figure 1. Two basic Web components

The figure 1 presents the abstract structural schema of a simple J2EE application. This application contains a Servlet instance (3.2.3), which is a Java Web service that usually returns an HTML response page, and a J2EE CMP

element that provides an object abstraction of a database table. The corresponding WebLang model is presented in the following code sequence. This definition not only abstracts the application architecture, but also specifies the application business behavior in the form of Java business code.

In this example, the Servlet method searches for a *Course* CMP bean instance. If the instance is found, its id is displayed on the output page; otherwise the *Course* bean is created.

```
servlet CourseManager {
  package pack;
  public void addCourse(String name, int id,
                                    PrintWriter out){
    try {
      Course c = courseHome.findByName(name);
      out.println("Course finded:" + c.getId());
    }catch (javax.ejb.FinderException fe) {
      courseHome.createACourse(name, id);
      out.println("Course created:" + id);
    }
  }
}
cmpbean Course {
  package pack;
  String name;
  int id;
  public Course createACourse(String name, int id);
  public Course findByName(String name) {
    query = "SELECT OBJECT (o) FROM Course o
             WHERE o.name = ?1"
  }
}
```

Public Methods — *Fields + Methods*

The WebLang generator reads these model inputs and produces mainly Java classes annotated with XDoclet tags, HTML or JSP pages, and XML script files.

3.2. Module Details

This section presents a syntax overview of the main modules. For the sake of simplicity, the specification of the modules is presented by examples rather than using BNF grammars.

3.2.1. Java Common Processing

The parsing of the following modules expects some mandatory or optional tokens, and is able to run an external Java 1.5 parser on specific blocks. The main Java parsing functions used are *BlockStatement*, *ClassBodyDeclaration* and the more localized *FieldDeclaration* and *MethodDeclaration*. Moreover, some parser extensions have been made for handling some specific declarations, such as the bean finder and creation methods.

```
Course c = courseHome.findByName("math");
Course c = courseHome.createACourse("math",12);
```

Each module processes the parsed Java information differently according to its needs, but a common processing is applied to most supported modules. The preceding code sequence shows the abstracted WebLang syntax for finding and creating a bean instance. These expressions can be used in all Java blocks, and are transformed by our tool into executable code compatible with the module in which they are defined. The bean type is automatically replaced by a reference to the remote or the local object proxy, and the bean home identifier by a reference to the corresponding home proxy object.

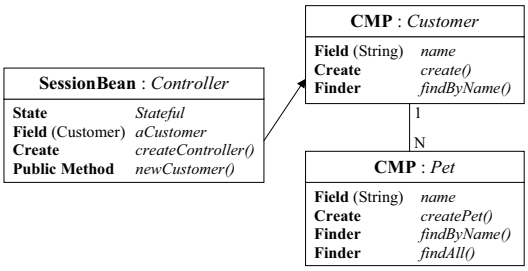


Figure 2. Interaction between three EJB modules

3.2.2. J2EE Enterprise Java Beans

WebLang is able to produce the J2EE EJBs classes under the specification of EJB 2.0. The figure 2 and the following code sequences show a typical interaction between three EJB modules, a Session bean and two CMP beans.

CMP Bean

A CMP bean (Container Managed Persistence Entity Beans) represents the object abstraction of a table in the database. Each instance of a defined CMP bean can be created, deleted and retrieved like a row of a database table. Some relations to other CMP beans can be defined after the package entry.

```
cmpbean Pet {
  package data;
  String name;
  public Pet createPet(String name);
  public Collection findAll() {
    query = "SELECT OBJECT (o) FROM Pet o"
  }
  public Pet findByName(String name) {
    query = "SELECT OBJECT (o) FROM Pet o
             WHERE o.name = ?1"
  }
}
cmpbean Customer {
  package data;
  relations (Pet = 1:N);
  String name;
  public Customer findByName(String name) {
    query = "SELECT OBJECT (o) FROM Customer o
             WHERE o.name = ?1"
  }
  public Customer create(String name);
}
```

CMP Fields — *Finders* — *Relation 1:N* — *Create*

CMP Relations: a relation must define the target bean name and the relation nature: *1:1, 1:1 target, 1:N* or *N:M*. When a relation is defined, the database table is configured with XDoclet tags, and getter-setter functions are generated in the class file.

CMP Fields: the Java class members are registered as CMP bean fields; they are mapped as table attributes and getter-setter methods are inserted in the CMP class.

CMP Finder Methods: some EJB finder methods are definable as specific method declarations. Their names must begin with the keyword *find* followed by an optional identifier name. The parameters may have either simple types or fully qualified types. This method declaration expects the definition of an EJB-QL query (see Sun EJB specification, Chapter 11).

EJB Creation Methods: some EJB creation methods are definable as specific method declarations. Their names must begin with the keyword *create* followed by an optional identifier name. The parameters must correspond to a subset of the attributes, placed in any order.

Java Methods: the method declarations are parsed and inserted in the generated class with the common transformations 3.2.1. The public methods are referenced in the bean's proxy and become accessible from the other modules.

Session Bean

The SessionBean is a service supplier, managed by the server and associated with a client session.

```
sbean Controller {
  package control;
  state Stateful;                                   Stateful
  Customer aCustomer;
  ControllerSession createController();
  public void newCustomer(String name){             Public Method
   aCustomer = customerHome.create(name);
  }
}
```

Session Type: the session can be declared *Stateful* and assigned permanently during a client session or *Stateless* and reassigned to other clients.

EJB Creation Methods: some EJB creation methods are definable in the module declaration, like for the CMP beans.

Java Methods: the method declarations are parsed and inserted in the generated class, like for the CMP beans.

3.2.3. Web Server Modules

The following client modules, Servlet and Struts, are by nature local to the server; they are typically hosted in a Web server, such as Apache Tomcat. Some others modules are remote such as the Java Client (see 3.2.4).

Servlet

A Servlet module is presented in Section 3.1. The generation of a module produces a standard Java Servlet class and a requesting HMTL page.

Java Methods: the method declarations are parsed and inserted in the generated class with the common transformations 3.2.1. Moreover, the public methods are processed specifically; they become accessible by calling the generated Servlet. For each public method, a requesting HTML form with all method's parameters, is included in the generated page.

Struts

A Struts module defines a state chart based on the Apache Struts framework. The following code sequence represents the business and the presentation layers of the application example, reviewed in Section 4.

```
struts Control {
 package store;
 stateMachine {
  statedef State_0 (pList);
  statedef State_1 (pConfirm);                      States definition

  switch (sessionState) {                           Java Block - FSM
   case State_S:
     makeForm(browseForm);
     sessionState = State_0;
     break;
   case State_0:
     clientForm.setNbOfItems(
       browseForm.getPetFormN().size());
     sessionState = State_1;
     break;
   case State_1:
     if (submit.equals("Confirm")) {
       String cfname = clientForm.getName();
       Customer cust;
       try {
         cust = customerHome.findByName(cfname);
       }catch(FinderException fe) {
         cust = customerHome.create(cfname);
       }
       for(PetForm pform: browseForm.getPetFormN()){
         if(pform.getFlag()){
           String pfname = pform.getName();
           cust.addPet(
             petHome.findByName(pfname));
         }
       }
     }
     sessionState = State_0; break;
  }
}

page pList {forms(browseForm.petFormN[](OK)); }    Pages
page pConfirm {forms(clientForm(Confirm),
                     cancelForm(Cancel)); }

private void makeForm(BrowseForm browseForm){      Java Method
  for(Pet pet : petHome.findAll()){
    PetForm pForm = new PetForm();
    pForm.setName(pet.getName());
    browseForm.getPetFormN().add(pForm);
  }
}

form BrowseForm {                                  Struts Action Forms
  package pack;
  forms (petFormN[]);
}
form ClientForm {
  package pack;
  fields(String name,
         int nbOfItems);
}

form PetForm {
  package pack;
  fields(String name,
         boolean flag);
}
form CancelForm {
  package pack;
}
```

The WebLang Struts module provides a safe Web application controller, in which the state of the client is preserved during the whole session. Some additional Java class files are automatically included in the project to manage the state machine. The generation of the module produces a Struts Action class, which contains the *state chart*. A JSP page is also produced for each declared *page*, and a Struts ActionForm class for each defined *form*.

State Definition: The definitions of the *states* are declared with a state name and an associated page name; $State0$ is reserved as the starting state.

Java State Chart: the state chart code is parsed by the Java *BlockStatement* function. Typically, a Java switch statement is anticipated to assign a new state according to the requested *form* values. However, the developer is free to manage the state evolution and to initialize the *form* values with the Java language.

Page Definition: each *page*, referenced in the state definition, must be declared formally, and is generated as a JSP page. A *page* must define a name and the list of included *forms*.

Forms: a Struts Action Form is basically a Javabean; it is used to host the *page* input and output data. The *form* defines some local fields and some included *sub-forms*.

3.2.4. Other Supported Modules

The WebLang language defines other modules by following the same approach: Message Driven Bean (MDB), Java Class, Java Client, RMI Object, Synchronous Java Class, Hibernate bean, JSP or HTML.

In addition to the presented proxy connections, WebLang supports the asynchronous exchanges between most modules in using the Java Messaging Service (JMS) specification.

4. DESIGN WITH WEBLANG

In order to demonstrate the value of our solution, we present in this section a realistic prototype of an elementary Web-store application. This three-tiered architecture provides a Web access to a *pet* database, and allows customers to make orders.

This application is composed of a database for storing persistent data; it mainly implements a Web service for handling the client requests, which accesses the system by using a Web browser. The full source code of this application is composed of the Struts and the CMP preceding code examples (3.2.3, 3.2.2). This application prototype is fully defined by WebLang with approximately fifty lines.

4.1. Scenario

A client accesses the *pet* database with a Web browser. The first page, displayed by the server, shows the list of all available *pets*. The client selects the elements that he wishes to order, and then submits his selection to the server. The server temporarily keeps the client selection, and responds with a confirmation page, showing the number of ordered *pets* and an empty field name. The client enters his name and clicks on the confirm button to finalize the order or on the cancel button to return directly to the starting page. If the order is confirmed, the selection is stored into the database; the customer instance is created if necessary and all selected *pets* are added to the customer's list.

4.2. Application Components

This application defines two CMP beans to store persistent data; the *Pet* table must be filled externally. The *Customer* CMP is linked to the *Pet* CMP with a 1:N relation. The Form components are used to handle the temporary data from and to the Pages. The Forms are basically Javabean objects with some Struts additions. The heart of the application is managed by the Struts component *Control*. It is mainly composed of a state chart that processes the client requests and accesses the database in a reliable way.

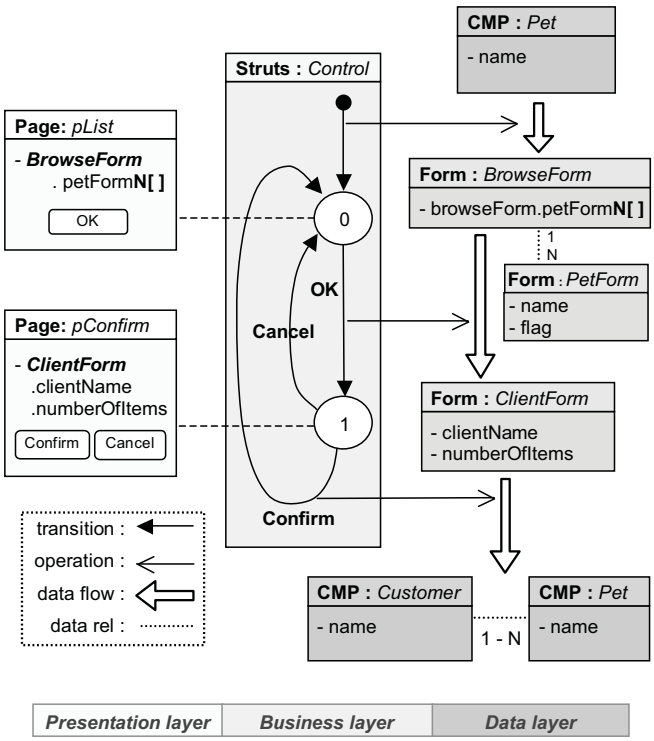


Figure 3. Elementary Web-Store

4.3. Diagram Overview

The figure 3 represents the architecture of this application. The WebLang components are all represented in the same diagram, as well as the principal behavior of the application. Currently, this diagram notation is informal, but it is for the most part usual for a developer. This schema is based on the following UML diagrams: state-chart, dataflow and collaboration diagrams. The boxes represent the

WebLang strongly typed components. The diagram is separated into the common three layers (presentation, business and data). The Struts instance is clearly the controller of the application, and each state, declared in the Struts module, is linked to a page. The data-flow evolution, represented in the schema by double arrows, is controlled by the transition of the state chart.

The business functionalities need to be extended to include the requirements of a commercial store, but the architecture defined by WebLang is really close to the final product. The generated prototype can be tested early in the development process, and the business functionalities can be developed on concrete and validated foundations.

Although we think that the edition of graphical models is less efficient and less flexible than editing a textual language with a good editor, the diagram visualization is interesting for documentation and development discussion, and we plan to release this tool extension.

5. CONCLUSION

We have presented a new approach for designing Web applications. The method is based on a language, WebLang, that is intentionally close to the technology. Our goal is to abstract the behavior and the structure of the application by providing well-defined components.

The advantage over some other modeling methods is that the WebLang language can specify the whole architecture of an application. The model is natural for a developer and more flexible than some graphic editions. The associated tool can compile and produce a usable prototype in one action. The architecture can then be tested early in the development process, thus avoiding certain implementation and integration problems.

Currently, WebLang only defines architecture for the Java J2EE platform. However, the concept is easily extendable to other technologies by adding new module declarations. We plan to support Web services soon, and to provide adapted modules for the Microsoft .Net platform in the future. Since the EJBs have no equivalent in the Microsoft specification, a .Net architecture is more difficult to devise in a standard and powerful way. The WebLang approach can provide a useful components set to help the developers to compose reliable applications.

The imminent introduction of Web services will transform WebLang into a SOA (Service Oriented Architecture [3]) designing tool, which will allow to compose heterogeneous application in a simple way.

5.1. Future Work

We are also implementing a generic designing tool, which extends the WebLang modeling approach to all kind of applications. This tool will allow the developers to specify and create their own DSL language, parser, generator and editor adapted to their needs.

This approach enables Language Oriented Programming [6, 9], which is a style of programming in which, rather than solving problems in GPLs, the programmer creates adapted DSLs and solves the problem in these languages.

References

[1] C. Atkinson and T. Kuehne. Aspect-oriented development with stratified frameworks. *IEEE Software, Volume 20, Issue 1, Jan.-Feb. 2003 Page(s):81 - 89*, 2003.

[2] C. Atkinson and T. Kuehne. Model-driven development: a metamodeling foundation. *IEEE Software, Volume 20, Issue 5, Sept.-Oct. 2003 Page(s):36 - 41*, 2003.

[3] J. Bloomberg. Principles of soa, 2003. ZapThink - ADTmag.com.

[4] S. Ceri, P. Fraternali, and A. Bongio. Webml: a modeling language for designing web sites. *Computer Networks (Netherlands)*, 33(1–6):137–157, 2000.

[5] G. Costagliola, F. Ferrucci, and R. Francese. Web engineering: Models and methodologies for the design of hypermedia applications, 2002.

[6] S. Dmitriev. Language oriented programming: The next programming paradigm, 2004.

[7] M. Fowler. *UML distilled: A brief Guide to the Standard Object Modelling Language*. Object Technology series. Addison Wesley, 3rd edition, 2004.

[8] M. Fowler. Language workbenches and model driven architecture, 2005.

[9] M. Fowler. Language workbenches: The killer-app for domain specific languages?, 2005.

[10] F. Garzotto, P. Paolini, and D. Schwabe. Hdm a model-based approach to hypertext application design. *ACM Trans. Inf. Syst.*, 11(1):1–26, 1993.

[11] E. Gorshkova and B. Novikov. Exploiting uml extensibility in the design of web applications, 2003.

[12] HUTN. *Human-Usable Textual Notation (HUTN) specification (OMG)*, 2002.

[13] A. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture-Practice and Promise*. Addison-Wesley, 2003.

[14] M. Matera, A. Maurino, S. Ceri, and P. Fraternali. Model-driven design of collaborative web applications. *Softw. Pract. Exper.*, 33(8):701–732, 2003.

[15] A. McNeile. Mda: The vision with the hole?, 2003.

[16] S. J. Mellor and M. J.Balcer. *Executable UML: A Foundation for Model-Driven Architecture*. Addison-Wesley, 2002.

[17] C. Petitpierre. *Software Engineering: The Implementation Phase*. EPFL-Press, 2006.

[18] B. Rumpe. Executable modeling with uml. a vision or a nightmare? *Issues and Trends of Information Technology Management in Contemporary Associations, Seattle*, pages 697–701, 2002.

[19] D. Thomas. Uml - unified or universal modeling language? *Object Technology, vol. 2, no. 1, January-February*, 2003.

[20] D. Thomas. Mda: Revenge of the modelers or uml utopia? *IEEE Software, vol. 21, no. 3, pp. 15-17*, 2004.

WECAP: A Web Environment for Project Planning

Lerina Aversano, Gerardo Canfora, Corrado Aaron Visaggio
Research Centre on Software Technology (RCOST)
viale Traiano,1 82100 Benevento, Italy
{Aversano, Canfora, Visaggio} @unisannio.it

Abstract

Competence management has been recently investigated with the aim of helping the planning of training programs for employees; leading the strategies for knowledge management within the organizations; and supporting the continuous check of skill gap. In this paper a web environment which manages competence across distributed software organizations is introduced. It supports the processes of team building in accordance with the project's business goals; enforces the project monitoring indicating the project's activities which lack necessary skills; helps the planning of the project, according to the available competencies.

1. Introduction

The approach adopted for managing competence, capabilities, and abilities within a software organization significantly impacts the failure or the success of projects. When staffing people, a project manager should select the competencies which fit better with the assigned tasks.
Unfortunately, this is not straightforward when dealing with software engineering for a number of reasons:

- the spectrum of software competencies is very wide and strongly differentiated: it's very difficult that a huge percentage of the required skills are concentrated in a few professionals; for instance, experts of artificial intelligence are not necessarily knowledgeable in database management system;
- software technologies undergo quick obsolescence;
- new paradigms, languages, and tools emerge every year: a long time experience might be not enough when a project needs the application of edge technologies;
- tacit knowledge plays a key role in software processes and, for its intrinsic nature, it can not be captured in easy-to-transfer form.

Moreover, many approaches adopted to store, retrieve and handle information about competencies are not directly applicable to software engineering .
So far, systems for managing competencies in software organizations address mainly the evolution of skills and abilities over time. Basically, such systems help maintain the knowledge about employees continuously update, throughout sophisticated mechanisms for defining e-learning programs which vary with accordance to the varying of business needs.
As discussed in the related work section, all the existing solutions do not let managing competence as a supporting process of the project management. This paper introduces a system that, conversely, integrates the competence management with the project management, and more precisely it supports:

- the team building, as it provides the project manager with different alternatives of project teams suitable for the project's goals, relying on the competencies available in the organization;
- the project monitoring, as it lets to accomplish statistical analysis on the fulfilment of required competencies at two levels of abstraction: project level and activity level;
- the human resources management, as it helps compare different employees on given competencies or abilities;
- the project planning (and re-planning), as it identifies activities which seriously lack skills and ability. The system lets project manager to re-plan activities or re-staff project on the fly.

The paper proceeds with the section of related work that discusses similarities and differences with other approaches in the literature and presents the context in which the system has been developed. The following section presents the system architecture, the competence and process models, and the main features. Finally concluding remarks and future works are discussed in the last section.

2. Related Work

Some authors [10] used competence for creating yellow pages or expert searches for leveraging tacit knowledge. Others suggest to derive goals for strategic development of knowledge, by relying on competence [11]. Competence profiles are also exploited in order to assign e-Learning courses to employees [8]; so far competencies are considered as a way to relate Knowledge Management and e-Learning activities [1]. Ley et al. [7] argue that Knowledge Management and e-learning initiatives, aiming at developing abilities and capabilities of the workforce, may be improved in two ways: by providing a closer connection between competencies and tasks, or by seeking competency development as an individually controlled learning process rather than a centrally managed development

initiative. Concerning competences management, [12] describes the use of ontology to realize: skill gap analysis, project team building, recruitment planning and training analysis. Unfortunately, this theoretical work was not yet transferred in industrial projects. Further works was accomplished in the direction of skill matching, as well as: [13] uses F-Logic-inferencing to derive competencies and compare profiles and [4] uses ontology based instance similarity for describing competencies. The problem with these approaches is that they do not discuss the integration with Human Resources Managing systems, in place in industry. [3] drew the inter-relations between Knowledge Management and Human Resource Management and explain how ontologies can be used for training planning. The authors focus on the algorithms and the business logic for mapping competence with employees, without any relation with the project management. Organizational Memory can be defined as the way an organization applies past knowledge to present activities. [5] conceptualizes and models the evolution of an organizational memory in order to account for metrics, modalities of cooperation, and the diversity of objects in the event space. [9] analyzes as Organizational Memory Systems can be used in order to support team building and management. The work is mainly headed to build the grounding theory. In Case Based Reasoning systems[6], experience is captured in a library of past cases, where each case typically contains a description of the problem and a description of the adopted solutions. In order to solve a new problem, the system searches for similar problems in the archive and propose those that fit better with the proposed case. Case based Reasoning was applied also to Project management as in [14]. The paper proposes an approach to improve workflow by collecting experience from its execution. However some other important issues have not been adequately coped, such as the integration with project management systems. In this sense, the system described in this paper provides the possibility to integrate these technologies.

2.1 The context: Virtual Districts Project

The system presented in this paper has been developed in the context of the Virtual Districts Project. It is an ongoing research project aiming to develop a framework of methods and tools supporting the start up and the evolution of virtual districts in the aerospace domain. It involves the Italian Space Agency (ASI) and a significant number of enterprises (about 40) operating in this domain. In particular, the project is focused on the following aspects:

- the analysis of main organizational problems of the enterprises that concur to the virtual district;
- the development of a knowledge management system that supports the need of the subject involved in the virtual district;
- the development of a middleware infrastructure that allow the creation of the virtual district, supporting the communication among the partners;
- the development of a workflow management infrastructure that support the automation of cross organizational project, exploiting the infrastructure in the previous points.

3. The WECAP environment

WECAP is the environment implemented to support critical activities related to the planning of software projects. It can represent practical means for collecting all the available qualitative and quantitative information regarding the competence, skills and workload of the enterprise employees. WECAP provides the information helpful to continuously control the status of the planned projects; such information is also used to improve the resources allocation. WECAP can be used both in the planning and monitoring stages of the project management. In the first stage, WECAP permits the allocation of the resources to the project by considering their competences. Subsequently, it supports the management of the project plan with dynamic analysis on the distribution of the people on the project activities, with the aim to ensure the necessary competences available for all the activities and tasks in the project.

3.1 Competence Model

The competence model captures the conceptual schema of skill / ability / know how employed and developed within the enterprise. As the managers of an enterprise need to share a coherent and common vocabulary when dealing with project staffing, a competence model should be defined. In order the WECAP system to be instantiated, we referred to a specific competence model. In particular, the key elements of this model are:

- the competence units, i.e. consistent aggregate of competencies, which are required to perform a specific activity: they are nucleus that can have a significant meaning in different area;
- the competencies which are the result of knowledge, ability and behaviours and which allow to realize specific task in a specific context.

The description of the competence is decomposed in single elements as in the following example:

> *Competence*: To be able to design software solution for e-commerce services
> *Expression*: to be able to
> *Verb/action*: design
> *Object:* software solutions
> *Context*: for e-commerce services

Modelling a project is not trivial and a number of different formalisms have been conceived in the attempt to capture all the various facets of interest [2]. In this context we use a model that specifies: the decomposition of the project in activities; the decomposition of the activities in tasks; for each activity the Name, Description, Objective, Cost, TimeFrame, Responsable, Project, and Competence;

and for each task the Name, Description, and Activity (it is assumed that the competence required for a task are the same of the activity they are part) . Then, in our case the emphasis is on the competence facet of the model.

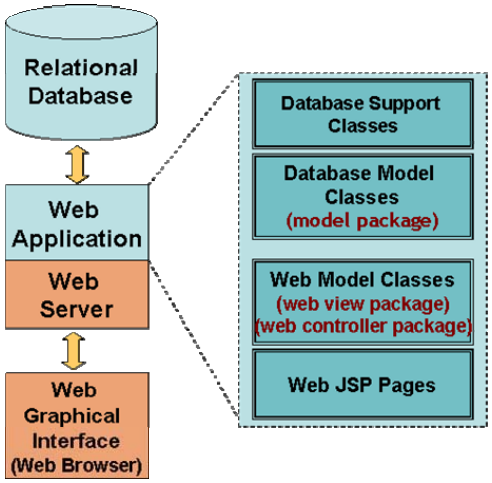


Figure 1: WECAP Architecture.

3.2 Use of WECAP environment

WECAP represents a practical means to achieve a rapid and effective allocation of the human resources. WECAP main functionalities are the following:

- to point up the most adequate allocation of each team member;
- to perform comparison among different staffing solutions;
- to provide a global view of the project that dynamically shows its status and highlights the possibilities to improve the distribution of the human resources.

The starting interface of WECAP is used for selecting the area (Project / Activities / Tasks / Competences / Statistics) to investigate. The Project area summarises all the information of interest for the project. It is mainly accessed by the Project Manager both for inserting data about new project and for querying/visualising data about on-going projects. The Activities and Tasks areas are used to define and visualize the structure of the project (with the requirements in term of competencies needed by each activity and each task and the related staffing status). The Competence area can be accessed only by the project Manager. This is used to manage the competence model. Finally there is the Statistics area. WECAP can be used to balance the workload on a project in order to optimise human resource allocation. If a project manager has limited resources who do not allow to adequately cover the competences required by all the project activities, s/he can use the system to understand how favour the activities belonging to the "critical path" but which can ensure the minimum Competence Gap. The WECAP Environment provides functionalities at two levels of abstraction:

- at project level it provides functionalities for Project Leaders/Team Leaders, who can view the status of the competences of the people they are in charge and can identities need for changes;
- at global level it provides functionalities for Project Managers, who can analyse the status of the competences of the people in a specific project, across multiple projects, and can modify the allocation of these.

This area allows for different kinds of analysis: ALLProject chart- which analyses all the project activities and compute the competence Gap at project level. The results are depicted in charts: Figure 2 shows there is a 25% of competence gap in the overall project; and in Figure 6 the competence gap for each project's activity is calculated. The competence gap is obtained by ranking the required competence set and the actual one. This facilitates the identification of needs for change.

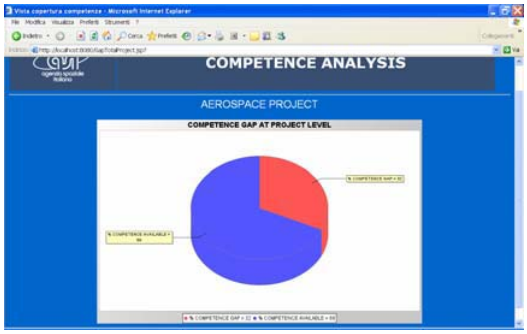


Figure 2: Overall Project's competence coverage.

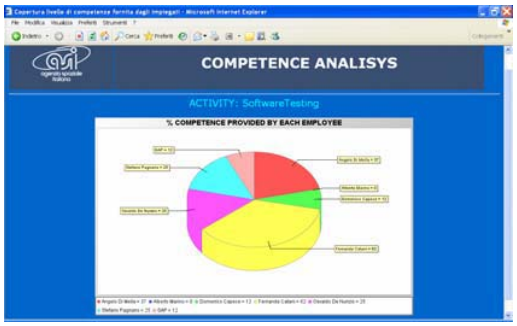


Figure 3: Personal competence coverage for the activity of Software Testing.

InProject chart- which analyses in detail the single project activities and compute the competence gap for each activity. The results are depicted in a chart, Figure 3 which reports the overall competence gap for the Software testing activity. Project HR chart-which analyses the project to understand the contribute in terms of competence of each staffed member. The results are depicted in a chart that reports the percentage of competence that each member provides in the project. Figure 3 shows the competence gap of each team member on the Software Testing activity. Beside this, a number of other features are provided,

such as the comparison among the skill of different team members.

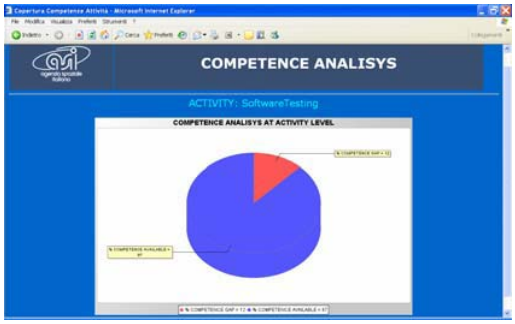


Figure 4: Projects' competence coverage for the activity of Software Testing

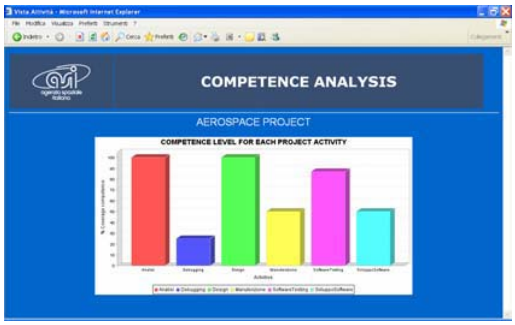


Figure 5: Competence gap for each project's activity

Conclusions

This paper introduced a software environment to support project management activities while staffing new projects and while monitoring on-going projects. The system, in particular, integrates the competence management with the project management, and more precisely it supports features for facilitating the team building; easing the project monitoring; and supporting the human resources management, as it helps compare different employees on given competencies or abilities. The system is part of a wider framework including methods and tools developed as port of a national research project, namely the Virtual District Project. The experimentation of the system, as the overall framework, in now going on and results will be available in a short term

References

[1] S. Adkins, "We are the problem: we are SellingSnakeOil",http://www.internettime.com/lcmt/archives/001014.html (last accessMarch 2006).

[2] P. Armenise, S. Bandinelli, C. Ghezzi, A. Morzenti, "A Survey and Assessment of Software Process Representation Formalisms", International Journal of Software Engineering and Knowledge Engineering, Vol. 3, no 3, 1993.

[3] E. Biesalski, A. Abecker, "Human Resource Management with Ontologies", proc. of Third Biennal Conference, WM 2005, Kaiserslautern, Germany, April 2005, in (eds K.-D. Althoff, A. Dengel, R. Bergmann, M. nick, and T. Roth-Berghofer) Professional Knowledge Management, LNAI 3782, Springer-Verlag 2005, Berlin-Heidelberg, pp. 499-507.

[4] S. Colucci, T. di Noia, E. Di Sciascio, F.M Donini, M. Mongello, M. Mottola, "A Formal Approach to Ontology Based Semantic Match of Skills description", Journal of Universal Computer Science, vol. 9, no. 12, 2003, Springer-Verlag, pp. 14371454.

[5] M. A. Evans and D.F. McMullen, "Proposing an Organizational Memory to Manage Knowledge Discovery in Scientific Collaborations" Proc. of Computer Supported Cooperative Work (CSCW'04), April 2004, Chicago, USA, ACM Press.

[6] I. Harrison, "Case Based Reasoning", 1997 (from: http://www.aiai.ed.ac.uk/links/cbr.html, march 2006).

[7] T. Ley, S. N. Lindstaedt, and D. Albert, Supporting Competency Development in Informal Workplace Learning", proc. of Thierd Biennal Conference WM 2005, Kaiserslautern 2005, in K.D.Althoff et al. (Eds.): WM 2005, LNAI 3782, Springer-Verlag Berlin Heidelberg, 2005, pp.189-202.

[8] T. Ley, A. Ulbrich, "Achieving benefits through integrating eLearning and Strategic Knowledge Management", European journal of Open and Distance Learning, 2002.

[9] H. Linger and F. Burstein, "Learning in Organizational Memory Information Systems: An Intelligent Decision Support Perspective" in Thirty-First Annual Hawaii International Conference on System Sciences, Kohala Coast, Hawaii, USA, January 6-9, 1998. Volume 4: IEEE Computer Society.

[10]J.R. Reich, P. Brockhausen, T. Lau, U. Reimer, "Ontology Based Skills Management: Goals, Opportunities and Challenges" in Journal of Universals Computer Science, Vol. 8, no. 5, 2002, pp. 505-515.

[11]G.J.B. Probst , A. Deussen, M.J. Eppler, S.P. aub, "Kompetenz-management: Wie Individuen und Organisationen Kompetez entwickeln" Gabler, Wiesbaden , 2000.

[12]J. Stader, A. Macintosh, "Capability Modelling and knowledge Management", proc. of the 19th Int'l Conference of the BSC Specialist Group on Knowledge-based system and applied artificial Intelligence, pp. 35-50, Springer-Verlag.

[13]Y. Sure, A. Maedche, S. Staab, "Leverage Corporate Skill Knowledge- From ProPer to OntoProPer" proc. of the 3rd Int'l Conference on Practical Aspects of Knowledge Mangement, 2000.

[14]B. Weber and W. Wild, "Towards the Agile Management of Business Preocesses", proc. of Third Biennal Conference , WM 2005, Kaiserslautern Germany, April 2005, in Professional Knowledge Management, LNAI 3782, Springer-Verlag 2005, Berlin-Heidelberg.

A Workflow Mining Tool based on Logs Statistical Analysis

Walid Gaaloul, Claude Godart
LORIA - INRIA - CNRS - UMR 7503
BP 239, F-54506 Vandœuvre-lès-Nancy Cedex, France
Email: gaaloul@loria.fr, godart@loria.fr

***Abstract*— As more and more Workflow management systems (WfMS)s log every event occurring during the process execution, customers become increasingly interested in managing process executions. Specifically, there is a desire for getting more visibility into process executions to be able to quickly spot problems. Our contribution is an original statistical technique to discover workflow patterns from event-based logs. Indeed, our approach is characterised by a "local" workflow patterns discovery that allows to cover partial results through a dynamic programming algorithm. Furthermore, we improve the discovery of the activities concurrent behaviour by proposing a dynamic mechanism of logs window sliding. Our approach has been implemented within the ProM framework, as a plug-in.**

keywords : workflow mining, workflow patterns, business process analysis, business process intelligence.

I. Introduction

WfMSs have traditionally focused on supporting the modeling of business processes, with the aim of enabling more efficient and more effective process executions. Business process intelligence is a discipline in which analysing execution traces (logs) of those complex systems will enable them to be well understood, controlled, and redesigned. Our paper is a contribution to this issue in the particular context of workflow mining by workflow logs analysis.

Business intelligence has a long and successful history of building statistical models [1]. In this paper, we propose an original statistical approach to discover workflow patterns from event-based logs. Our approach starts by collecting logs from workflow processes instances as they took place. Then, it builds, through statistical techniques, a graphical intermediary representation modelling elementary dependencies over workflow activities executions. These dependencies are then refined to discover workflow patterns. Those discovered workflow patterns are then composed iteratively until discovering the global workflow model.

This paper is structured as follows. Section II explains our workflow logs model. Section III details our workflow patterns mining algorithm. Section IV discusses related work, implementation and perspectives issues, before concluding.

II. Workflow Logs Model

The workflow specification might not be concerned with the details of the activities however it would have to deal, at least, with the externally visible completion events of activities (such as aborted, failed, and completed). Currently, most of WfMSs logs all events occurring during process execution. We expect the activities to be traceable, meaning that the system should in somehow keep track of ongoing and past executions. A workflow log is composed of a set of EventStreams (definition 2.1). Each EventStream traces the execution of one case (instance). It consists of a set of events (Event) that captures the activities life cycle performed in a particular workflow instance. An Event is described by the activity identifier that it concerns, the current activity state (aborted, failed, completed or compensated) and the time when it occurs (TimeStamp). A Window defines a set of Events over an EventStream. Finally, a Partition builds a set of partially overlapping Windows partition over an EventStream.

Definition 2.1: (EventStream)
An EventStream represents the history of a workflow instance events as a tuple EventStream= (begin, end, sequenceLog, SOccurrence) where:

✓begin : TimeStamp is the moment of log beginning ;

✓end : TimeStamp is the moment of log end;

✓sequenceLog : Event* is an ordered Event set belonging to a workflow instance;

✓SOccurrence : int is the activity instance number. A WorkflowLog is a set of EventStreams. WorkflowLog=(workflowID, {EventStream_i, $0 \leq i <$ number of workflow instances}) where EventStream_i is the event stream of the i^{th} workflow instance.

Here is an EventStream related to an instance of workflow in figure 1. This EventStream was filtered to take only events with completed as state.

$\mathbf{L}$ = EventStream((13/5/2005,5:42:12), (14/5/2005, 14:01:54), [Event("A_1", completed, (13/5/2005, 5:42:12)), Event("A_2", completed, (13/5/2005,11:11:12)), Event("A_4", completed, (13/5/2005,14:01:54)), Event("A_3", completed, (14/5/2005, 00:01:54)), Event("A_5", completed, (14/5/2005,5:45:54)), Event("A_7", completed, (14/5/2005,10:32:55)), Event("A_9", completed, (14/5/2005,14:01:54))])

III. Mining structural workflow patterns

As we stated before, we start by collecting WorkflowLog from workflow instances as they took place. Then we build, through statistical techniques, a graphical intermediary representation modelling **elementary dependencies** over work-

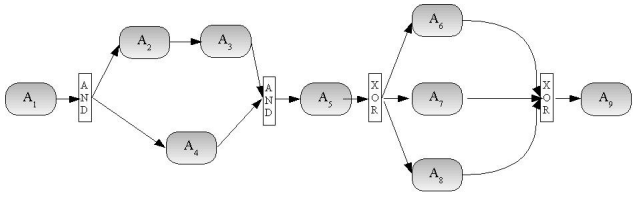


Fig. 1. Workflow running example

flow logs (section III-A). These dependencies are refined by **advanced structural workflow patterns** (section III-B). An elementary dependency is an "immediate" dependency[1] linking two activities in the sense that the termination of the first causes the activation of the last. Thus, the event of termination of the first activity is considered as the pre-condition of the activation of the last and reciprocally the activation of the last is considered as a post condition of the termination of the first activity. While an advanced structural workflow pattern is a set of elementary dependencies that defines an advanced structure to express specific behaviour, in terms of control flow, linking these dependencies. A pattern is the abstraction from a concrete form which keeps recurring in specific non arbitrary contexts. Thus, a workflow pattern [2] can be seen as an abstract description of a recurrent class of interactions based on (primitive) activation dependency.

A. Discovering elementary dependencies

The aim of this section is to explain our algorithm for discovering elementary dependencies among a WorkflowLog and build an intermediary model representing those dependencies: statistical dependency table (or SDT).

1) Discovering direct dependencies: In order to discover direct dependencies from a WorkflowLog, we need an intermediary representation of this WorkflowLog through a statistical analysis. We call this intermediary representation : statistical dependency table (or SDT) which is based on a notion of frequency table [3]. As workflow patterns are described only by control flow dependencies, this table captures control flow direct dependencies which are related exclusively to activities "terminated" state dependencies reporting "correct" (i.e., without "exceptions") executions. There is no need to use other EventStreams relating to failure executions containing failed or aborted or compensated states. In fact, these cases concern only transactional behaviour and dependencies which tailors the mechanisms for failures handling and recovery. These issues are out of the scope of our paper. Nevertheless, in [4], [5] we use these events to discover and improve workflow transactional behaviour. Consequently, to mine workflow patterns, we need to filter the analysed WorkflowLog and take only EventStreams of instances executed "correctly".

Basically, SDT is built through a statistical calculus that extracts elementary dependencies between activities of a WorkflowLog that are executed without "exceptions" (*i.e.* they reached successfully their completed state). We denote by

[1]Terms *immediate* and *direct* will be used interchangeably in the remainder of the paper

P	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9
A_1	0	0	0	0	0	0	0	0	0
A_2	**0.54**	0	0	0.46	0	0	0	0	0
A_3	0	**0.69**	0	0.31	0	0	0	0	0
A_4	**0.46**	0.31	0.23	0	0	0	0	0	0
A_5	0	0	**0.77**	**0.23**	0	0	0	0	0
A_6	0	0	0	0	1	0	0	0	0
A_7	0	0	0	0	1	0	0	0	0
A_8	0	0	0	0	0	0	0	0	0
A_9	0	0	0	0	0	0.27	0.62	0.11	0

$\#A_1 = \#A_2 = \#A_3 = \#A_4 = \#A_5 = \#A_9 = 100$, $\#A_6 = 27, \#A_7 = 62, \#A_8 = 0.11$

TABLE I
INITIAL STATISTICAL DEPENDENCIES TABLE ($P(x/y)$) AND ACTIVITIES FREQUENCIES ($\#$)

$\text{WorkflowLog}_{completed}$ this workflow logs selection. Thus, the unique necessary condition to discover elementary dependencies is to have workflow logs containing at least the completed event states. These features allow to mine control flow from "poor" logs which contain only completed event states. By the way, any information system using transactional systems or workflow management systems offer this information in some form.

For each activity A, we extract from $\text{WorkflowLog}_{completed}$ the following information in the statistical dependency table (SDT): (i) The overall occurrence number of this activity (denoted $\#A$) and (ii) The elementary dependencies to previous activities B_i (denoted $P(A/B_i)$). The size of SDT is $n * n$, where n is the number of workflow activities. The (*m,n*) table entry (notation $P(A_{0\leq i<n}/A_{0\leq j<n})$) is the frequency of the j^{th} activity immediately preceding the i^{th} activity. The initial SDT in table I represents of the SDT of our workflow example given in figure 1. For instance, in this table P(A_3/A_2)=0.69 expresses that if A_2 occurs, then, we have 69% of chance that A_3 occurs directly after A_2 in the workflow logs. As it was calculated SDT presents some problems to express correctly activities dependencies relating to the concurrent behaviour. In the following, we detail these issues and propose solutions to correct them.

2) Discarding erroneous dependencies: If we assume that each EventStream from WorkflowLog comes from a sequential (i.e no concurrent behaviour) workflow, a zero entry in SDT represents a causal independence and a non-zero entry means a causal dependency relation (*i.e.* sequential or conditional relation). But, in case of concurrent behaviour, as we can see in workflow patterns (like and-split, and-join, etc.), the EventStreams may contain interleaved events sequences from concurrent threads. As consequence, some entries in initial SDT can indicate non-zero entries that do not correspond to dependencies. For instance, the events stream given in section II "suggests" erroneous causal dependencies between A_2 and A_4 , and between A_4 and A_3. Indeed, A_2 comes immediately before A_4 and A_4 comes immediately before A_3 in this events stream. These erroneous entries are reported by $P(A_4/A_2)$ and $P(A_3/A_4)$ in initial SDT which are different to zero. These entries are erroneous because there are no causal dependencies between these activities as suggested (i.e. noisy

SDT). Underlined values in table I report this behaviour for other similar cases.

Formally, two activities A and B are in concurrence *iff* $P(A/B)$ and $P(B/A)$ entries in SDT are non-zero entries in SDT. Based on this definition, we propose an algorithm [6] to discover services parallelism and then mark the erroneous entries in SDT. This algorithm scans the initial SDT and marks concurrent services dependencies by changing their values to (-1). Through this marking, we can eliminate the confusion caused by concurrent behaviors producing these erroneous non-zero entries.

3) Discovering indirect dependencies.: For concurrency reasons, an activity might not depend on its immediate predecessor in the events stream, but it might depend on another "indirectly" preceding activities. As an example of this behaviour, A_4 is logged between A_2 and A_3 in the events stream given in section II. As consequence, A_2 does not occur always immediately before A_3 in the workflow logs. Thus, we have only $P(A_3/A_2) = 0.69$ that is an under estimated dependency frequency. In fact, the right value is 1 because the execution of A_3 depends exclusively on A_2. Similarly, values in bold in initial SDT report this behaviour for other cases.

Definition 3.1: Window

A log window defines a log slide over an events stream S : **stream** (bStream, eStream, sLog, workflowocc). Formally, we define a log window as a triplet **window**(wLog, bWin, eWin):

∘ (bWin : TimeStamp) and (eWin : TimeStamp) are the moment of the window beginning and end (with bStream $\leq$ bWin and eWin $\leq$ eStream)

∘ wLog $\subset$ sLog and $\forall$ e: **event** $\in$ S.sLog where bWin $\leq$ e.TimeStamp $\leq$ eWin $\Rightarrow$ e $\in$ wLog.

To discover these indirect dependencies, we introduce the notion of *activity concurrent window* (definition 3.1). An *activity concurrent window* (ACW) is related to the activity of its last event covering its directly and indirectly preceding activities. Initially, the width of ACW of an activity is equal to 2. Every time, this activity is in concurrence with an other activity we add 1 to this width. If this activity is not in concurrence with other activities and has preceding concurrent activities, then we add their number to ACW width. For example, the activity A_4 is in concurrence with A_2 and A_3 the width of its ACW is equal to 4. Based on this we give an algorithm [6] that calculates the ACW width for each service and regroups them in the ACW table. This algorithm scans the "marked" **SDT** and updates the ACW table in consequence.

Definition 3.2: Partition

A **partition** builds a set of partially overlapping windows over an events stream.

Partition : WorkflowLog $\rightarrow$ (Window)*

S : EventStream(sLog, workflowocc, bStream, eStream) $\rightarrow$ $\{w_i$:Window; 0≤i<n$\}$:

∘ w_1.bWin = bStream and w_n.eWin = eStream,

∘ $\forall w : window \in partition$, e:Event= the last event in w, *width*(w)= ACWT[e.ActivityID],

∘ $\forall$ 0≤i<n; w_{i+1}.wLog - {the last e:Event in w_{i+1}.wLog} $\subset$ w_i.wLog and w_{i+1}.wLog $\neq$ w_i.wLog.

After that, we proceed through an EventStreams partition (definition 3.2) that builds a set of partially overlapping windows over the EventStreams using the ACW table. Finally, we give an algorithm [6] that computes the final SDT. For each ACW, it computes for its last activity the frequencies of its preceded activities. The final SDT will be found by dividing each row entry by the frequency of the row's activity. Note that, our approach adjust **dynamically**, through the width of ACW, the process calculating activities dependencies. Indeed, this width is sensible to concurrent behaviour : it increases in case of concurrence and is "neutral" in case on concurrent behaviour absence. Thus, our algorithm that adapts its behaviour to new "concurrent" context. This strategy allows the improvement of the algorithm's complexity and runtime execution. Now by applying these algorithms, we can compute the final SDT (table II) which will be used to discover workflow patterns.

P	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9
A_1	0	0	0	0	0	0	0	0	0
A_2	**1**	0	0	-1	0	0	0	0	0
A_3	0	**1**	0	-1	0	0	0	0	0
A_4	**1**	-1	-1	0	0	0	0	0	0
A_5	0	0	**1**	**1**	0	0	0	0	0
A_6	0	0	0	0	1	0	0	0	0
A_7	0	0	0	0	1	0	0	0	0
A_8	0	0	0	0	0	0	0	0	0
A_9	0	0	0	0	0	0.27	0.62	0.11	0
$\#A_1 = \#A_2 = \#A_3 = \#A_4 = \#A_5 = \#A_9 = 100,$ $\#A_6 = 27, \#A_7 = 62, \#A_8 = 0.11$									

TABLE II
FINAL STATISTICAL DEPENDENCIES TABLE ($P(x/y)$) AND ACTIVITIES FREQUENCIES (#)

B. Discovering advanced dependencies : Workflow Patterns

1) Patterns Statistical Properties: We have identified three kinds of statistical properties (sequential, conditional and concurrent) which describe the main behaviours of workflow patterns. Then, we have specified these properties using SDT's statistics. We use theses properties to identify separately workflow patterns from workflow logs. This behaviour provides a dynamic algorithm that builds global solution (i.e. global workflow) based on local solutions (i.e. workflows patterns) iteratively. We begin with the statistical exclusive dependency property (property 1) which characterises, by the way, the sequence pattern.

Property 1: ***Mutual exclusive dependency property*** (as *P1*):

A **mutual** exclusive dependency relation between an activity A_i and its immediately previous activity A_j specifies that the enactment of the activity A_i depends only on the completion of activity A_j and the completion of A_j enacts only the execution of A_i. It is expressed in terms of:

∘ activities frequencies : $\#A_i = \#A_j$

∘ activities dependencies : $P(A_i/A_j) = 1 \wedge 0 \leq k, l < n; k \neq j; P(A_i/A_k) = 0 \wedge \forall l \neq i; P(A_l/A_j) = 0.$

The next two statistical properties: concurrency property (property 2) and choice property (property 3) are used to

insulate statistical patterns behaviour in terms of concurrence and choice after a "fork" or before a "join" operator.

Property 2: ***Concurrency property*** (as *P2*):
A **concurrency** relation between a set of activities $\{A_i, 0 \leq i < n\}$ belonging to the same workflow specifies how, in terms of concurrency, the enactment of these activities is performed. This set of activities is commonly found after a "fork" operator or before a "join" operator. We have distinguished three activities concurrency behaviours:

• **P2.1: Global concurrency** where in the same instantiation the whole activities are performed simultaneously : $\forall 0 \leq i \neq j < n;\ \#A_i = \#A_j \wedge P(A_i/A_j) = -1$

• **P2.2: Partial concurrency** where in the same instantiation we have at least a partial concurrent execution of activities : $(\exists 0 \leq i \neq j < n;\ P(A_i/A_j) = -1)$

• **P2.3: No concurrency** where there is no concurrency between activities: $\forall (0 \leq i \neq j < n; \wedge P(A_i/A_j) \neq -1)$

Property 3: ***Choice property*** (as *P3*):
A **choice** is a relation between the two operands before and after the "join" and the "fork" operator. It specifies, in terms of control flow, how the workflow instance performs the choice of activities' operands activations (*i.e.* which activities are executed after a "fork" operator or before a "join" operator). The two operands of the "fork" operator (respectively the "join" operator) performing this relation are : (operand 1) an activity A from which comes (respectively to which) a single thread of control which splits (respectively converges) into (respectively from) (operand 2) multiple activities $\{A_i, 0 \leq i < n\}$. We have distinguished three activities choice behaviours :

• **P3.1: Free choice** where a part of activities from the second operand are chosen. We have in terms of activities frequencies $(\#A \leq \Sigma_{i=0}^{n-1}(\#A_i)) \wedge (\#A_i \leq \#A)$ and in terms of activities dependencies we have :

◦ In "fork" operator (A_i occurs certainly after A occurrence): $\forall 0 \leq i < n; P(A_i/A) = 1$

◦ In "join" operator (A occurs certainly after some A_i occurrences "1 <", but not always after all A_i "< n") : $1 < \Sigma_{i=0}^{n-1} P(A/A_i) < n$

• **P3.2: Single choice** where only one activity is chosen from the second operand. We have in terms of activities frequencies $(\#A = \Sigma_{i=0}^{n-1}(\#A_i)))$ and in terms of activities dependencies we have :

◦ In "fork" operator (A_i occurs certainly after A occurrence): $\forall 0 \leq i < n; P(A_i/A) = 1$

◦ In "join" operator (A occurs certainly after only one of A_i occurrences): $\Sigma_{i=0}^{n-1} P(A/A_i) = 1$

• **P3.3: No choice** where all activities in the second operand are executed. We have in terms of activities frequencies $\forall 0 \leq i < n$, $\#A = \#A_i$ and in terms of activities dependencies we have :

◦ In "fork" operator (A_i occurs certainly after A occurrence): $\forall 0 \leq i < n; P(A_i/A) = 1$

◦ In "join" operator (A occurs certainly after all A_i occurrences): $\forall 0 \leq i < n; P(A/A_i) = 1$

Using statistical properties 1, 2 and 3, the last step is the identification of workflow patterns through a set of rules. In fact, each pattern has its own statistical rules which abstract statistically its causal dependencies, and identify it in a unique manner. These rules allow, if workflow logs is complete, the discovery of the whole workflow patterns included in the mined workflow. To be complete, workflow logs should cover all possible cases (i.e. if a specific routing element can appear in the mined workflow model, the log sshould contain an example of this behaviour in at least one case).

Our control flow mining rules are characterised by a "local" workflow patterns discovery. We propose a different approach that follows **dynamic programming** technique that deals with concurrency and dynamic programming algorithm and mines iteratively the global workflow application by composing local patterns mining techniques. Indeed, these rules proceed through a **local logs analysing** that allows to **recover partial results** of mining workflow patterns. In fact, to discover a particular workflow pattern we need only events relating to pattern's elements. Thus, even using only fractions of workflow logs, we can discover correctly corresponding workflow patterns (which their events belong to these fractions).

We divided the workflows patterns in three categories (c.f figure 2) : sequence, split and join patterns. In the following, we present rules to discover the most useful workflow patterns belonging to these three categories.

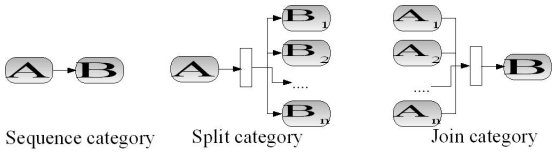


Fig. 2. Workflow patterns categories

2) Discovering sequence pattern: In this category, we find out only the sequence pattern (table III). In this pattern, the enactment of the activity B depends only on the completion of activity A. So we have used the statistical exclusive dependency property to ensure this relation linking B to A.

For instance, by applying the rules on this pattern over the FSDT table (*i.e.* table II), we discover a sequence pattern linking A_2 and A_3. Indeed, $(\#A_2 = \#A_3)$ and $(P(A_2/A_3) = 1)$ and $\forall A_{0 \leq i < n} \neq A_2; P(A_3/A_i) = 0$ and $\forall A_{0 \leq j < n} \neq A_2; P(A_j/A_3) = 0$.

3) Discovering fork *patterns:* This category (table III) has a "fork" operator where a single thread of control splits into multiple threads of control which can be, according to the used pattern, executed or not.

The dependency between the activities A and B_i before and after "fork" operator differs in the three patterns of this category: xor-split, and-split, or-split. These dependencies are characterised by the statistical choice properties. The xor-split pattern, where one of several branches is chosen after "fork" operator, adopts the single choice property (P3.2). and-split and xor-split patterns differentiate themselves through the no choice (P3.3) and free choice (P3.1) properties. Effectively, only a part of activities are executed in the or-split pattern after a "fork" operator, while all the B_i activities are executed in the

sequence	Rules		
	$(\#B = \#A) \wedge (P(B/A) = 1) \wedge \forall A_{0 \leq i < n} \neq A; P(B/A_i) = 0 \wedge \forall B_{0 \leq j < n} \neq B; P(B_j/A) = 0$		
split	**Rules**	join	**Rules**
(xor)	$(\Sigma_{i=0}^{n-1}\ (\#B_i)$=#A) $\wedge$ $(\forall 0 \leq i < n; P(B_i/A) = 1) \wedge$ $(\forall 0 \leq i \neq j < n;\ P(B_i/B_j) = 0)$	(xor)	$(\Sigma_{i=0}^{n-1}\ (\#A_i)$=#B) $\wedge$ $\Sigma_{i=0}^{n-1}$ P(B/A_i)=1) $\wedge$ $\forall 0 \leq i \neq j < n;\ P(A_i/A_j) = 0$
(and)	$((\forall 0 \leq i < n; \#B_i$=#A) $\wedge$ $(\forall 0 \leq i < n;\ P(B_i/A) = 1) \wedge$ $(\forall 0 \leq i \neq j < n\ P(B_i/B_j) = -1)$	(and)	$(\forall 0 \leq i < n;\ \#A_i$=#B) $\wedge$ $(\forall 0 \leq i < n;\ P(B/A_i) = 1) \wedge$ $(\forall 0 \leq i \neq j < n\ P(A_i/A_j) = -1)$
(or)	$\#A \leq \Sigma_{i=0}^{n-1}\ (\#B_i)) \wedge$ $(\forall 0 \leq i < n;\ \#B_i \leq \#A)$ $(\forall 0 \leq i < n;\ P(B_i/A) = 1) \wedge$ $(\exists 0 \leq i \neq j < n; P(B_i/B_j) = -1)$	(M-out -of-N)	$(m * \#B \leq \Sigma_{i=0}^{n-1}\ (\#A_i))$ $\wedge\ (\forall 0 \leq i < n;\ \#A_i \leq \#B)$ $(m \leq \Sigma_{i=0}^{n-1}\ P(B/A_i) \leq n)$ $\wedge\ (\exists 0 \leq i \neq j < n;\ P(A_i/A_j) = -1)$

TABLE III
RULES OF sequence, split AND join PATTERNS

and-split pattern. The non-parallelism between B_i, in the xor-split pattern are ensured by the no concurrency property while the partial and the global parallelism in or-split and and-split is identified through the application of the statistical partial and global concurrency properties. For instance, the FSDT table (*i.e.* table II), indicates that we have an and-split pattern linking A_1, A_2 and A_4. In fact,there is a global parallelism between A_2 and A_4 and these activities depend exclusively on A_1.

4) Discovering join *patterns:* This category (table III) has a "join" operator where multiple threads of control merge in a single thread of control. The number of necessary branches for the activation of the activity B after the "join" operator depends on the used pattern. To identify the three patterns of this category: xor-join pattern, and-join pattern and M-out-of-N-Join pattern we have analysed dependencies between the activities A_i and B before and after "join". Thus, the single choice (P3.2) and the no concurrency (P2.3) properties are used to identify the xor-join pattern where two or more alternative branches come together without synchronisation and none of the alternative branches is ever executed in parallel. As for the and-join pattern where multiple parallel activities converge into one single thread of control, the no choice (P3.3) and the global concurrency (P2.3) are both used to discover this pattern. In contrary of the M-out-of-N-Join pattern, where we need only the termination of M activities from the incoming n parallel paths to enact the B activity, The concurrency between A_i would be partial (P2.2) and the choice is free (P3.1). For instance, using FSDT table (*i.e.* table II) we mine an xor-join pattern linking A_6, A_7, A_8 and A_9. In fact, the FSDT's entries of these activities indicate non concurrent behaviour between A_6, A_7 and A_8 and the execution of A_9 depend on the termination of all these last activities.

IV. DISCUSSION

In this paper, we discussed issues related to patterns workflow mining from event-based Logs. Obvious applications of workflow patterns mining exist in model driven business process software engineering, both for bottom up approaches used in business process alignement [7], [8], and for top down approaches used in workflow generation [9]. The idea of applying process mining in the context of workflow management was first introduced in [10]. This work proposes methods for automatically deriving a formal model of a process from a log of events related to its executions and is based on workflow graphs. Cook and Wolf [11] investigated similar issues in the context of software engineering processes. Herbst [12] presents an inductive learning component used to support the acquisition and adaptation of sequential process models, generalising execution traces from different workflow instances to a workflow model covering all traces. Starting from the same kind of process logs, van der Aalst et al. propose techniques to discover workflow models based on Petri nets. Beside analysing process structure, there exist related works dealing with process behaviour reporting, such as [13], [14] that describe a tool that provides several features, such as analysing deadline expirations, predicting exceptions, process instances monitoring.

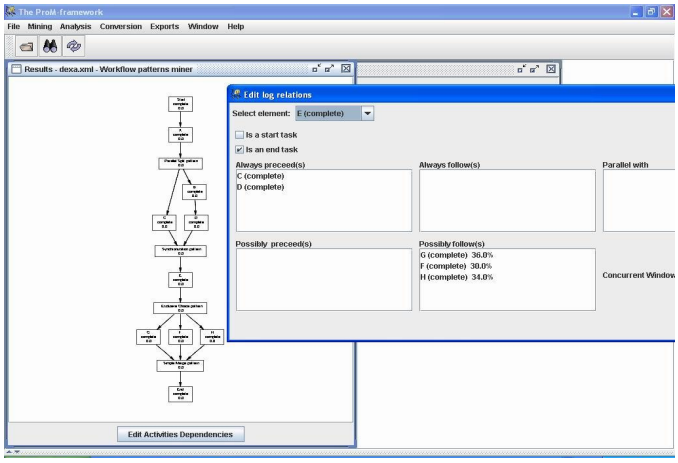


Fig. 3. Workflow patterns miner plug-in

Our contribution proposes a new approach characterised by a **partial discovery** of workflow patterns. This approach recovers partial results from logs fractions. It discovers **more complex features** with a better specification of "fork" operator (and-split, or-split, xor-split patterns) and "join" operator (and-join, xor-Join, and M-out-of-N-Join patterns). It deals better with concurrency through the introduction "*concurrent window*" that deals **dynamically** with concurrence. It seems to be

Tools	EMiT [15]	Little Thumb [16]	InWoLvE [17]	Process Miner [18]	Workflow patterns miner plug-in
Structure	Graph	Graph	Graph	Block	**Patterns**
Local discovery	No	No	No	No	**Yes**
Parallelism	Yes	Yes	Yes	Yes	**Yes**
Non-free choice	No	No	No	No	**Yes**
Basic Loops	Yes	Yes	Yes	Yes	**Yes**
Short Loops	Yes	Yes	No	No	**No**
Noise	No	Yes	Yes	No	**No**
Time	Yes	No	No	No	**No**

TABLE IV
COMPARING PROCESS MINING TOOLS

more simple in computing. This simplicity will not affect its efficiency in processing the concurrent aspect of workflow. Our approach has been implemented within the ProM framework [19], as a plug-in. ProM is an plugable environment for process mining. The ProM framework is flexible with respect to the input and output format, and is also open enough to allow for the easy reuse of code during the implementation of new process mining ideas. Figure 3 shows a screenshot of our Workflow patterns miner plug-in.

Table IV compares our plug-in to workflow mining tools representing previous studied approaches. We focus on seven aspects: **structure** of the target discovering language, **local discovery** dealing with incomplete parts of logs (opposed to global and complete logs analysis), **parallelism** (a fork path beginning with and-split and ending with and-join), **non-free choice** (NFC processes mix synchronisation and choice in one construct), **loops** (basics cyclic workflow transitions, or paths), **Short loops** (mono- or bi- activity(ies) loops), **noise** (situation where logs are incomplete or contains errors or non-representative exceptional instances), and **time** (event time stamp information used to calculate performance indicators such as waiting/synchronisation times, flow times, load/utilisation rate, etc.). There are others process mining tools (concerning for instance, Social network mining, workflow analysis frameworks), but they are out of the scope of this paper.

Our approach can be distinguished by supporting **local discovery** through a set of control flow ming rules that are characterised by a "local" workflow patterns discovery enabling **partial results** to be discovered correctly. Moreover, even if non-free choice (NFC) construct is mentioned as an example of a workflow pattern that is difficult to mine, WorkflowMiner discovers M-out-of-N-Join pattern which can be seen as a generalisation of the useful Discriminator pattern that were proven to be inherently non free-choice. None of related works can deal with such constructs. Our current work is about discovering complex patterns by using more metrics (*e.g.* entropy, periodicity, etc.) and by enriching the workflow logs. We are also interested in discovering more complex transactional characteristics of cooperative workflows [5].

REFERENCES

[1] C. Apte, E. Bibelnieks, R. Natarajan, E. Pednault, F. Tipu, D. Campbell, and B. Nelson. Segmentation-based modeling for advanced targeted marketing. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 408–413, New York, NY, USA, 2001. ACM Press.

[2] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.

[3] J. E. Cook and A. L. Wolf. Event-based detection of concurrency. In *6th ACM SIGSOFT international symposium on Foundations of software engineering*. ACM Press, 1998.

[4] W. Gaaloul, S. Bhiri, and C. Godart. Discovering workflow transactional behaviour event-based log. In *12th International Conference on Cooperative Information Systems (CoopIS'04)*, LNCS, Larnaca, Cyprus, October 25-29, 2004. Springer-Verlag.

[5] W. Gaaloul and C. Godart. Mining workflow recovery from event based logs. In *Business Process Management*, volume 3649, pages 169–185, 2005.

[6] W. Gaaloul, K. Baïna, and C. Godart. Towards mining structural workflow patterns. In Kim Viborg Andersen, John K. Debenham, and Roland Wagner, editors, *DEXA*, volume 3588 of *LNCS*, pages 24–33. Springer, 2005.

[7] W. M. P. van der Aalst. Business alignment: Using process mining as a tool for delta analysis. In *CAiSE Workshops (2)*, pages 138–145, 2004.

[8] B. Benatallah, F. Casati, and F. Toumani. Analysis and management of web service protocols. In *ER*, pages 524–541, 2004.

[9] K. Baïna, B. Benatallah, F. Casati, and F. Toumani. Model-driven web service development. In *CAiSE*, pages 290–306, 2004.

[10] R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. *Lecture Notes in Computer Science*, 1377:469–498, 1998.

[11] J. E. Cook and A. L. Wolf. Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 7(3):215–249, 1998.

[12] J. Herbst. A machine learning approach to workflow management. In *Machine Learning: ECML 2000, 11th European Conference on Machine Learning, Barcelona, Catalonia, Spain*, volume 1810, pages 183–194. Springer, May 2000.

[13] M. Sayal, F. Casati, M.C. Shan, and U. Dayal. Business process cockpit. *Proceedings of 28th International Conference on Very Large Data Bases (VLDB'02)*, pages 880–883, 2002.

[14] D. Grigori, F. Casati, M. Castellanos, U. Dayal, M. Sayal, and M-C. Shan. Business process intelligence. *Comput. Ind.*, 53(3):321–343, 2004.

[15] Wil M. P. van der Aalst and B. F. van Dongen. Discovering workflow performance models from timed logs. In *1st International Conference on Engineering and Deployment of Cooperative Information Systems*, pages 45–63. Springer-Verlag, 2002.

[16] A. J. M. M. Weijters and W. M. P. van der Aalst. Workflow mining: Discovering workflow models from event-based data. In *ECAI Workshop on Knowledge Discovery and Spatial Data*, pages 78–84, 2002.

[17] J. Herbst and D. Karagiannis. Workflow mining with inwolve. *Comput. Ind.*, 53(3):245–264, 2004.

[18] G. Schimm. Process Miner - A Tool for Mining Process Schemes from Event-Based Data. In *European Conference on Logics in AI*, pages 525–528. Springer-Verlag, 2002.

[19] Boudewijn F. van Dongen, Ana Karla A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and Wil M. P. van der Aalst. The prom framework: A new era in process mining tool support. In Gianfranco Ciardo and Philippe Darondeau, editors, *ICATPN*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer, 2005.

Task Anticipation: a quantitative analysis using workflow process simulation

Igor Steinmacher[1, 2, 3], José Valdeni de Lima[1], Elisa Hatsue M. Huzita[3]
[1]*Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS) – Brasil*
[2]*Companhia de Informática do Paraná (CELEPAR) – Brasil*
[3]*Departamento de Informática – Universidade Estadual de Maringá (UEM) – Brasil*
igor@celepar.pr.gov.br, valdeni@inf.ufrgs.br, elisa@din.uem.br

Abstract

Traditional Workflow Management Systems (WfMS) are too rigid, posing problems when cooperation and human behavior are important factors. Under these circumstances, there is a need for a relaxation of the end-begin dependency. Task anticipation can provide this, relaxing the flow of control and dataflow among tasks, allowing exchange of results among sequential tasks even before the conclusion of the former task. This anticipation promotes a better cooperation and augments the performance in terms of process execution time. This paper presents some ways to use task anticipation in WfMS, simulating its behavior and comparing its performance to traditional process execution. Simulation results show that using anticipation is statistically better than the traditional approach (presenting profits of up to 33%). Taking into account possible additional effort anticipation shows better performance when initiated before 40% of the predecessors' execution.

1. Introduction

Workflow Management Systems have attracted much attention recently because of their simple model for defining, executing and monitoring processes. However, traditional WfMSs are too rigid, posing problems when cooperation and human behavior are important factors.

Aalst and Hee [1] say that the rigid and inflexible character of the early (and some of the contemporary) products scared away many potential users. Hagen and Alonso [7] show the same point of view, saying that the inflexibility turns workflow technologies restricted to a few domains. In this sense, so many researches point out these as the main reason to the resistance of using WfMS in some domains [8], [10], [16], [17], [18].

So, it becomes evident that is necessary to provide ways to relax some restrictions imposed by the actual systems. The restriction tackled here is the task impossibility of produce, supply or use intermediate results. So, in the traditional WfMSs, tasks are black boxes. Relaxing this restriction the tasks could be triggered before the conclusion of their predecessors using other tasks intermediate results: resulting in task anticipation [5], [6]. This task anticipation is the main focus of this paper.

Task anticipation promotes a better cooperation and augments the performance in terms of process execution time. The better cooperation is related to the intermediate results exchange, allowing two or more tasks to work with a same instance of a given data while it is still being produced. The performance is directly related to the level of parallelism created among tasks, as two or more tasks can be executed in parallel even though they were initially defined to execute in a strictly sequential manner.

The goal of this work is to present a study related to flexible workflow execution, focusing on identifying mechanisms to provide task anticipation and quantifying its performance augmentation. These mechanisms were classified according to their related phase (definition/ execution time). As result, two classes of mechanisms were specified: a priori and a posteriori anticipation.

After defining the mechanisms, some simulations were carefully carried, focusing on comparing the performance of the process execution using anticipation and using traditional execution. These simulations generate results related to processes execution time, considering anticipation and some extra efforts related to the extra communication and intermediate results exchange.

The rest of the paper is organized as follows. Section 2 presents a bibliographic revision related to workflow flexibilization. Section 3 introduces the task anticipation and the possible ways to use it. Section 4 presents the simulations carried. Finally, section 5 presents conclusions.

2. Related Researches

Workflow systems flexibility is a topic explored in many current researches. Such researches objective is to make workflow management systems useful to a large number of domains. Necessary flexibility focuses on kill the extremely rigid structure found in the traditional WfMS.

This section presents the researches and approaches related to flexibility. Analyzing the proposals, we have created a classification for the flexibility mechanisms: *a priori* flexibility and *a posteriori* flexibility.

A priori flexibility represents the flexibility offered during process modeling, related to process definition time. They create ways to make process execution less restrictive through new modeling elements or techniques.

A posteriori flexibility presents ways to make WfMSs less rigid through changes in processes execution. These changes are related to dynamic modifications in process and modifications in the way that processes are executed.

2.1 *A priori* flexibility

The first approach to be presented introduces the *Coo-operator* [4], developed to enhance modeling of virtual workflow processes executed by virtual enterprises with special interest for cooperative situations. This operator does not cover all the aspect of coordination and cooperation. Creating a new modeling element increases the effort necessary to model the process, demanding from

the workflow designer an anticipated configuration of the ways the information exchange will be proceeded.

Freeflow [3] proposes using different kinds of relationships between tasks. The proposed relationships are declarative, capable to separate the tasks dependencies and user information. This separation is made through a state-based task model which distinguishes user states and system states. This approach presents new dependencies based on the control flow, but do not treat the data flow.

Raposo *et al.* [15] also presents modification on relationships between tasks to create flexible processes, separating the interdependence and coordination mechanisms. It allows the deployment of different coordination policies in the same collaborative system by only changing the coordination mechanisms. This approach allows a great number of possibilities to define the processes, but, again, increases the modeling effort because of the new dependencies offered.

The approach presented in [9], [10] proposes that the correct definition of task behavior is the base for modeling less restrictive processes and to allow dynamic modification. Then, it is presented a way to specify different kinds of control flow focusing on defining less restrictive workflows. These dependencies are defined by E-C-A (Event-Condition-Action) rules. The flexibility and the degree of freedom offered are the positive points of this approach. However, the same problem of past approaches appears here: the extra complexity inserted by the necessity of creating and using new modeling elements.

2.2 *A posteriori* flexibility

An approach with *a posteriori* flexibility is those proposed by Nickerson [14], which presents the problems that long transactions can cause. In this way an event-based workflow model is proposed. This model allows changing the course of an in-progress long task. This approach relaxes the task isolation restriction through external events. However, it was not presented a way to relax the dataflow, allowing the results exchange.

In ADEPT project [16], [17], a model that offers support for dynamic process changing is proposed. ADEPT is a conceptual workflow model which is base on graphs with a rigid formal base for syntax and semantic. Based on this model, a minimum but complete set of operations that supports changing the structure of workflow processes instances, while they are executed, allied to restrictions that guarantee the consistency and correctness of the process - $ADEPT_{flex}$ [16]. Although allowing dynamic process adjusting result in extra flexibility, the tasks are still executing in an isolated way.

Co-flow project [11] presents an approach with intelligent workflow support. Intelligent agents are qualified to adapt and extend preplanned processes for specific situations, without influencing other cases based on the same definition. Co-flow is another approach that presents solution for the problem of the dynamic process changing. Using this approach would demand high cost and the tasks would continue as blackboxes, executing in isolated way.

Coo-flow [6] is another approach that presents *a posteriori* flexibility, more specifically it proposes the task anticipation. It presents a workflow-based technology to coordinate process with creative interactions. To allow task anticipation in execution time, two new states were proposed and an intermediate results exchanging protocol was used. This approach was the base of our work.

3. Task Anticipation

Anticipation means that a task is allowed to start its execution before the expected time [18]. It is possible to anticipate the start of a task because of the conclusion of its predecessors before the predefined time. But, it is not this kind of anticipation that is presented here. The concept of anticipation used here is starting a task without the conclusion of its direct predecessors. Evidently, this start happens during the execution of the predecessors, and its conclusion depends on the conclusion of the predecessors.

The advantage on using anticipation is the increasing of cooperation between tasks and the decreasing of the process total execution time. This occurs because of the parallelism created by intermediate results exchanging between tasks modeled as strictly sequential tasks. This parallelism is illustrated on Figure 1. Figure 1(a) presents the traditional process flow, conditioning the start of a task to the end of its predecessor. Figure 1(b) presents the same process but using the concept of task anticipation (at “revision” task).

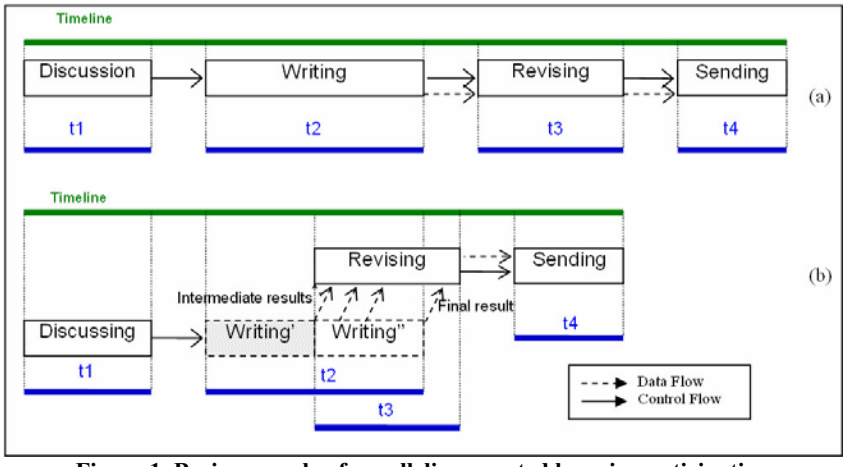


Figure 1. Basic example of parallelism created by using anticipation

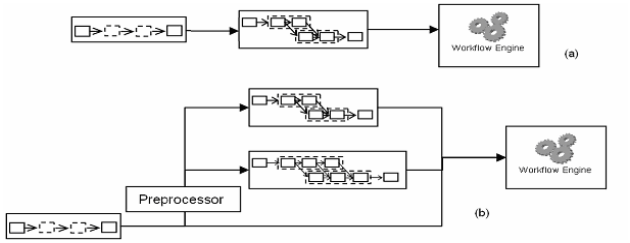


Figure 2. *A priori* anticipation management: manual approach (a) and automatic approach (b).

In the current paper, we separate the anticipation management in two different classes: *a priori* management and *a posteriori* management. Through *a priori* anticipation management, changes are made on process definition time. *A posteriori* anticipation demands some changes on workflow engine, in order to relax task isolation and atomicity restrictions. This approach changes the way that the workflow process is executed.

3.1 *A priori* management

Providing anticipation *a priori* is intuitive. It is called *a priori* because the anticipation is fully managed in process definition time, before the process execution

Anticipation is provided by *a priori* management through modeling the tasks allowed to exchange intermediate results in a lower granularity level. This can be made without using any different element or concept to model an anticipatable task. An anticipatable task and its predecessors are "broken" into smaller tasks. So they become partly sequential and partly parallel. *A priori* management can be provided in two ways:

Manual: Workflow designers have to identify and design all the tasks in the correct granularity level at definition time. An anticipatable task must have its predecessors divided into subtasks the anticipation. Process definition must contain the exact number of intermediate results that can be sent to the anticipatable tasks.

Automatic: Workflow designer just has to identify which tasks are allowed to be anticipated. The changes are inserted into the process definition when it is instantiated, through a preprocessor. If the process is defined in XPDL or BPEL, it is possible to use a XSLT preceding workflow engine, being possible to choose the number of interactions between anticipatable task and its predecessors.

Using manual approach it is not necessary to insert any different information in traditional process definition model. However, the extra effort required is so big, because it is necessary to analyze each task and the number of necessary (and required) results exchange.

Using the automatic approach, the necessary effort is lower than using manual approach. But it is necessary to include a new attribute into the process definition model. This can exclude some existent definition tools. Another turn off of automatic approach is the necessity of creating a preprocessor module before the workflow engine. But this preprocessor has the advantage of transforming one process definition into many different instances, according to the number of results exchanges defined for each anticipatable task. Figure 2 illustrates how *a priori* anticipation works in both cases: manual management (Figure 2(a)) and automatic management (Figure 2(b)).

3.2 *A posteriori* management

A posteriori anticipation management tries to make it easier to model processes with anticipation. In this case task anticipation is controlled at execution time, by workflow engine. To support the anticipation in execution time it is necessary to change the workflow engine. The changes are made in order to allow the tasks to start their execution with a different pre-conditions set. In the case of anticipation, allowing the tasks to be triggered before the conclusion of their predecessors.

Two different ways to provide this kind of management had been identified and are presented in the current paper. These kinds of anticipation will be presented follow.

A posteriori anticipation cannot be provided fully by workflow execution engine. It is necessary that workflow designers identify the tasks allowed to anticipate (may be through an extra task attribute). In other words, it is necessary to the workflow engine to know which tasks can be anticipated and those which can not be.

3.2.1 Modifying tasks state diagram

In this kind of *a posteriori* anticipation management the tasks state diagram is changed. Two new states are created, representing the tasks which have all their anticipation conditions fulfilled (ready to anticipate state) and the tasks that are executing in anticipated way (anticipating state). We have used Grigori approach [5] and WfMC state diagram as base to create the state diagram proposed here. This diagram is presented in Figure 3.

When a process is instantiated all its tasks enter the initial state, in this case the "notReady" state. This state is divided into two sub-states: anticipatable and notAnticipatable. A task is anticipatable if it is identified as a task that is allowed to be anticipated. If the task has not been identified as so, it becomes notAnticipatable. If the workflow process definition was designed with a tool that does not allow the identification, this can be made after the instantiation, in execution time. The bidirectional transition between these sub-states represents this possibility. So, the project manager can define anticipatable tasks in a flexible way, allowing or prohibiting a task to be anticipated according to the instance context. A task becomes ready when its anticipation preconditions are fulfilled. To represent these preconditions the states anticipatable and notAnticipatable have been divided into sub-states. These states concern to:

Anticipation: a task is identified as anticipatable or not (anticipatable/ notAnticipatable);

Dataflow: Input data of the task is available as final result (dataOK), as intermediate result (dataInterm) or are not available yet (dataNotOK);

Control flow: the trigger dependency of the task was fulfilled (controlOK), satisfies the *anticipation condition*[1] (controlAnt) or was not satisfied (controlNotOK)

So, a task becomes ready.toExecute when its predecessors are concluded and its input data are final results. And it becomes ready.toAnticipate when it is anticipatable, its predecessors are running and its input data

[1] The anticipation condition used here is: a task may only be anticipated after its predecessors begin and may only be concluded after its predecessors' conclusion. This kind of dependency is called Medium Dependency [18].

are intermediate results. In both cases the tasks are sent to the worklist of agents able to execute them.

When required by an agent the task enters the state running. If the task required was ready.toAnticipate it becomes anticipating and if it was ready.toExecute it becomes an executing task. When an anticipating task has all its conditions fulfilled (i.e., its predecessors are concluded), it becomes an executing task. The only transition to concluded state has the executing state as origin, so, only the tasks whose predecessors are concluded can be concluded, obeying the *Medium Dependency* [18].

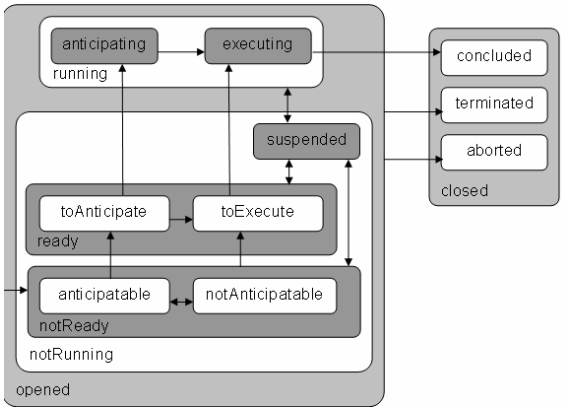


Figure 3. Modified state diagram for tasks, with new states to provide anticipation managed *a posteriori*

3.2.2 Using dynamic process changing

This class of a posteriori anticipation management uses the approach proposed in [17]. Here we have identified that the *task insertion* operation is useful and can be used to provide anticipation in execution time.

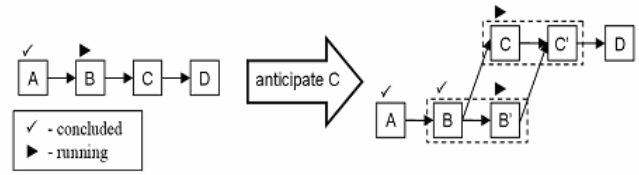


Figure 4. *A posteriori* anticipation management using dynamic process changing.

Using this approach, the workflow engine is the responsible for allowing the tasks to anticipate their start when the anticipation conditions are fulfilled. When task conditions are satisfied and some agent start its anticipation, workflow engine "break" the anticipated task and its predecessors. This operation generates a parallel flow between sequential tasks during process execution.

Figure 4 illustrates the *a posteriori* anticipation using dynamic process changing. Task *C* requested anticipation during the execution of task *B*. So, the system immediately inserted new tasks (*B'* and *C'*) representing the final part of *B* and the initial part of *C*. When *B* is concluded, *C* is also marked as concluded, and *C'* is triggered.

4. Simulating Anticipation

Computer simulation is an efficient way to study the processes and preview their performance [12], [19]. Through simulation it is possible to analyze some aspects of a process in a quantitative and dynamic way [19]. Aiello [2] said that through simulations the developers can execute a process in different scenarios while varying values and input distributions. At the end of simulations, tools may provide valuable information about the process.

Focusing the performance analysis of using task anticipation, some simulations were carefully carried. The results show, in a quantitative way, a comparative between the traditional execution of a process and an execution using anticipation. The results are based on two data analysis: the mean process execution time values and a statistical analysis of the results based on the pairwise t-test (testing which values are considered statistically different from traditional execution values with 95% confidence).

4.1 Test Bed and Scenarios

The test bed used in our simulations is a basic process of creating and submitting a scientific paper. We used this process because it sounds familiar and contains the desirable characteristics to use anticipation: it is a predominantly intellectual process; there is data exchange between tasks. The process used as test bed is presented in Figure 5. In the Figure shaded tasks can be anticipated; values over tasks are the minimum and maximum execution time for each task (in hours).

The results obtained are based on analysis of total process execution time varying some values:

Task execution time: The time expended by a task to be concluded. When set to variable, the values are generated through a Beta PERT distribution;

Anticipated start: Represents the moment when anticipated tasks are started. This value is relative to the task predecessors' execution. For example, if the anticipated start is 0.5, then, the anticipatable tasks will be started after 50% of the predecessors' task execution time. When set to variable, the values are generated through a linear distribution in the interval [0.1, 1];

Additional effort: This variable simulates the human behavior, representing the effort relative to anticipation. This effort may include extra communication, results exchanging, new results waiting and conflicts resolution.

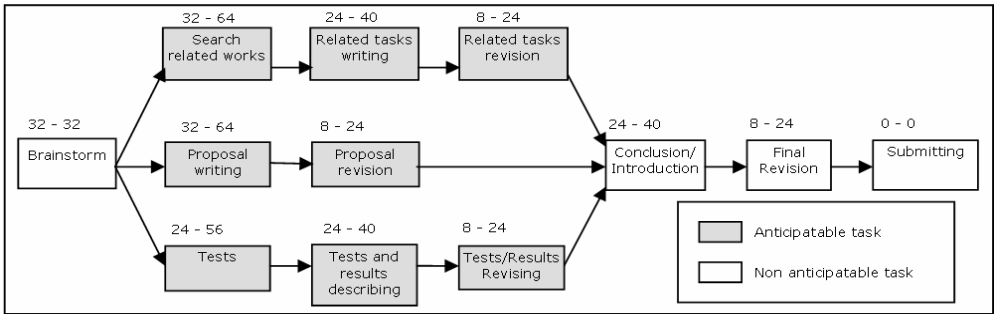


Figure 5. Test bed used in simulations: a basic paper writing and submitting process

Two simulation scenarios were created. For each simulation scenario 500 cases were generated. The use of randomized values tries to express human behavior. The scenarios and their results are presented following.

First Scenario. The first simulation scenario confronts the total process execution time using traditional execution and anticipation. In this scenario the cases were generated varying the *task execution time* and applied to 10 different values of *anticipated started* (from 0.1 to 1). In this scenario we do not consider any *additional effort*.

The results of this scenario are presented in Table 1 and show that task anticipation makes the process execution time decrease, if compared to traditional execution. The numbers show that the better results appear when the anticipation is started sooner.

Table 1. Process execution time varying anticipation start

Anticipation Started	*Mean Process Execution Time*
10%	119,89
20%	123,32
30%	126,87
40%	132,68
50%	140,52
60%	149,34
70%	156,28
80%	164,46
90%	171,83
Randomized	147,38
Traditional Execution	178,67

Comparing anticipation started at 10% of predecessors' task execution time and traditional execution, there is a process execution time decrease close to 30%. Comparing an intermediate value of *anticipation started* (50%) and the traditional execution, the decrease is next to 20%. When the *anticipation started* values are randomized, the profit is close to that obtained with *anticipation started* at 50% (decreasing the process execution time 20%).

Second Scenario. The second scenario uses the *additional effort* variable. The cases were the same used in the first scenario. But in this scenario we simulate the cases with each pair of *additional effort* (from 0 to 1) and *anticipated start* (from 0.1 to 1) values.

This scenario results are presented in Table 2. Table 3 presents the results combining Table 2 values and pairwise t-test with 95% confidence. In this table there are three possible values: *same*, *no* and *yes*. *Same* indicates that the value achieved by traditional execution and the compared value are statistically not different; *yes* indicates that the compared value represents better performance than traditional execution, and *no* indicates that it presents worse performance than traditional execution.

Obtained results focus in producing a behavior pattern linking necessary effort to anticipate tasks and anticipation start. With the data and graphic it is possible to verify the values where task anticipation is advantageous (in terms of execution time). Confronting these values with real cases historical data, manager can free or block the anticipation of a task, for example.

Analyzing the numbers, anticipation best results occur when the tasks are anticipated at the beginning of the predecessors (10% and 20%), when the advantage appears in all simulated cases. In the worst case the additional effort is 100% (duplicating execution time) and the obtained result is better than using traditional execution.

When the anticipation is started between 30% and 40% of predecessors' execution time, the advantage appears to additional effort values lesser than 80%. For anticipation starting at 50%, the advantage appears for additional effort from 0% to 50%. Setting the anticipation start after 60% of execution of predecessors the advantage only appears when the effort is not greater than 20%.

It is explicit that the delayed begin of anticipation is not advantageous for the process. When the tasks are anticipated after 80% of their predecessors, results are better than traditional execution only when there is no additional effort to anticipate.

Table 2. Process execution time varying the anticipation start and the additional effort

Effort / Start	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%	Randomized
10%	119,89	124,39	130,57	135,64	140,53	144,59	148,66	155,46	159,16	165,10	169,15	148,51
20%	123,32	129,08	133,35	138,30	143,71	147,91	152,30	158,20	163,21	168,09	173,62	153,46
30%	126,87	133,52	138,03	142,84	148,66	155,64	158,14	164,41	168,41	173,51	178,27	158,12
40%	132,68	138,96	144,53	150,04	155,22	159,49	165,16	170,98	175,81	180,91	186,94	164,39
50%	140,52	147,92	153,54	159,37	164,50	169,78	175,84	181,56	187,01	193,33	198,69	173,70
60%	149,34	155,72	161,91	168,59	174,51	180,04	186,61	193,75	199,09	205,29	210,97	183,26
70%	156,28	163,44	170,61	178,14	184,36	190,10	196,48	204,40	210,75	216,78	224,06	194,44
80%	164,46	172,08	179,63	186,13	192,86	199,50	207,23	215,86	221,85	230,54	236,68	204,09
90%	171,83	181,21	188,67	195,90	203,97	211,04	219,52	225,84	233,34	240,86	247,33	215,29
Randomized	147,38	153,27	158,86	166,02	172,11	176,74	183,06	189,04	197,40	203,00	207,73	179,20
Traditional Execution	178,67	178,67	178,67	178,67	178,67	178,67	178,67	178,67	178,67	178,67	178,67	178,67

Table 3. Better than Traditional Execution with 95% confidence

Effort / Start	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%	Randomized
10%	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
20%	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Same	Yes
30%	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Same	Same	Yes
40%	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Same	Same	Same	Same	Yes
50%	Yes	Yes	Yes	Yes	Yes	Yes	Same	Same	Same	No	No	Same
60%	Yes	Yes	Yes	Yes	Same	Same	Same	No	No	No	No	Same
70%	Yes	Yes	Same	Same	Same	No	No	No	No	No	No	No
80%	Yes	Same	Same	Same	No	No	No	No	No	No	No	No
90%	Yes	Same	No	No	No	No	No	No	No	No	No	No
Randomized	Yes	Yes	Yes	Yes	Same	Same	Same	No	No	No	No	Same

When randomizing start values, the anticipation is better with additional effort until 40%. When randomizing both variables the result was good, presenting a value similar to the mean execution time obtained in traditional execution. We said it is a good result because it shows that a process execution using anticipation with additional effort has the same performance as the same process using traditional execution flow. So, we can have the cooperation advantage introduced by anticipation without losing more time than using traditional execution. Increasing cooperation, we may have better quality products. This motivates us to apply anticipation to real cases to evaluate possible cooperation increasing and final product's quality.

5. Conclusions

The goal of the current paper is to present the task anticipation as a way of relax some restrictions of current Workflows Management Systems, focusing on its performance analysis. Anticipation can increase the performance of cooperative processes through the parallelism created among tasks. To evaluate the performance of task anticipation confronting traditional process execution some measurements had been carried. The results of the simulations had allowed identifying to significant profits in favor of the use of the anticipation and behavior patterns that can be used in future tests.

The results demonstrate that it is necessary to start the anticipation before the half of the execution of tasks predecessors to anticipation become more efficient. In our test bed, anticipation presented profits of up to 33% (tasks starting anticipation with 10% of their predecessors' execution time). In the average case, the profits with relation to the traditional execution had been close to 18%. Also with additional effort, anticipation had been advantageous in most part of the cases. When the anticipation was initiated before 40% it presented better performance. Anticipating tasks next to the end to its predecessors can bring losses, if the necessary extra effort for such anticipation exists. For anticipation started after 70% of the beginning of the predecessors these losses appear when the effort is superior 30%.

Besides the profits, simulations had presented some behavior patterns linking anticipation start, additional effort and process execution time. These patterns can be used to enable/disable anticipation according to historical analysis of additional effort inserted by anticipation. For example, the anticipation of a task can be disabled if its predecessors already exceeded 60% of its mean execution time.

When using randomized values for *additional effort* and *anticipation start* the mean process execution time was the same as using traditional execution. So, we can have more cooperation and better final products without losing time. In our simulations the possible profits related to final product quality and to cooperation increasing had not been analyzed. To make it possible it is necessary to apply anticipation to real cases, comparing results of processes with and without using anticipation approach. Also effort and time of execution had not been analyzed costs relating.

Other results of this research: the workflow simulator used to carry the simulations presented here, a workflow editor and a workflow engine implementing anticipation concepts.

6. References

[1] Aalst, W.; Hee, K. Workflow Management: Models, Methods and Systems. The MIT Press – Massachussets Institute of Technology, Massachussets, 2002.

[2] Aiello, R. Workflow Performance Evaluation, (PhD. Thesis) – Università di Salerno, Salerno, 2004.

[3] Dourish, P.; Holmes, J.; Maclean, A.; Zbyslaw, A;.Marqvarsen, P. Freeflow: Mediating Between Representation and Action in Workflow Systems. In: ACM conference on computer supported cooperative work, 1996, Boston, EUA. Nova York: p. 190-198.

[4] Godart, C.; Perrin, O.; Skaf, H. Coo: a Workflow Operator to Improve the Cooperation Modeling in Virtual Processes. In: International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises, Australia, 1999.

[5] Grigori, D. Eléments de flexibilité des systèmes de workflow pour la définition et l'exécution de procédés coopératifs. 2001. (PhD Thesis) – Université Henri Poincaré, Nancy 1. 2001.

[6] Grigori, D.; Charoy, F.; Godart, C. Coo-Flow: A process technology to support cooperative processes. International Journal of Software Engineering and Knowledge Engineering, Jan 2004.

[7] Hagen, C.; Alonso, G. Beyond the black box: Event-based inter-process communication in process support systems. In: International Conference on distributed computing systems, 1999

[8] Joeris, G.; Herzog, O. Towards Object-Oriented Modeling and Enacting of Processes. TZI-Report 07/98, Universidade de Bremen, 1998.

[9] Joeris, G. Defining Flexible Workflow Execution Behaviors. In: Enterprise-Wide and Crossenterprise Workflow Management: Concepts, Systems, Applications, Germany. 1999.

[10] Joeris, G.; Herzog, O. Towards Flexible and High-Level Modeling and Enacting of Processes. In: international conference on advanced information systems engineering, 11., Heidelberg, Germany, 1999. p. 88-102.

[11] Kramler, G.; Retschitzegger, W. Towards Intelligent Support of Workflow. In: Americas Conference on Information Systems, Long Beach. 2000. p. 581-585.

[12] Laguna, M.; Marklund, J. Business Process Modeling, Simulation, and Design. Prentice Hall, 2004.

[13] Lima, J.; Quint, V.; Layaida, N.; Edelweiss, N.; Zeve, C.; Pinheiro, M.; Telecken, T. The Conception of Cooperative Environment for Editing Multimedia Documents with Workflow Technology (CEMT). In: PROTEM-CC, 2001.

[14] Nickerson, J.V. Event-based Workflow and the Management Interface. In: Hawaii International Conference on System Sciences (HICSS), 36., 2003, Big Island, Hawaii. Proceedings… Washington: IEEE Computer Society, 2003.

[15] Raposo, A.; Magalhães, L.; Ricarte, I.; Fuks, H. Coordination of Collaborative Activities: A Framework for Definition of Tasks Interdependencies. In: International Workshop on Groupware (CRIWG), 7., 2001, Germany. 2001.

[16] Reichert, M.; Dadam, P. Adeptflex – Supporting Dynamic Changes of Workflow Without Loosing Control. Journal of Intelligent Information System: Special Issue on Workflow Management Systems, v.10, n.2, p.93-129, 1998.

[17] Reichert, M.; Rinderle, S.; Dadam, P. ADEPT Workflow Management System: Flexible Support for Enterprise-Wide Business Processes. In: Business process management international conference (BPM), 2003, Netherlands. 2003

[18] Steinmacher, I. Estudo e Análise de Desempenho da Antecipação de Tarefas em Sistemas Gerenciadores de Workflow.(Master Thesis), Universidade Federal do Rio Grande do Sul, Brazil. 2005.

[19] Tramontina, G., Wainer, J., Ellis, C. Applying scheduling techniques to minimize the number of late jobs in workflow systems. In Proceedings of the 2004 ACM Symposium on Applied Computing, USA, 2004. ACM Press.

AN.P2P – an Active Peer-to-peer System

[1]Chi-Hung Chi, [2]Mu Su, [1]Lin Liu, [1]HongGuang Wang
[1]Tsinghua University, Beijing, China
[2]National University of Singapore, Singapore
Email: chichihung@mail.tsinghua.edu.cn

ABSTRACT. This paper proposes an active P2P architecture - AN.P2P - to implement effective content delivery on P2P networks for heterogeneous service requirements. Firstly, the AN.P2P acts as a middleware tier between upper applications and underlying P2P networking substrates. It enables the peers to perform value-added functions in a transparent manner to both the upper applications and underlying P2P substrate. Secondly, the AN.P2P uses mobile applications to implement the content adaptation modules so that any peer can adopt new functions dynamically by downloading corresponding mobile applications. Thirdly, AN.P2P can replicate the original objects within the network. The recipient peers can reuse the object replicas and the downloaded applications to serve other peers with diverse requirements. Our simulation shows that AN.P2P can achieve reasonable overall performance.

1. INTRODUCTION

Recently, peer-to-peer (P2P) applications have witnessed more heterogeneous service requirements due to the emergence of diverse devices and network connections. Without system support for content adaptation and service customization, conventional P2P applications [1][2][3][4] cannot address these requirements effectively.

A naive solution to provide such value-added functions within P2P is the "owner-side adaptation". It allows the user to install the relevant applications on the peer. Upon receiving a query, the peer may invoke these applications to transform the content object authored by current peer. Finally, the generated content presentation will be sent to other peers. However, the main drawback of this method is that the fully adapted content presentation has significantly reduced reusability for other peers with diverse presentation requirements. Hence, the owner-side adaptation method may compromise the peer-sharing benefit and the overall system performance. Moreover, other peers that attempt to transform the retrieved objects may perform inappropriate operations without necessary directives from the content owner.

This paper proposes an active P2P architecture - AN.P2P - to implement efficient P2P content delivery for heterogeneous presentation requirements. In the paper, we use "content presentation" as a general term to represent the way of delivering or presenting a content object. A presentation may be a multimedia data format, an individual literal language, or a transmission method like downloading or streaming.

Our key idea came from the observation that despite of the heterogeneous requirements for content presentations, the processes to generate the presentations should be homogeneous. It means that various content presentations are generated from the same original object and adaptation workflow. Hence, the AN.P2P enable a peer to replicate the original content object with an adaptation workflow to other peers. The recipient peer can thus reuse the original content object to generate different content presentations for other peers with different presentation requirements. Moreover, we leverage on mobile agent techniques to implement the tasks in the adaptation workflow, so that any peer can adopt new applications by downloading them on the fly.

2. SYSTEM ARCHITECTURE

2.1. Components

The general architecture of AN.P2P is shown in Fig. 1, where the AN.P2P acts as a middleware tier between the P2P substrate and the file sharing utility. The file sharing utility interacts with the end user, including presenting the searching GUI, displaying the searching result, and maintaining the local file-sharing directory where the user put the shared files. The AN.P2P middleware receives the query commands from the software utility and relays it to the P2P substrate, which will send query messages into the P2P network. Similarly, the AN.P2P middleware also receives the reply from the P2P substrate and relays it back to the software utility. Moreover, the AN.P2P middleware can perform other tasks such as content adaptation, cache lookup and content replication. There are three storage components in the AN.P2P middleware, which are the User Profile, the AN.P2P Cache and the ANlet Pool.

Firstly, the User Profile stores the presentation requirement of the user and the end device. The profile will be used by the AN.P2P mobile applications to perform suitable content adaptation. Typical profile entries include the preference of the user, the device screen size, computation capability, network speed and

so on. We intend to leverage on existing techniques, such as CC/PP, to specify the user profile.

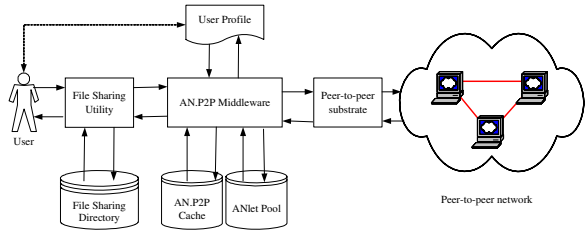


Fig. 1 AN.P2P System Components

Secondly, the AN.P2P Cache is used to store the retrieved content objects. The cache and the file-sharing directory are different units. The file-sharing directory is controlled by the file-sharing software to store the shared and retrieved files. However, the AN.P2P cache is controlled by AN.P2P middleware, who can store objects temporarily and evict them according to its own policy.

Thirdly, the ANlet Pool is used to store the downloaded mobile applications. In addition, the user can also store frequently used applications in the ANlet Pool. Hence, when the AN.P2P middleware requires such a common application it can reuse a local one instead of downloading remotely. Furthermore, AN.P2P uses a standard Java interface, called ANlet, to invoke the applications. In particular, an important ANlet callback function is given in equation (1).

AppNetContent modResponse (queryMsg msg, (1)
AppNetContent content);

This function has two input parameters, the query message and the content object. The query message contains identifier of queried content and the user profile, while the AppNetContent contains the retrieved content object with its attributes. The output of the function is the transformed object together with its new attributes.

2.2. Workflow

In AN.P2P, the content owner can associate the content object with a piece of adaptation workflow composed by multiple tasks. The usage of workflow structure can facilitate the composition of the whole content adaptation process with standard application modules. Moreover, decomposing workflow into tasks also facilitates the progressive deployment of applications on demand.

In particular, an AN.P2P workflow is composed of sub-units, named segment. Each segment contains a rule, a specific task, an input object, and an output object. Each task can be instantiated using an ANlet module, such as an image transcoder, a watermarking program, etc. Each segment also gives a location from which the ANlet can be fetched. When the condition of the rule is satisfied, the specified task will be executed. Otherwise, if the rule is not satisfied the task should not be executed. A workflow concatenates segments sequentially. If feeding the original content object to a workflow, it can transform the object through multiple tasks and generate the final content presentation, while the objects transmitted between segments are intermediate objects. Fig. 2 presents an example workflow, where the squares stand for the original, intermediate and final objects, the circles for segments, and the arrows for links between segments. The AN.P2P workflow is a rather simple data flow compared to other workflow specifications like BPEL, because we intended to make it a loosely coupled structure while restricting complex logic within specific applications implementation.

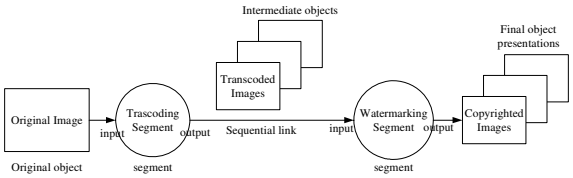


Fig. 2 Example AN.P2P Workflow

2.3. Operation Schemes

Depending on whether the P2P substrate supports proactive object placement between nodes, the AN.P2P defines two operation schemes, which are the caching scheme and the replication scheme as shown in Fig. 3 and 4 respectively. In this paper, we call the peer publishing the original content as the home peer (e.g. P_3). On unstructured P2P, the home peer is the node resided by the content owner; on structured P2P, the home peer is determined by the DHT hashing function. In addition, we call the peer issuing the query as the requesting peer (e.g. P_0), all peers forwarding the query as intermediate peers (e.g. P_1 and P_2), and the peer serving the query as the target peer. Since any peer with a replica of the queried object can serve the requesting peer, the target peer can be the home peer or an intermediate peer.

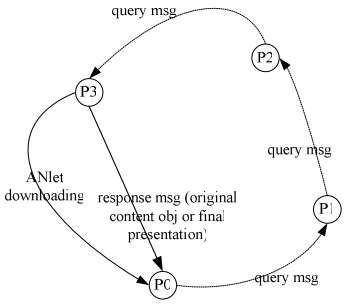


Fig. 3 Caching Scheme

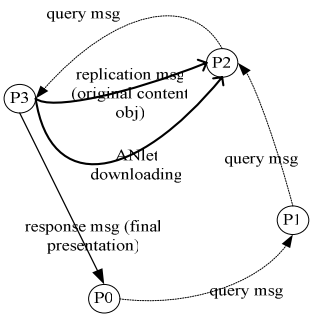


Fig. 4 Replication Scheme

2.3.1. Caching Scheme

The caching scheme doesn't require proactive object placement between nodes so that it can be applied upon any P2P substrate. To author a piece of content, the content owner publishes the original content object and its associated workflow on the home peer. On the other side, when a client requests the content, the requesting peer will encapsulate content identifier and the user profile into a query message, which will be sent through the P2P routing substrate.

When an intermediate peer receives a query message, it will look up local cache for a usable item. We assume that only the home peer has the content object at the beginning. Hence the query message will be forwarded to peers till it reaches the home peer. After receiving the query message for a content published by it, the home peer may behavior in either of two ways. In the first way, it generates a particular content presentation and send the result to the requesting peer. In specific, the peer needs to load the ANlet modules according to the workflow specification and then feed the original content object to the ANlets to generate the final content presentation for the user. However in the second way, the home peer sends the original content object with its workflow to the requesting peer directly. The determination to choose either way is made by particular deployment strategy.

Upon receiving a response message, the requesting peer needs to check the message to see if there is an associated workflow. If no workflow, the response object is deemed as the final content presentation and will be passed to the file-sharing utility. Otherwise, if there is a workflow, the requesting peer needs to execute the tasks in the workflow before passing the result to the client. In specific, the peer first instantiates the workflow tasks by loading applications from its ANlet pool. If there is an ANlet miss, the peer will download an ANlet using the location given by the workflow. After that, the peer feeds the response object to ANlets to generate the final presentation according to the user profile.

In addition, both the response message and the downloaded ANlets will be stored by the requesting peer into the AN.P2P Cache and ANlet Pool respectively. If this peer receives a new query for a cached object, it can serve the requesting peer directly. Similar to the home peer, this peer may send the content object and its workflow to the new requesting peer, or load applications to generate the content presentation and reply it the new requesting peer.

2.3.2. Replication Scheme

The replication scheme is applied to networks supporting proactive object replication. It is similar to the caching scheme in content authoring and request generation. When receiving a query message, a peer will lookups its local cache for a usable object. If there is a hit, the peer will serve the requesting peer. Otherwise, if it is a miss, the query message will be forwarded.

Different from the caching scheme, when a target peer receives a query message, it will generate a content presentation for the client and reply a response message . Moreover, the target peer will also replicate the original content object and its workflow to other intermediate peer. The recipient peer will store the replication message in the cache. When the peer receives a new query message for a cached content object, it will the serve the new requesting peer directly.

Compared to the caching scheme, the replication scheme populates object using replication messages, which may cause some overhead. However, we will explain that the replication scheme is efficient to improve the system performance greatly on structured P2P networks.

4. IMPLEMENTATION

We have implemented an AN.P2P prototype based on the Pastry structured substrate [5] and the common overlay interface [6], as shown in Fig 5.. The AN.P2P acts as a middleware tier between the upper software utility and the underlying Pastry substrate. A trace generator is also designed to simulate users to issue the object insertion and query commands according to the input trace file. The system performance is measured by the detector. We also implemented a workflow to provide the digital right management (DRM) in a P2P network with heterogeneous user requirements. Recently, DRM has become an urgent requirement to prevent the rampant piracy in P2P media-sharing applications. Iwata and Abe [7] proposed several P2P based DRM models. In general, when authoring content, the media object will be encapsulated into a secured container. The secured object is distributed in the network and any peer can

download it freely. In order to render the content, the recipient's media player needs to retrieve a license that supplies the key to disclose the secured object.

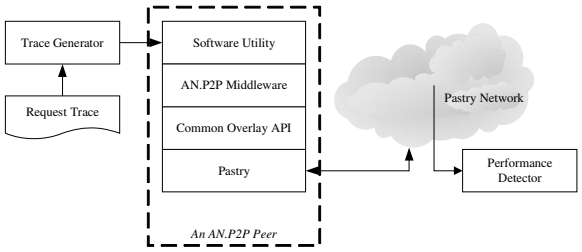


Fig. 5 AN.P2P Implementation

Our AN.P2P DRM workflow is shown in Fig. 2, the workflow involves two ANlets. Each of them contains the key to disclose the associated secured object, transform it, and write it back securely using the key. The first ANlet is a media trimmer that transforms original media object to a thinner version according to the type of the client device. The second ANlet watermarks the object using the client certificate. After receiving the response object, the recipient media player needs to obtain the license and render the media in conventional ways.

5. RELATED WORK

There have been many P2P file sharing systems in use or under development (such as Napster [1], Gnutella [2], and KaZaA [3]). Based on the fundamental P2P file-sharing systems, recent researchers proposed many augmented application and studies. Top-K [8] and Squirrel [9] targeted to implement the P2P based community storage. Waldvogel [10] designed a replica enumeration method allowed for controlled replication. Gopalakrishnan [11] proposed the LAR, a system neutral replication protocol.

Different to above systems, our study aimed to implement an active P2P system to share content with multiple presentations for heterogeneous client requirements. The AN.P2P enables the content object distribution with its mobile applications that can adapt the content presentation for different peer nodes. Moreover, it uses workflow to decompose the adaptation process into multiple application modules, so that the progressive application deployment can be achieved. All these make the AN.P2P a feasible mechanism to provide pervasive content delivery in P2P networks.

6. CONCLUSION

This paper proposed an active P2P architecture - AN.P2P - to implement pervasive P2P content delivery for the heterogeneous requirements. The AN.P2P enables the peer nodes to populate the original content objects and its workflow within the network. The recipient peers can reuse the original content objects and the downloaded mobile applications to serve other peers with diverse requirements. Our implementation shows that AN.P2P could be a promising active P2P architecture to provide pervasive content delivery in heterogeneous environment.

ACKNOWLEDGEMENT

This research is supported by the funding 2004CB719400 of China.

REFERENCES

[1] Napster. http://www.napster.com.
[2] Gnutella, http://www.gnutella.com.
[3] KaZaA, http://www.kazaa.com.
[4] A. Rowstron, P. Druschel, "Storage Management and Caching in PAST, a Large-scale, Persistent Peer-to-peer Storage Utility", Proceedings of ACM Symposium on Operating Systems Principles, Nov. 2001.
[5] A. Rowstron, P. Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-scale Peer-to-peer Systems", Proceedings of the 18th IFIP/ACM International Conference of Distributed Systems Platforms, Nov. 2001.
[6] F. Dabek, B. Zhao, P. Druschel, I. Stoica, "Towards a Common API for Structured Peer-to-Peer Overlays", Proceedings of 2nd International Workshop on Peer-to-Peer Systems, 2003.
[7] T. Iwata, T. Abe, K. Ueda, H. Sunaga, "A DRM System Suitable for P2P Content Delivery and the Study on its Implementation", Proceeding of the 9th Asia-Pacific Conf. on Comm., Vol.2, 21-24, pp.806-811, 2003.
[8] J. Kangasharju, K. W. Ross, D. A. Turner, "Adaptive Content Management in Structured P2P Communities", Proceedings of 21st ACM Symposium on Principles of Distributed Computing, 2002.
[9] S. Iyer, A. Rowstron, P. Drusche, "Squirrel: A Decentralized Peer-to-peer Web Cache", Proceedings of the 21st ACM Symposium on Principles of Distributed Computing, July 2002.
[10] M. Waldvogel, P. Hurley, D. Bauer, "Dynamic Replica Management in Distributed Hash Tables", IBM Research Report, July 2003.
[11] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, P. Kelenher, "Adaptive Replication in Peer-to-Peer Systems", Proceedings of 24th Inter. Conf. on Distributed Computing System, 2004.

Proceedings

International Workshop on Applications of Artificial Intelligence in Energy Production and Environmental Systems Engineering (AAIEPESE 2006)

Editor

Christine W. Chan, University of Regina, Canada

Artificial Intelligence and Environmental Systems Engineering

Ni-Bin Chang
Department of Civil and Environmental Engineering
University of Central Florida, Orlando, FL, USA

Abstract: The general reality of paradigm collision between artificial intelligence and environmental systems engineering has been ignored before 1990s although both of them bear quite the same nature in theory and methodology. The synergy between these two disciplines may enable us to address extended concerns of environmental engineering by enhancing understanding of issues and problems related to sustainable development in the future. This paper tends to bridge the gap and combine both regimes into a unified framework so as to enlarge their application potentials.

I. INTRODUCTION

Artificial intelligence (AI) is the scientific understanding of the mechanisms underlying thought and intelligent behavior and their embodiment in machines [1]. Artificial intelligence concepts like search, planning, learning, natural language, and so on have mature theoretical underpinnings and extensive practical histories of applications. It has promoted the quality of environmental engineering to some extent in the past decades albeit such progress is not ubiquitous. From theoretical point of view, the rational being applied may include logic, fuzzy set, systems engineering, informatics, cybernetics, repository, ontology, and semantic web. Some research fields which are quite popular today include expert system, data and knowledge engineering (DKE), pattern recognition, intelligent decision support system, AI languages, and programming techniques. The tools that can be organized to support all levels of AI applications cover knowledge acquisition (KA), artificial life, computing intelligence, distributed artificial intelligence, multi-agent system (MAS), speech and natural language interfaces, intelligent agent, robotics, statistical learning, machine learning, intelligent systems engineering and design methodologies, fuzzy inference and control, knowledge and information management, organizational memory knowledge systems, intelligent information retrieval, distributed AI algorithms, techniques and applications, machine vision, data mining, and learning and adaptive system. In general, AI involves using a number of systems analysis algorithms which are quite similar to these being used in environmental systems engineering. They include but are not limited to neural network, evolutionary computation and algorithm, genetic algorithm, Bayesian networks and stochastic reasoning, genetic programming, evolutionary strategy, fuzzy reasoning, rough set, support vector machine (SVM), rule based reasoning, and case based reasoning.

Environmental engineering is designed to handle all reverent engineering efforts for promoting human being's life and sustain the natural environment. It aims at developing technologies and strategies for several specializations including air pollution source management, ambient air quality management, environmental noise control, water resources management, surface water and wastewater treatment, ground water management, drinking water supply, solid waste management, hazardous and medical waste management, environmental health risk assessment and risk communication, health hazard assessment, and industrial hygiene field services. Recent advances have extended the focus into new areas of environmental health engineering and occupational health sciences. The analysis of the environment in terms of physical, chemical, and biological processes and the relationships between these components and those between them and human society has been becoming essential over time. It is believed that the most effective alternative can be found if problems can be considered in their totality and overall aspects. The fragmented approach taking piecemeal into account separately becomes no longer acceptable in environmental decision making. Therefore, environmental systems engineering, being created for handling environmental systems issues by a holistic approach, has been an interdisciplinary area between environmental engineering and systems engineering since three decades ago. The analytical procedure frequently involves the preparation of an assembly of methods, procedures, or techniques united by regulated interaction to form an organized whole for systems analysis. Its works have to be done at many levels of abstraction from analyzing the basic systems science theory to management disciplines; to economic and policy aspects; and to broad integration of high-level decision science, information and communication technologies.

The needs for economic growth and environmental conservation have led to the umbrella concept of sustainability that emphasizes collaborative decision making for achieving overall balance in ecosystems and human society. It leads to develop optimal management strategies for improving environmental management at various spatial and temporal scales in dealing with different environmental systems. To search for the sustainable development strategy, the decision-making process must reconcile often diverse and conflicting viewpoints related to various socio-political, economic, environmental, scientific, and technical issues. As sustainability directly affects the human beings community as a whole, the decision making process is inherently multidisciplinary, multi-agency, and multi-stakeholder in nature. Hence, environmental management and decision-making generally have multi-objective, interactive, dynamic, and uncertain features. Complexities exist in the determination of system parameters, reflection of interactive relationships, formulation of modeling approaches, interpretation of research outputs, and implementation of recommended

policies. Many challenges exist in the application of modeling techniques to environmental management owing to their sparingly-addressable structure. In addition, most of environmental models fro systems analysis can only deal with limited spatial and temporal units in a system due to difficulties in computational requirement and data availability. What decision makers desire to know might be either detailed plans based on much finer units or just a broad justification. This could lead to incompatibility between the researchers' outputs and the users' demands, and raises the question about usefulness of the modeling solutions. This has created tremendous opportunities for interdisciplinary collaboration between AI techniques and environmental systems engineering. The following analysis presents a thorough review of all up-to-date applications and a deliberation of future perspectives in the nexus of these two paradigms.

II. LITERATURE REVIEW

Research fields of expert system, data and knowledge engineering, pattern recognition, and intelligent decision support system have been applied to tackle some environmental systems, and hence, the following discussions will focused on how these AI techniques can be effectively used to aid in systems analysis in environmental systems engineering regime.

2.1 Data and Knowledge Engineering

Data and knowledge engineering stimulates the exchange of ideas and interaction between these two related fields of interest. It is the technique applied by knowledge engineers to build intelligent systems, such as Expert Systems, Knowledge Based Systems, Knowledge based Decision Support Systems, Expert Database Systems, etc. The "Transfer View" of DKE is to apply conventional knowledge engineering techniques to transfer human knowledge into artificial intelligent systems. The "Modeling View" of DKE is to model the knowledge and problem solving techniques of the domain expert into the artificial intelligent system. Hence, knowledge representation and knowledge modeling are two main foci. Knowledge representation is a multidisciplinary subject that applies theories and techniques from three other fields: 1) logic reasoning provides the formal structure and rules of inference; 2) ontology defines the kinds of things that exist in the application domain; and 3) computation supports the applications that distinguish knowledge representation from pure philosophy. Knowledge modeling is the concept of representing information and the logic in a digitally reusable format for purpose of capturing, sharing and processing knowledge to simulate intelligence. It is designed to construct intelligent systems in which systems are viewed as comprising domain models that satisfy the data requirements of reusable problem-solving methods. Such automated tools normally concerns: 1) elucidation of new reusable problem-solving methods; 2) development of techniques for creating, editing, and maintaining domain ontologies; 3) establishing more principled mechanisms for mapping domain models to the data requirements of problem-solving methods; 4) creation of intelligent tools that aid system builders in applying our approach to construction of complex software systems; and 5) elucidation of model-based approaches to the construction of human-computer interfaces. As most human inference is a physical response to perceived physical relationships and acquaintance with the preserved profiles of those relationships, a crucial requirement for knowledge modeling is to provide sufficient support in expressivity in the ontological engineering. Some relevant issues have been investigated for groundwater contamination and remediation in which dynamic semantics or task-oriented knowledge are often represented by rules [2] or problem-solving methods [3]. The development of an ontological engineering environment is tied with knowledge representation mechanisms for modeling diverse knowledge types, consisting of the physical, semantic, or inferential properties of relationships. Besides, ontology management, evolution, alignment, and development of collaboration are also critical. Very few previous environmental applications can be found out in this field. Chan [4] firstly presents the processes of knowledge acquisition and ontology development for structuring the knowledge base for a petrochemical plant that was designed as an expert system to aid in wastewater disposal and oil recycling.

2.2 Expert System

Expert knowledge is a combination of a theoretical understanding of the problem and a collection of heuristic problem-solving rules in the domain to improve science, engineering, and management potentials. Expert systems are constructed by obtaining this knowledge from a human expert and coding it into a form that an end-user may apply it to solve similar problems using computers. This reliance on the professional knowledge of a human domain expert for solving a predefined problem is a major feature of expert systems. Recent advancements of AI-based technologies for management and control of pollution prevention, waste minimization, and hazard mitigation have shown some potential via the use of expert systems [5]. Some expert systems have been developed for managing the surface and ground water systems in 1990s. These which have been fully described in the literature include, but are not limited to, the demonstration expert system to aid in assessing groundwater contamination potential by organic chemicals [6], the capability of an expert system was built to help manage and control of a composting process [7], to prioritize the groundwater contaminant sources and remedial actions [8]-[10], to determine the probability of pesticide leaching [11], and to design an advisory system that supports designers in pollution prevention by informing them about environmental ramifications of their designs [12]. The EPAS system ranks the alternative designs based on how they rate on the five factors of technical adequacy, health and safety, environmental ramifications, regulatory requirements,

and cost. The system consists of information on the environmental, regulatory, health and safety, and cost factors of a design. Chowdury and Canter [13] reviewed thirty nine expert systems that focused on risk assessment of treatment procedures for hazardous waste sites cleanup activities. Among the thirty nine systems, nine of them summarized information with a set of illustrations, They have mainly covered the groundwater remediation areas including hazardous waste site investigations (RPI), assessment of potential risk that organic chemicals pose for ground water contamination (DEMOTOX), determination of hazardous waste site cleanup procedures (HAWAMAX), evaluation of the relative risk of hazardous waste sites to human health and the environment (DPM), prioritization of potential or actual sources of ground water contamination in wellhead protection areas (WASES), evaluation of potential risk that pesticides pose for ground water (EXPRES), assistance of promoting the accuracy, timeliness, and cost-effectiveness of field sampling, chemical analyses, and analytical data validation within the Superfund program (ESES), provision of remediation recommendations and determination of remedial action costs (CORA), and application of location, field data, size calculations and user type as primary variables to evaluate sewer system permit applications (SEPIC). The use of fuzzy set theory to aid in the inference in expert systems is definitely a fruitful area to delve into. Geng et al. [14] presented an expert system for management of petroleum-contaminated sites in which permeability, types of soil media, and states of petroleum wastes are considered as fuzzy numbers. Later on, Chen et al. [15] developed an expert system for management of petroleum-contaminated sites which supports enhanced efficiencies in decision making for engineers working on specific site remediation projects. An expert decision support system for industrial water pollution control [16], for monitoring and diagnosis of an oil production and separation facility [17], and for water quality management in river basin [18] were presented separately in early 2000s. Further, Hu et al. [19] developed a fuzzy rule based expert system, called site characterization system (SCSS), which employed methods of fuzzy knowledge representation and fuzzy reasoning to convert uncertain system inputs to fuzzy linguistic information. The system can provide decision support to users in selecting suitable remediation techniques for petroleum-contaminated sites. The development of an expert system for remediation of petroleum-contaminated sites was also reported [20].

2.3 Intelligent Decision Support System

A Decision Support System (DSS) is an interactive computer-based system or subsystem which is intended to help decision makers use communications technologies, data, documents, knowledge and/or models to identify and solve problems and aid in decision making. Five types of specific DSS have received wide attention and they include: 1) communication-driven DSS; 2) data-driven DSS; 3) document-driven DSS; 4) knowledge-driven DSS; and 5) model-driven DSS. However, advances in AI are providing additional power to greatly expand the potential of DSS by incorporating various analyses in complex working environments. As described in Figure 1 below, an intelligent decision support system (IDSS) may utilize tools, such as cognitive science and philosophy, information systems and data processing, numerical simulation and optimization, knowledge-based expert systems, logic and meta programming, intelligent multi-agent technologies, and soft computing technology, to enhance applications.

AI-based decision support systems (DSSs) have been developed for a wide range of environmental applications. They include solid waste management [21], chemical emergency preparedness and response [22], scrap vehicle recycling [23], and combined sewer overflow [24], urban river water quality management [25], and nuclear emergency preparedness and response [26], groundwater remediation [27], air pollution control [28], water supply [29], internet-based fuzzy multicriteria decision support system for planning integrated solid waste management [30], and groundwater remediation [31].

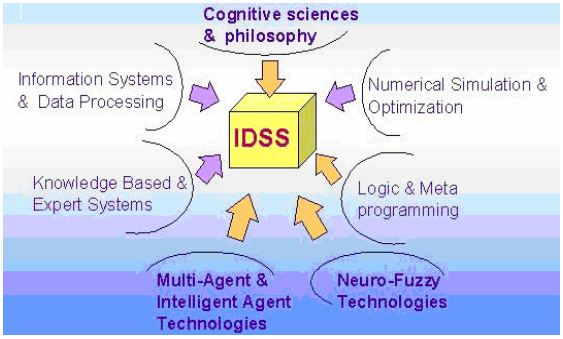


Figure 1. the integration between AI technologies and decision support system.

2.4 Pattern Recognition

Pattern recognition, which is one of the fields within the area of machine learning, aims to classify data (patterns) based on either *a priori* knowledge or on statistical information extracted from the patterns. The patterns to be classified are usually groups of measurements or observations, defining points in an appropriate multidimensional space. However, a complete pattern recognition system consists of a sensor that gathers the observations to be classified or described; a feature extraction mechanism that computes numeric or symbolic information from the observations; and a classification or description scheme focusing on classifying or describing observations based on the extracted features.

The classification or description scheme may include the following approaches: statistical (or decision theoretic) and syntactic (or structural). Statistical pattern recognition is based on statistical characterizations of patterns embedded in the database, assuming that the

patterns are generated by a probabilistic process. Structural pattern recognition is based on the structural interrelationships of features. A wide range of algorithms can be applied for pattern recognition, from very simple Bayesian classifiers and time series analysis to much more complicated neural networks, genetic programming, fuzzy clustering and forecasting, and support vector machine. The classification or description scheme can be done by either supervised learning or unsupervised learning. The former has set of patterns that is termed the training set and the latter has not given an *a priori* labelling of patterns, instead it establishes the classes itself based on the statistical regularities of the patterns.

Remote sensing, with multi-spatial, multi-spectral, and multi-temporal resolutions plus newly developed pattern recognition techniques, provides a powerful tool for measuring environmental systems with various scales. The analysis of satellite images had evolved from ad hoc methods of utilizing spatial and temporal context to the application of artificial-intelligence-oriented procedures of hierarchical scene analysis [32]. Artificial neural network and genetic programming were applied for various types of image processing to aid in water quality management [33]-[37] and estimation of land surface hydrometeorological parameters [38]-[41], ecological factors [42]-[47], and air pollution impact [48] region wide.

Artificial intelligence-based process control that can be used to support environmental systems engineering is also a promising tool to aid in control efficiency in various treatment systems. Complexity and challenges could arise from the possible integration among traditional programming logic control, intelligent control, and the domain knowledge models that drive the system functionally correct to meet engineering goals. For example, dynamic control for in situ groundwater bioremediation requires expertise in both areas of bioremediation and biochemistry [49]. Various hybrid fuzzy control technologies have been applied to improve operational efficiency for municipal incinerators [50], petrochemical wastewater treatment plant [51], and paper mill wastewater treatment plant [52]. Forecasting analysis using neural network models can support high-end local and regional planning for wastewater recycling [53], water supply [54], toxic emissions from several municipal incinerators [55], and nitrogen removal processes in a wastewater treatment plant [56].

III. Future Perspectives

Application of artificial intelligence (AI) can potentially make this decision-making process cost-effective, efficient, and feasible [33]. Applicability of AI techniques to environmental systems engineering is affected by many factors. Firstly, environmental systems are complicated, where a number of factors and interrelationships are hard to be expressed as mathematical formulas. Secondly, information for a number of system parameters is often unavailable, such that rough estimations have to be made. Also, a large portion of information that is available may not be quantifiable; instead, this type of information could be simply the implicit knowledge from decision makers. Thus, inputs for an AI modeling system may only be a small part of the entire information in a study system; consequently, the modeling outputs may only be useful for providing part of decision support, while another part should be from solid investigations of the ambiguous and intangible information. Developing hybrid AI technologies, such as the integration of DKE, pattern recognition, and expert system into a more solid IDSS, might be a way to bridge the gap. Combining various sensors and distributed system may enable us to explore decision-making in an Internet environment to aid in; 1) environmental monitoring and surveillance, 2) engineering forecasting, 3) environmental risk assessment, 4) short-term and long-term environmental planning, 5) composite environmental engineering system design, 6) pollution prevention and industrial ecology, 7) crisis/emergency management, and 8) post-incident damage evaluation and remediation.

IV. CONCLUSION

This paper provides readers an overview of how the AI can be integrated into the environmental systems engineering regime and what would be the possible applications to improve the quality of engineering analysis. In recent years, many emerging methods focusing on computational complexity allow comparison of AI algorithms, while software engineering techniques reduce the effort necessary to develop and maintain complex image processing systems [32]. Advances in computer systems architecture, commercial database technology, and man-machine communications should be closely monitored by the both communities. It is suggested that the completion of planning, design, and operation based on the available data means halfway only, and the remaining half is to examine how information or rules that are unavailable but may present as implicit knowledge to decision makers or stakeholders, which could be collected and analyzed by various AI approaches. This might open a new era in the nexus of environmental cyberinfrastructure and informatics.

REFERENCES

[1] American Association for Artificial Intelligence (AAAI), 2006.

[2] Antoniou, G. and van Harmelen, F. (2004). A Semantic Web Primer, Cambridge, MIT Press.

[3] Omelayenko, B., Crubezy, M., Fensel, D., Benjamins, R., Wielinga, B., Motta, E., Musen, M., and Ding, Y. (2005). UPML: The language and tool support for making the semantic web alive, in Fensel et al., (eds.), MIT Press, 140-170.

[4] Chan, C.W., Peng, Y., and Chen, L. L. (2002). Knowledge acquisition and ontology modeling for construction of a control and monitoring expert system. International Journal of Systems Science, 33(6), 485-503.

[5] Chan, C.W. and Huang, G.H. (2003). Artificial intelligence for management and control of pollution minimization and mitigation processes. Engineering Applications of Artificial Intelligence, Special Issue on Application of Al for Management and Control of Pollution Minimization and Mitigation Processes, 16(2), 75-90.

[6] Ludvigsen, P.J., Sims, R.C., and Grenney, W.J. (1986). A demonstration expert system to aid in assessing groundwater contamination potential by organic chemicals. In W.T. Lenocker (Ed.), Proceedings of the fourth conference on computing in civil engineering (pp. 687–698). New York: American Society of Civil Engineers.

[7] Rynk, R.F., (1992). Computer-integrated monitoring and control of a composting process using an expert system. Ph.D. Dissertation, University of Massachusetts, Amhurst, MA.

[8] Chenu, M. T. and Crenaca, J. A. (1990). The const of remedial action model: expert system applications. In J. M. Hushon (Ed.), Proceedings of the symposium on expert systems for environmental applications Washington, DC: American Chemical Society, pp. 162–175.

[9] Canter, L.W., Knox, R. C., and Sabatini, D. A. (1993). Expert system for prioritization of ground water contaminant sources. Report to US Environmental Protection Agency, Ada, Oklahoma.

[10]Crowe, A.S. and Mutch, J.P. (1994). An expert system approach for assessing the potential for pesticide contamination of ground water. Ground Water, 32(3), 487–498.

[11]Arora, P.A. and McTernan, W. F. (1994). An expert system to determine the probability of pesticide leaching, Agricultural Water Management, 25(1), 57-70.

[12]Betts, K.S., (1998). Preventing pollution by design. Environmental Science and Technology, 32(13), 318–320.

[13]Chowdhury, A.K.M.M. and Canter, L.W. (1998). Expert systems for ground water management. Journal of Environmental System, 26(1), 89–110.

[14]Geng, L., Chen, Z., Chan, C. W., and Huang, G. H. (2001). An intelligent decision support system for management of petroleum contaminated sites. Expert Systems with Applications, 20, 251–260.

[15]Chen, Z., Huang, G. H., Chan, C. W., and Geng, L. (2003). Development of an expert system for the remediation of petroleum-contaminated sites. Environmental Modeling and Assessment, 8, 323–334.

[16]Cheng, J., Yang, Z., and Chan, C.W. (2003). An expert decision support system for industrial water pollution control. Engineering Applications of Artificial Intelligence, Special Issue on Application of Al for Management and Control of Pollution Minimization and Mitigation Processes, 16(2), 159-166.

[17]Chan, C.W. (2005). An expert decision support system for monitoring and diagnosis of an oil production and separation facility, Expert Systems with Application, 29(1), 131-143.

[18]Cheng, H.G., Yang, Z.F., and Chan, C.W. (2003). An expert system for decision support of municipal water pollution control. Engineering Applications of Artificial Intelligence, 16, 159–166

[19]Hu, Z. Y., Chan, C. W. and Huang, G. H. (2003). A fuzzy expert system for site characterization. Expert Systems with Applications, 24, 123–131.

[20] Chen, Z., Huang, G.H., Chan, C.W. and Geng , L. (2003). Development of an expert system for remediation of petroleum-contaminated sites", Environmental Modeling and Assessment, 8(4), 323-334.

[21]Chang, N.B. and Wang, S.F., (1996). The development of an environmental decision support system for municipal solid waste management. Computers, Environment and Urban System, 20(3), 201-212.

[22]Chang, N. B., Wei, Y. L., Tseng, C. C., and Kao, C. Y., (1997). The design of a GIS-based decision support system for chemical emergency preparedness and response in an urban environment. Computers, Environment and Urban System, 21(1), 67-94.

[23]Chang, Y. C., Chang, N. B., and Ma, G. D. (2001). Internet web-based information system for handling scrap vehicles disposal in Taiwan. Environmental Modeling and Assessment, 6(4), 237-248.

[24]Chen, J.C., Chang, N. B., Chang, Y. C. and Lee, M. T. (2003). Mitigating the impacts of combined sewer overflow in an urban river system via web-based share-vision modeling analysis. Journal of Civil Engineering and Environmental Systems, 20(4), 213-230.

[25]Chang, Y. C. and Chang, N. B. (2002). The design of a web-based decision support system for the sustainable management of an urban river system. Water Science and Technology, 46(6), 131-139, 2002.

[26]Chang, N. B. and Hsu, H. Y., "Decision Support System for Emergency Response and Risk Management of Nuclear Power Plants," Proceedings of Australia Taiwan Joint Symposium on Environment Modeling and Management," Taichung, Taiwan, June 28-30, 2000, pp 212-227.

[27]Li, J. B., Huang, G. H., and Zeng, G. M. (2001). An integrated decision support system for the management of petroleum-contaminated sites. Journal of Environmental Science and Health, Part A, 36(7), 23–46.

[28]Zhou, Q., Huang, G.H., and Chan, C.W. (2004). Development of an intelligent decision support system for air pollution control at coal-fired power plants. Expert Systems with Application, 26(3), 335-356.

[29]Olsen, O.R., Dickson, S.E. and Baetz, B.W. (2006). Decision support system for rural water supply in the Nilgiris District of South India. Journal of Environmental Informatics, 7(1), 1-13.

[30]Zeng, Y. and Trauth, K.M. (2005). Internet-based fuzzy multicriteria decision support system for planning integrated solid waste management. Journal of Environmental Informatics, 6(1) 1-15.

[31]He, L., Chan, C.W., Huang, G.H. and Zeng, G.M. (2006), PDSS: A probabilistic reasoning-based decision support system for selecting remediation technologies for petroleum-contaminated sites. Expert Systems with Applications, 30(4), 783-795.

[32]Nagy, G. (1984). Advances in information extraction techniques. Remote Sensing of Environment, 15(2), 167-175.

[33]Panda, S.S., Garg, V. and Chaubey, I. (2004). Artificial neural networks application in lake water quality estimation using satellite imagery. Journal of Environmental Informatics, 4(2), 65-74.

[34]Kishino, M., Tanaka, A., and Ishizaka, J., (2005). Retrieval of Chlorophyll a, suspended solids, and colored dissolved organic matter in Tokyo Bay using ASTER data. Remote Sensing of Environment, 99, 66–74.

[35]Pozdnyakov, D., Shuchman, R., Korosov, A. and Hatt, C. (2005). Operational algorithm for the retrieval of water quality in the Great Lakes. Remote Sensing of Environment, 97, 352 – 370.

[36]Zhang, Y.Z., Pulliainen, J., Koponen, S., and Hallikainen, M., (2002). Application of an empirical neural network to surface water quality estimation in the Gulf of Finland using combined optical data and microwave data. Remote Sensing of Environment, 81(2-3), 327-336.

[37]Keiner, L.E. and Yan, X.H., (1998). A neural network model for estimating sea surface chlorophyll and sediments from thematic mapper imagery. Remote Sensing of Environment, 66(2), 153-165.

[38]Yang, J.S., Wang, Y.Q. and August, P.V. (2004). Estimation of land surface temperature using spatial interpolation and satellite-derived surface emissivity. Journal of Environmental Informatics, 4(1), 40-47.

[39] Tedesco , M., Pulliainen, J., Takala, M., Hallikainen, M. and Pampaloni, P. (2004). Artificial neural network-based techniques for the retrieval of SWE and snow depth from SSM/I data. Remote Sensing of Environment, 90(1), 76-85.

[40]Frate, F.D., Ferrazzoli, P. and Schiavon, G. (2003). Retrieving soil moisture and agricultural variables by microwave radiometry using neural networks. Remote Sensing of Environment, 84, (2), 174-183.

[41]Makkeasorn, A., Chang, N.B., Beaman. M., Wyatt, C., and Slater, C. (2006). Soil moisture prediction in a semi-arid reservoir watershed using RADARSAT satellite image and genetic programming. Water Resources Research, in press.

[42]Guo, X., Wilmshurst, J., McCanny, S., Fargey, P. and Richard P. (2004). Measuring spatial and vertical heterogeneity of grasslands using remote sensing techniques. Journal of Environmental Informatics, 3(1), 24-32.

[43]Carpenter,G.A., Gopal, S., Macomber, S., Martens, S., and Woodcock, C.E. (1999). A Neural Network Method for Mixture Estimation for Vegetation Mapping. Remote Sensing of Environment, 70(2), 138-152.

[44]Filippi, A.M. and Jensen, J.R. (2006). Fuzzy learning vector quantization for hyperspectral coastal vegetation classification. Remote Sensing of Environment, 100, 512 – 530.

[45]Muukkonen, P. and Heiskanen, J. (2005). Estimating biomass for boreal forests using ASTER satellite data combined with standwise forest inventory data. Remote Sensing of Environment, 99, 434 – 447.

[46] Ingram, J.C., Dawson1, T.P., and Whittaker, R. J. (2005). Mapping tropical forest structure in southeastern Madagascar using remote sensing and artificial neural networks. Remote Sensing of Environment, 94, 491–507.

[47]Fang, H.L. and Liang, S. L. (2005). A hybrid inversion method for mapping leaf area index from MODIS data: experiments and application to broadleaf and needleleaf canopies Remote Sensing of Environment, 94, 405–424.

[48]Niang, A., Badran, F., Moulin, C., Cre´pon, M., and Thiria, S. (2006). Retrieval of aerosol type and optical thickness over the Mediterranean from SeaWiFS images using an automatic neural classification method. Remote Sensing of Environment 100, 82 – 94.

[49]Hu, Z.Y., Chan, C.W. and Huang, G.H. (2003). A fuzzy process controller for in situ groundwater bioremediation. Engineering Applications of Artificial Intelligence, 16 (2) 131–147.

[50]Chang, N. B and Chen, W. C. (2000). Fuzzy controller design for municipal incinerators with the aid of genetic algorithms and genetic programming techniques. Waste Management & Research, 18, 341-351.

[51]Chen, W.C., Chang, N.B. and Chen, J.C. (2003). Rough set-based fuzzy neural controller design for industrial wastewater treatment. Water Research, 37, (1), 78-90.

[52]Zeng, G.M., Qin, X.S., He, L., Huang, G.H., Liu, H.L. and Lin, Y.P. (2003). A neural network predictive control system for paper mill wastewater treatment. Engineering Applications of Artificial Intelligence 16, 121–129.

[53]Chen, J.C., Chang, N.B., and Shieh, W.K. (2003). Assessing wastewater reclamation potential by neural networks model. Engineering Applications of Artificial Intelligence, 16(2), 149-157.

[54]Lertpalangsunti, N., Chan, C.W., Mason, R. and Tontiwachwuthikul, P. (1999). A tollset for construction of hybrid intelligent forecasting systems: application for water demand prediction. Artificial Intelligences in Engineering, 13(1), 21-42.

[55]Chang, N.B. and Chen, W.C. (2000). Prediction of PCDDs/PCDFs emissions from municipal incinerators by genetic programming and neural network modeling. Waste Management & Research, 18, 341-351.

[56]Yang, Y.H., Guergachi, A., and Khan, G. (2006). Support vector machines for environmental informatics: application to modelling the nitrogen removal processes in wastewater treatment systems. Journal of Environmental Informatics, 7(1), 14–25.

Reengineering a Rule-Based System towards a Planning System

Kaddour Boukerche
Department of Computer Science
Université du Québec à Montréal, Canada
boukerche.kaddour@courrier.uqam.ca

Hakim Lounis
Department of Computer Science
Université du Québec à Montréal, Canada
lounis.hakim@uqam.ca

Abstract: *This paper is about implementing a solution based on a hierarchical task network (HTN) planning system for operating and optimizing energy production at Alcan's ltd. hydropower network. This solution is reengineered from a former one using a knowledge base expressed with rules. We show that the reengineering process is not straightforward despite the architectural similarities between rule-based systems and planners. We also demonstrate that the planning solution is not only more suitable but it also helps reaching relevant software qualities and improving the whole performance of the system.*

1. Introduction

Alcan Ltd. is one of the two world's biggest players in the aluminum industry. The objective of planning the operation of its hydropower network can be summarized as the satisfaction of the following requirements: effective use of water, account of future hydrological uncertainty, satisfaction of energy need, and, respect of safety constraints. To reach these goals, a decision-making process of water stock management and energy production is used. It consists of four steps: (1) weather/hydro measurements and gathering of the data, (2) data analysis, (3) weather, and, hydrological forecasting, and, (4) planning. In these planning tasks, information processing systems based on mathematical models tested for this kind of applications, are used for optimization and simulation purposes. A great part of the application contains what we consider as expert knowledge within its source code. However, Alcan analysts at different steps of their decision-making process use most of the knowledge implicitly and in a non-automated way. The absence of such a separation between the knowledge level and the inference or reasoning one, has at least two immediate consequences: the restriction in the possibilities of investigation and exploration wanted by Alcan analysts, and, the difficulty of maintaining and making evolve such a system.

One of the most popular solutions for separating knowledge from inference procedures is Knowledge-Based Systems (KBS). KBS are used in numerous application domains, of which the field of hydroelectricity in which this works fits [1] [3] [6]. First, in a former project, we have reengineered the legacy system toward a KBS called HYPERPIK (HYdro PowEr Resources Planning based on Inference and Knowledge); this KBS reproduce the Alcan expert's reasoning [4]. HYPERPIK offered a more flexible solution than its predecessor and allowed Alcan analysts to integrate the necessary expert knowledge and simulate several production energy scenarios. However, the daily use of HYPERPIK showed its limitations in term of some software quality requirements and performance capability concerning advanced simulation scenarios.

All the scenarios carried out in this system simulate the production planning process to lead the system satisfy a certain energy demand. Actually, this objective is very simple and satisfies only a part of the analysts needs. Indeed, they would like to be able to achieve more complexes and more realistic objectives such as: "*to produce 2000 MW (megawatts) at the whole system while producing 900 MW at Saguenay subsystem without discharging*". Unfortunately, such objectives are out of the range of HYPERPIK. There are two main reasons for this. First, the rule-based system does not incorporate necessary goal component for goal-oriented reasoning process rather it proceeds by an exploratory search by using the expert knowledge it has. Secondly, rules are not able to exploit efficiently the hierarchical structure of the hydropower network. Moreover, the problem of the energy production at Alcan is a planning one, since it is a matter of finding a plan that achieves a goal task. Because of the planning nature of the problem and the hierarchical task oriented nature of the planning process, we decided to investigate for a new approach based on HTN planning techniques.

However, building the new solution requires, in a prior step, reengineering the former system. In the beginning, due to the similarities between the two approaches, we thought that the move from the rule-base to the planning domain knowledge base would be straightforward. We were quickly disappointed and we learned that these similarities are only architectural. Indeed, even though the two approaches separate knowledge from reasoning and use close structures to represent rules and operators, their objectives and their engineering processes are completely different.

The paper presents an approach based on HTN planning techniques for operating and optimizing a hydropower network. It also discusses the reengineering process. The implementation uses JSHOP2 [2] (a Java implementation of

the well known planner SHOP2 [5]) as planning component. The paper is organized as follows. Section 2 gives some background on the Alcan hydropower network; section 3 describes the former solution, i.e., the rule-based one and presents its limitations. Section 4 describes the HTN based solution. Section 4.1 gives some background on HTN planning. Section 4.2 shows the domain and problem descriptions in JSHOP2, and, section 4.3 presents the reengineered problem-solving strategies for operating and optimizing the energy production and discusses the problems encountered during the reengineering process. Finally, section 5 presents our conclusions and discusses the lessons we learn.

2. The Alcan hydropower system background

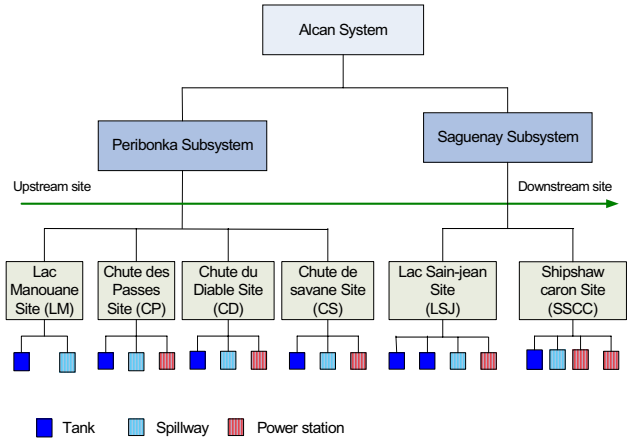


Figure 1: the hierarchical structure of Alcan hydropower network

The Alcan hydropower network under study has a total surface area of 73 800 km², a territory larger than the province of New Brunswick (Canada). The network has, on average, an annual energy capacity of approximately 2000 megawatts (MW); it includes 7 tanks, 6 spillways, 6 hydroelectric power stations, 28 reserve installations, 43 turbine-alternators groups (TAG), 850 kilometres of energy transport lines, a network of about thirty hydro-meteorological stations, etc. The Alcan system is composed of two subsystems. Each one is composed, in its turn, by a set of sites. Each site is an aggregate of tanks, spillways and power stations resources. These resources are directly concerned with the energy production.

The sites are organized in a linear disposition; they are located one downstream the other. The most upstream site is *Lac Manouane* and the most downstream one is *Shipshaw*. Figure 1 illustrates the hierarchical structure of the hydropower network. This structure deeply affects the operation and planning process.

3. The rule-based system and its limitations

KBS are generally involved in classification, diagnosis and planning tasks [4] [9]. Usually, for implementing a KBS, one can use rules as knowledge representation formalism. Rules are an example of explicit knowledge; they allow you to separate the expertise from the application code. Since expert rules are externalized from the application code, they can be changed independently without recompiling the application. The rules have a simple structure, composed of a header, a condition part and an action part. The header part defines the name of the rule, the packet to which the rule is attached, and its priority (if needed). For instance, the name of the rule presented in figure 2 is `manageLSJ-SiteAtItsMaxLevel`; it belongs to the packet containing the strategies for managing the LSJ site and its priority is 14.

```
rule manageLSJ-SiteAtItsMaxLevel{

  packet = StrategiesManagingLSJ-Site;

  priority = 14 ;

  when{
     ?LSJt: Tank("T-LSJ");
     ?LSJt.getLevel()>
          ?LSJt.getNiveauMaxOperation();
     ?lsjs: Site("LSJ");
     ?LSJt.getTotalFlow)-
      ?lsjs.getMaximumTurbinalFlow -
     ?lsjs.getMinimalDischargingFlow()>0
  }
  then {
    modify ?lsjs{
   ?lsjs.managingLSJTankAtItsMaxLevel();
    }
  }
}
```

Figure 2: the rule operating LSJ site at its maximum level

The HYPERPIK implementation yielded about 150 rules organized in 16 packets. A packet allows to group rules with regard to their goal in the whole process. It is currently used within the hydropower resources management group at Alcan Ltd [4]. This system results from a long collaborative process between authors of this paper and Alcan analysts. The rule-based system is an interesting solution, especially in terms of reaching more flexibility and usability since the rules are quite intuitive. However, for several reasons that we will explain further in the text, the daily use of this system shows the limits described bellow.

The system proceeds to plan by firing the applicable rules until a state satisfying the requested energy demand is reached. Since rules-based do not incorporate goal component, we were constrained to code a mechanism that controls the inference engine and detects the desired state. Thus, for each new goal category, it is necessary to code a mechanism, which detects it, and then recompile the code. For sure, it is not an easy task for non-programmers to code mechanisms that detect a goal such as "*produce 2000 MW at the whole system while producing 900 MW at Saguenay subsystem without discharging*". This goal can not be satisfied by a single state but by a set of states traversed by

the plan. It is a severe limitation, which does not allow the analysts to simulate realistic scenarios. A flexible solution would be to incorporate a very expressive goal component and then, eliminate the need to mechanisms detecting goals.

Decisions made in HYPERPIK are based only on information about the current state and the knowledge contained in the rules. In large applications, such as hydropower production, knowing the current state is not enough for obtaining the expected efficiency. This is another motivation for the need to some kind of goal information, which describes situations that are desirable. From this information, one can derive some interesting search control knowledge.

Moreover, the tuning of the rule knowledge base is a long and a meticulous work, which required many adjustments. Indeed, for each simulation we have to group rules in packets and adjust their priorities, in order to be close to simulation scenarios wanted by experts. Building packets means breaking down the overall expertise into multiple smaller parts, where each part handles a specific task. This corresponds to one-shot decomposition of the rule-base into packets. It's neither a natural nor a flexible way to reduce tasks into subtasks. Rules, packets, and, priorities don't reflect the decomposition process; the knowledge base becomes difficult to understand, maintain, and, evolve.

4. Representing and operating the hydropower network with HTN formalism

In recent years, the planning community has focused on algorithms and has improved significantly the complexity of problems that can be solved [13], but the problems of acquiring, constructing, validating and maintaining this kind knowledge are still considerable. In fact, in the planning area, the problem of knowledge engineering is still an obstacle to make planning technology more accessible [7]. A few years ago, tools for planning domain acquisition were almost inexistent, except the two pioneering knowledge-based planning systems O-Plan [11] and SIPE [12]; they have been led by necessity to consider knowledge acquisition issues which resulted in dedicated tool support. But these tools are quite specific to their respective planners.

The knowledge engineering carelessness in the planning community justifies its inaccessibility and explains the recourse to KBS to solve planning problems [4] [9]. However, more recently, it seems that a wind of change has blown in this community, and more research dedicated to planning domain knowledge engineering is arising. In 2005, we had attended the first international competition for tools supporting the planning domain knowledge engineering [8]. This is an encouraging step that will help the discipline of planning to be more accessible to real world applications [10]. Finally, to our knowledge, our work is a precursor by discussing reengineering problems of converting rules towards planning domain knowledge.

4.1 HTN planning background

The problem described in this work is a planning one and knowledge-based planners have been used to solve large real world problems. There is no denying that most practical planning systems have used HTN techniques [11] [12] [5], and, anyone currently considering a serious planning application would be well advised to consider this approach [14]. The success of these planners is due to the use of domain specific knowledge. This knowledge guides efficiently plans generation and acts as control search knowledge.

Traditionally, a planning problem is posed as one of finding an actions sequence that will transform a modeled world from an initial state into a desired goal state, given only the description of the executable actions in the domain. In HTN planning, we identify two kinds of tasks: primitive tasks that could not be reduced and compound (or abstract) tasks that are reducible into primitive ones. For instance, in the Alcan context, computing the level of a tank is considered as a primitive task.

For each non-primitive task, the planner chooses an applicable decomposition procedure called '*method*', instantiates it to reduce the task into a completely ordered or partially ordered set of subtasks. Each subtask can be compound or primitive. The decomposition process continues until a sequence of primitive tasks called plan is obtained. If the plan later turns out to be infeasible, the planning system will need to backtrack and try other methods. Methods describe strategies or "standard operating procedures" that one would normally use to perform tasks in some domain.

4.2 Planning domain knowledge representation

An HTN planning problem is defined by a set of operators, a set of methods, an initial state, and also, a set of tasks to perform. An operator indicates how a primitive task can be performed. Each operator `o` is annotated with the prefix '!' and has a head `head(o)` consisting of the operator's name and a list of parameters, a precondition expression `pred(o)` indicating what should be true in the current state in order for the operator to be applicable, and a delete list `del(o)` and an add list `add(o)` giving the operator's negative and positive effects.

Figure 3 contains an example of an operator for performing the action of closing a valve. The operator's name is `!closeValve` and the parameter `?valve` is a variable (indicated by the prefix `?`) which will contain the value of the valve to close. The operator is applicable only if the valve is not closed, `(not(closed ?valve))` at the current state. The effects of the operator application change the resulting state description by removing the fact

`(not(closed ?valve))` which is no longer true, and, by adding the new true fact `(closed ?valve)`.

```
(:operator (!closeValve ?valve)
   ;; precondition
   (not(closed ?valve))

   ;; delete list
   (not(closed ?valve))

   ;; add list
    (closed ?valve)
)
```

Figure 3: a HTN operator schema for closing valves

Figure 4 contains an example of a method describing how to compute the task of producing a certain amount of energy at the hydropower system. The method has no precondition (the empty parentheses in line 3), and, decomposes the task into two subtasks. In the first subtask, the subsystem *Peribonka* is charged to produce a part of the energy; in the second subtask, the *Saguenay* subsystem must provide the rest.

The initial state is described by the properties initial values of the hydroelectric resources (e.g., tanks levels, GTA and valves, etc.). Finally, *produce a total energy demand of 2000MW* is an example of a task to perform.

4.3 Reengineering scenarios for operating and optimizing energy production

KBS and planners continue to keep a logical separation between knowledge representation and reasoning engine. This architectural similarity and the close formalisms of rule and operator representation led us in the beginning to believe that the conversion of rules to operators and methods would be straightforward. More precisely, we thought that each rule would be converted to an operator. We were quickly disappointed and learned later that despite these similarities, there are some peculiarities of planning that clearly distinguish planning knowledge engineering from the general field of KBS. Moreover, we learned that the rule-base is not sufficient to derive the planning domain knowledge. The ultimate use of the planning domain model is to be part of a system involved in the 'synthetic' task of plan generation. This distinguishes it from the more traditional diagnostic or classification problems familiar to knowledge-based systems. Also, the knowledge elicited in planning is largely knowledge about operators and how operators act on the object of the application domain [7].

```
(:method (computeTotalDemandedW ?w)
   ;; no precondition
   ()
   ;; the demanded energy is distributed between
   ;;  two subsystems Peribonka and Saguenay

   ;; subtask1: Peribonka subsystem energy production
   ((computePeribonkaW  ?w)

   ;;  subtask2: Saguenay subsystem energy production
   (computeSaguenayW(- ?w (call
                           getPeribonkaW))))
)
```

Figure 4: a HTN method for computing an amount of energy in the hydropower network

Given a demand of energy to satisfy, the process of planning starts by choosing an applicable method to this task and decomposes it into subtasks. When no method exists, this issue of solution fails (i.e., the followed branch of the space search is abandoned). For instance, in figure 4 the task, which consists in producing 2000 MW, is reduced into two subtasks in order to distribute the energy production between the two subsystems *Péribonka* and *Saguenay*. In its turn, the production energy task for each subsystem is decomposed into subtasks in order to distribute the energy production on the sites that belong to it.

Both figures 2 and 6 illustrate the same strategy, which manages the site *Lac Saint-Jean* at its maximum level. The rule and the method are functionally equivalent, both of them calculate: (1) the quantity of water that will be

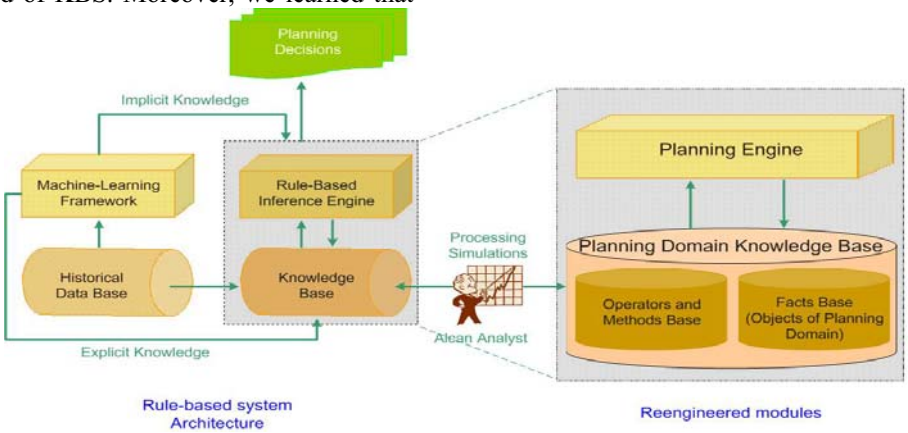


Figure 5: the reengineered modules from the HYPERPIK system

routed to power stations from the tank, (2) the amount of water that will be diverted from this tank down on spillway, and, (3) the produced energy at the power house. The interesting question to answer is about the possibility of deriving the method from the rule? Figure 5 presents the modules to reengineer.

```
(:method (manageLSJ-siteAtMaxLevel ?lev)

   ;; precondition
   (
   (assign ?totalApport
   (call getLSJTotalApport))
   (assign ?TurbFlow
   (call getLSJMaxTurbinalFlow))
   (assign ?w_prod
   (call computeLSJEnergyProduction
   ?turbFlow))
   )
   ;; primitive subtasks
   (
   ;; operating LSJ Tank
   (!computeFlowToTurbineFromLSJTank
   ?lev ?approtTotal)

   ;; operating LSJ Spillway
   (!computeDischargingFlowAtLSJSpillway
   ?lev ?disFlow)

   ;; operating LSJ Power Station
   (!computeLSJ-PowerStationW
?turbFlow ?w_prod)
)
```

Figure 6: a HTN method operating the LSJ site at its maximum level

Answering this question required a deep exam of the rule and its correspondent packet. Once this is done, we reported two main observations. First, we learned that the expertise of managing sites is broken up to smaller parts and each part is dedicated to a particular site. Secondly, we did not succeeded to know how this decomposition was done so we proceeded to further investigations in the objects domain source code. Unfortunately, all we have found is that System is an aggregate of Subsystems and subsystem is an aggregate of Sites. Indeed, it helps to discover the hierarchical structure of the hydro-network but does not help to know how the decomposition is done. The coding of the method required the elicitation of the decomposition knowledge used by analysts.

Now, it remains to know from where primitive tasks specified in the subtasks part and their corresponding operators did they come, knowing that in the rule there is no information about operators? A part of the necessary knowledge for building the operators schemas is hidden in the source code. Actually, operators schemas corresponding to primitive tasks illustrated in figure 6 are partly extracted from the source code of the function `managingLSJTankAtItsMaxLevel()` (see figure 2). Even thought the function source code contains all the objects instances that can form the parameters list of the operator. It remains not clear for us how to obtain such a list. The problem is: which instances to choose and in which order? Despite the structure similarity between rules and operators, there are important differences between the two concepts, which make the move from one to another not an easy task.

In HYPERPIK, Alcan analysts opted for a solution which privileges extracting the knowledge that will be useful in the development of new scenarios. For that, it was sufficient to abstract the knowledge to site level component. This degree of abstraction made it possible to reduce the number of rules and at the same time facilitate, in a certain way, the tuning of the knowledge base. On the other hand, in the planning solution, the focus is put on the synthesis of a plan composed by actions, which operate tanks, spillways, and, power stations resources. In this case, abstraction on the site level component is not suitable any more. This shift in the abstraction levels adopted in the two systems explains partly the problems encountered during the reengineering process.

Finally, the most important characteristic of HTN is its great expressive power. HTN expressiveness makes it possible for analysts to carry out a very large spectrum of scenarios, which can perform complex and realistic objectives. In a HTN planning solution, analysts have the possibility to code strategies for performing tasks such as producing *2000 MW in the whole Alcan system while producing 900 MW at the Saguenay*, *without discharging at Peribonka.*

```
(:method (alcanProductionConstPeribonka
                                    ?w ?peribW)
  ;; no precondition
  ()

  ;; product ?peribW amount of energy at Peribonka sybsystem
  ((computePeribonkaW ?peribW)

  ;; Product the rest of the demanded energy from
  ;; Saguenay subsystem
   (computeSaguenayW (- ?w
                        (call getPeribonkaW)))
  )
)
```

Figure 7: the method constraining *Saguenay* to produce 900 MW while the total energy demand is 2000 MW

In order to achieve the goal formulated above, we note in figure 7, that the adopted approach consists in trying to produce 900 MW at *Saguenay* first and then producing the remainder of the demanded energy at *Péribonka*. In this solution, the planning process is calculated from downstream to upstream resources. The order is no longer predefined nor a barrier for performing various objectives.

5. Conclusions and lessons learned

In this paper, we described an undergoing work consisting of implementing a HTN planning based solution for operating and optimizing hydro energy production. So far, the results are promising. The following points could summarize the strengths of the proposed solution:

- Greater flexibility of the tool during its use within the planning process, by facilitating the exploration of new power network management scenarios for accomplishing a wide range of complex and realistic tasks, without changing any line of source code. It is the main need of Alcan analysts.
- Better performances, since HTN methods acts as powerful search control knowledge. Knowing the goal task to perform, the implemented expert strategies narrow the planning process. In rule-based system the planning process is exploratory.
- Better evolvability, since the tasks decomposition methods are well structured. Adding and updating strategies becomes a convenient task. Analysts can easily target the interested methods and bring the necessary improvements.
- Better reusability of the planning domain knowledge base. We have designed standard strategies that could be used in several simulation scenarios.
- The understandability of the planning knowledge is higher than it was in the rule-base because HTN methods are closer to the way that analysts think about the problem. HTN is more suitable for capturing the hierarchical nature of the planning process than the flat structures of rules and packets.

Nevertheless, HTN planners are unable to handle tasks that were not explicitly anticipated by the designer. To overcome this problem and to bring more flexibility to the formalism, we think that HTN must incorporate a component for handling declarative goals. Finally, concerning the reengineering of rules towards a planning domain base and vice-versa, it exists a big gap to fill and maybe all a discipline to put in place. Actually, solutions based on KBS and planning techniques are used more and more, and, the problem of methods and tools to support their reengineering process will be posed.

6. References

[1] J. Chang Tiao, D. Moore, “Reservoir Operation by the Use of an Expert System”, Ohio Univ, Athens, OH, USA. Proc. of the 1994 ASCE National Conference on Hydraulic Engineering. Buffalo, NY, USA, 1994.

[2] O. Ilghami and D. S. Nau. “A general approach to synthesize problem-specific planners”. Tech. Rep. CS-TR-4597, UMIACS-TR-2004-40, University of Maryland, Oct. 2003.

[3] A. S. Leslie, A. Moyes, J. R. McDonald, G. M. Burt, J. McGowan, W. Charlesworth, “CEPE, Intelligent System for the Management of a Hydro-electric Scheme”, Strathclyde Univ., Glasgow, UK. 31st Universities Power Engineering Conference, Iraklio, Greece, 1996.

[4] H. Lounis, K. Boukerche & H. Sahraoui, “Reengineering an Industrial Legacy Software Towards an Object-Oriented Knowledge-Based System”. In proceedings of the 16th international conference on Software Engineering and Knowledge Engineering (SEKE'04), June 2004, Banff, Alberta, Canada.

[5] D. Nau, T.C. Au, O. Ilghami, U. Kuter, J. W. Murdock, D. Wu, and F. Yaman. “SHOP2: An HTN planning system”. Journal of Artificial Intelligence Research 20, Dec. 2003.

[6] S. Samarasingh., A. McKinnon, J. Bright, “Expert System for Flood Management in Lake Manapouri”. IEEE Comput. Soc. Press, Los Alamitos, CA, USA. First New Zealand International Two-Stream Conference on Artificial Neural Networks and Expert Systems, Dunedin, New Zealand, 1993.

[7] T. L. McCluskey, R. Simpson. “Knowledge Formulation for AI Planning”. In Engineering Knowledge in the Age of the Semantic Web; E.Motta, N.Shadbolt, A.Stutt and N.Gibbins (eds), Lecture Notes in Artificial Intelligence, No. 3257, pp 449 - 465, (Proceedings of EKAW'2004), published by Springer, October 2004.

[8] Roman Barták and Lee McCluskey, editors. “Working notes of the ICAPS'05 First International Competition on Knowledge Engineering for Planning and Scheduling”, Monterey, CA (USA), June 2005. AAAI, AAAI Press.

[9] C. Ullrich. “Pedagogical rules in ACTIVEMATH and their pedagogical foundations”. Seki Report SR-03-03, Universitat des Saarlandes, FB Informatik, 2003.

[10] Carsten Ullrich. “Course Generation Based on HTN Planning”. Proceedings of 13th Annual Workshop of the SIG Adaptivity and User Modeling in Interactive Systems, 74-79, 2005

[11] A. Tate, S. T. Polyak, and P. Jarvis. “TF Method: An Initial Framework for Modeling and Analyzing Planning Domains”. Technical report, University of Edinburgh, 1998.

[12] K. Myers and D. Wilkins. “The Act-Editor User’s Guide: A Manual for Version2.2”. SRI International Artificial Intelligence Center, 1997.

[13] D. McDermott. “International Planning Competitions (1998, 2000, 2002, 2004)”. At http://cs-www.cs.yale.edu/homes/dvm/

[14] D. Smith, J. Frank, and A. Jonsson. “Bridging the gap between planning and scheduling”. Knowledge Engineering Review, 15(1):61-94, 2000.

Reliability Analysis of Pipe and Filter Architecture Style

Swapna S. Gokhale
Dept. of CSE
University of Connecticut
Storrs, CT 06269
Email: ssg@engr.uconn.edu

Sherif Yacoub
Hewlett-Packard Labs
Barcelona Research Office
Barcelona, Spain
Email: sherif.yacoub@hp.com

Abstract

A number of software architecture styles have been identified and these styles have been analyzed extensively for different non functional attributes including performance, maintainability, flexibility and modifiability. Reliability analysis of architecture styles, however, has been relatively less investigated. In this paper we present a comprehensive reliability analysis methodology for one such architecture style, namely, the pipe and filter style. We consider two variants of the pipe and filter style, namely, linear topology with and without feedback. We consider the impact of (i) error propagation and downstream error correction, and (ii) deterministic number of iterations of the feedback loop with filter reliabilities a function of the number of iterations, on the application reliability. We illustrate the methodology on an industrial case study of a Document Analysis and Understanding Application.

1 Introduction and motivation

Software architectural choices have a profound influence on the non functional attributes of a system and architecture analysis can be used to assess the degree to which a given architecture supports important quality attributes such as performance [7, 12] and reliability [6, 8, 14]. Over the years, some commonly occurring patterns of the structural organization of components and connectors, referred to as architecture styles [4], have been identified. These styles have been analyzed extensively for different quality attributes including performance, maintainability, and modifiability. Little attention has been provided, however, towards the reliability analysis of architecture styles [2, 11].

In this paper we develop a comprehensive reliability analysis methodology for one such architecture style, namely, the pipe and filter style. We consider two variants of pipe and filter style, namely linear topology with and without feedback. The methodology considers the impact of (i) error propagation and downstream error correction, and (ii) deterministic number of iterations of the feedback loop with filter reliabilities a function of the number of iterations, on the application reliability. We obtain analytical expressions for application reliability in terms of the pipe and filter reliabilities and the feedback loop parameters. We illustrate the methodology using an industrial case study of a Document Analysis and Understanding Application.

The layout of the paper is as follows: Section 2 provides an overview of the pipe and filter style. Section 3 describes the analysis methodology. Section 4 illustrates the methodology with a case study. Section 5 summarizes the related research. Section 6 offers concluding remarks and outlines future research directions.

2 Overview of pipe and filter style

In the pipe and filter architecture style, each component has a set of inputs and outputs. A component reads a stream of data on its input, and produces a stream of data on its outputs. Input is transformed both locally and incrementally so that output begins before the entire input stream is consumed. Components are termed filters; connectors serve as conduits for the information streams and are termed pipes. The main characteristics of this style include the condition that filters must be independent entities, and they need not know the identities of the upstream and the downstream filters. They may specify the input format and guarantee what appears on the output, but they may not know which components appear at the ends of those pipes.

Though the pipe and filter architecture appears structurally simple, it is the basic underlying style of many applications as it offers many advantages [4]. Filters are usually stand alone and can be treated as black boxes that encapsulate specific functionality. This encapsulation helps ensure quality attributes such as information hiding, low coupling, high cohesion, and reuse. This style has been in use in many

critical practical applications, e.g. operating systems (Unix pipes and filters) and compilers (laxer, parser, semantic analyzer, code generation). It is also used in applications that are heavily based on data flow between concurrent, mostly independent processes. For such applications, reliability is of paramount importance.

3 Reliability analysis methodology

We consider a linear topology of n filters, labeled sequentially from 1 through n starting at the left. The filters are connected by pipes, and each pipe is labeled $(a, a+1)$, where a ranges from 1 through $n-1$. We consider two cases of the linear topology. In the first case, termed as "linear topology without feedback", shown in Figure 1, each element of input data is processed sequentially starting from the first filter all the way to the n^{th} filter. The second case includes a feedback loop in the topology, between filters n_1 through n_2 as shown in Figure 2 and is termed as "linear topology with feedback". Thus, each element of input data is processed by filters 1 through $n_1 - 1$ and by filters $n_2 + 1$ through n just once, and may be processed multiple times by filters n_1 through n_2. A feedback loop may be used in the architecture to ensure that the quality of the output produced is acceptable, according to application-specific criteria. For example, in a document transformation system (described in Section 4), the final output can be checked for "look-and-feel" and to ensure that no part of the page has been cut off by the processing algorithms. In a image processing system, the output can be checked for acceptable image resolution. After the data is processed by filter n_2, the quality of the output is checked and if the output does not satisfy the desired quality, it is discarded and the data element is subject to another round of processing by filters n_1 through n_2. This process is repeated until either the quality of the output obtained from filter n_2 is satisfactory or a certain maximum number of iterations denoted m are completed. In each round of processing, a modified or other feed forward algorithm is applied to improve the quality. In addition, the components used may be reconfigured, tuned or replaced to increase the possibility of obtaining an acceptable result. The maximum number of iterations m is determined by the number of possible configurations of the filters that can be explored to improve quality. The output from each feedback iteration is retained until a decision to break from the loop is reached, after which the best output from the history of iterations is selected. We let q_r denote the probability that the output obtained from filter n_2 in the r^{th} iteration satisfies the quality criteria. In this case, each time the data element is processed by filters n_1 through n_2 to improve the quality, there is a risk that the filters may fail. However, since the data element was processed correctly in the previous rounds, the probability of failure decreases for each subsequent round. Thus, the probability of processing an element correctly for filters n_1 through n_2 is an increasing function of the number of iteration.

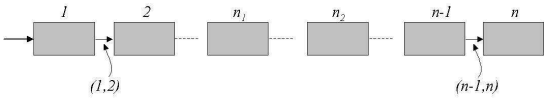


Figure 1. Linear topology without feedback

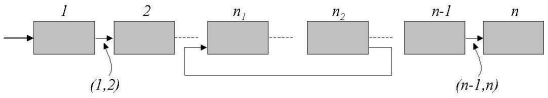


Figure 2. Linear topology with feedback

Each filter receives one element of data at a time and it processes that element. The input to the first filter is provided externally, while the inputs to filters 2 through n are provided by the outputs of the upstream filters 1 through $n-1$ respectively. This input/output relationship among the filters holds even in the case of the topology with feedback. In this case, if additional rounds of processing by filters n_1 through n_2 are necessary, then the input to filter n_1 in these additional rounds is received from filter $n_1 - 1$ as in the first round and not from filter n_2. Thus, the feedback loop is a control loop and not a data loop.

The processing of each data element by a filter results in three possibilities, it produces a correct or an incorrect result, or it does not produce any output. Which one of these three possibilities is considered as a filter failure depends on the employed policy. We consider two policies, namely, conservative and opportunistic. In the conservative policy, when a filter produces incorrect result or no result, it is considered as failed. In this case, the incorrect result is not transmitted via the pipe to the downstream filter. On the other hand, in the opportunistic policy, a filter is considered as failed only when it produces no output. If a filter produces an incorrect output, it is passed on to the subsequent downstream filter for further processing. Thus, in this policy, the downstream filters have the opportunity to correct the error committed by an upstream filter and provide correct output despite the fact that one or more of the filters may have produced an incorrect result, due to which this policy is termed opportunistic. Assuming that the external input is always correct, in the conservative policy, the inputs received by each filter are always correct, whereas in the opportunistic policy, the inputs received by each one of the filters except the first one can be either correct or incorrect. We consider three scenarios based on the combination of the policies and the topologies described above, namely, a linear topology without feedback using the conservative

policy, linear topology without feedback using the opportunistic policy and linear topology with feedback using the conservative policy.

For each filter, we use the following notation to represent the different conditional and total probabilities and inputs:
$p_j(C)$ – Probability that filter j produces correct output.
$p_j(I)$ – Probability that filter j produces incorrect output.
$p_j(N)$ – Probability that filter j produces no output.
$p_j(C|C)$ – Probability that filter j produces correct output upon receiving correct input.
$p_j(C|I)$ – Probability that filter j produces correct output upon receiving incorrect input.
$p_j(I|C)$ – Probability that filter j produces incorrect output upon receiving correct input.
$p_j(I|I)$ – Probability that filter j produces incorrect output upon receiving incorrect input.
$p_j(N|C)$ – Probability that filter j produces no output upon receiving correct input.
$p_j(N|I)$ – Probability that filter j produces no output upon receiving incorrect input.
$I_j(C)$ – Probability that filter j receives correct input.
$I_j(I)$ – Probability that filter j receives incorrect input.

Using the theorem of total probability, for filter j, the probability of correct, incorrect and no output is given by:

$$p_j(C) = I_j(C)p_j(C|C) + I_j(I)p_j(C|I) \quad (1)$$

$$p_j(I) = I_j(C)p_j(I|C) + I_j(I)p_j(I|I) \quad (2)$$

$$p_j(N) = I_j(C)p_j(N|C) + I_j(I)p_j(N|I) \quad (3)$$

Since for the linear topology with feedback, only the conservative policy is employed, $p_j(C) = p_j(C|C)$ for all the filters. In this case, for filters n_1 through n_2, the probability of producing a correct output is a function of the number of processing iteration. We let $p_j(V, C)$ denote this iteration-dependent probability of producing a correct output, where V is the number of iteration. The rationale for determining the functional form of $p_j(V, C)$ with respect to the number of iterations is as follows. As the number of iterations increase, the probability of producing correct output should approach 1.0, but it should never exceed 1.0. Secondly, the rate at which this probability increases should decrease as the number of iterations increase. Thus, the incremental gain afforded in going from the first iteration to the second should be less than the incremental gain in going from the second iteration to the third and so on. To satisfy the above rationale, we assume that $p_j(V, C)$ takes the following functional form:

$$p_j(V, C) = p_j(C)^{\frac{1}{V*d}} \quad (4)$$

where the parameter d determines the rate at which $p_j(V, C)$ approaches 1.0. d needs to be adjusted in such a way that $p_j(V, C)$ approaches 1.0 in the maximum possible number of iterations. $p_j(C)$ is the probability of producing a correct output in the first iteration. The higher the values of $p_j(C)$ and d, the more rapidly $p_j(V, C)$ approaches 1.0.

A pipe does not process and modify data, it only serves as a vehicle to transport data. Thus, the failure of a pipe implies unavailability of the channel used for the transport of data. We let $p_{a,a+1}$ denote the probability that the pipe $(a, a + 1)$ is available to transport a single element of data.

The reliability of the application, defined as the probability of processing a single element of input data correctly for the three scenarios is obtained as follows.

I: Topology without feedback, conservative policy
In this case, since the incorrect output of each filter is regarded as a failure and is not forwarded to the downstream filter for further processing, for each filter including the first one $I_j(I) = 0$, and $I_j(C) = 1$. Thus, from Equations (1), (2), and (3), $p_j(C) = p_j(C|C)$, $p_j(I) = p_j(I|C)$ and $p_j(N) = p_j(N|C)$. The application reliability is given by:

$$R = \left(\prod_{i=1}^{n} p_j(C)\right)\left(\prod_{i=1}^{n-1} p_{i,i+1}\right) \quad (5)$$

II: Topology without feedback, opportunistic policy
In this case, filter j ($j > 1$) receives both incorrect and correct inputs, when the outputs produced by its upstream filter $j - 1$ are incorrect and correct respectively. Thus, based on the probabilities of correct and incorrect outputs of filter $j - 1$, namely, $p_{j-1}(C)$ and $p_{j-1}(I)$, $I_j(C)$ and $I_j(I)$ can be obtained using the following expressions:

$$I_j(C) = \frac{p_{j-1}(C)}{p_{j-1}(C) + p_{j-1}(I)} \quad (6)$$

$$I_j(I) = \frac{p_{j-1}(I)}{p_{j-1}(I) + p_{j-1}(C)} \quad (7)$$

The normalization in Equations (6) and (7) is necessary since of the three types of outputs produced by filter $j - 1$, only two are fed to filter j. When filter $j - 1$ produces no output, no input is fed into the downstream filter j and the application is considered as failed.

Using the expressions for $I_j(C)$ and $I_j(I)$ from Equations (6) and (7), Equations (1)-(3) can be written as:

$$p_j(C) = \frac{p_{j-1}(C)p_j(C|C) + p_{j-1}(I)p_j(C|I)}{p_{j-1}(C) + p_{j-1}(I)} \quad (8)$$

$$p_j(I) = \frac{p_{j-1}(C)p_j(I|C) + p_{j-1}(I)p_j(I|I)}{p_{j-1}(C) + p_{j-1}(I)} \quad (9)$$

$$p_j(N) = \frac{p_{j-1}(C)p_j(N|C) + p_{j-1}(I)p_j(N|I)}{p_{j-1}(C) + p_{j-1}(I)} \quad (10)$$

Equations (8)-(10) hold for filters 2 through n. Assuming that the external input to the first filter is always correct, $p_1(C)$ and $p_1(I)$ are given by $p_1(C|C)$ and $p_1(I|C)$ and the application reliability is given by:

$$R = \left(\prod_{i=1}^{n-1}(p_i(C) + p_i(I))\right) p_n(C) \left(\prod_{i=1}^{n-1} p_{i,i+1}\right) \quad (11)$$

Equation (11) indicates that for filters 1 through $n-1$, both correct and incorrect outputs are regarded as success. For filter n, however, only the case of correct output is regarded as success, while producing incorrect and no output are considered as failures.

III: Topology with feedback

In this case, to obtain an expression for the application reliability, we first need to obtain an expression for the average reliability for multiple processing iterations by filters n_1 through n_2. In the first round, the probability that filters n_1 through n_2 process a data element without incurring a failure is given by $\prod_{j=n_1}^{n_2} p_j(C) \prod_{j=n_1}^{n_2-1} p_{j,j+1}$. With probability q_1 the output produced by the filter cascade n_1 through n_2 is of acceptable quality. Thus, with probability $(1-q_1)$ the output produced is of unacceptable quality requiring a second processing round. In the second round, the probability that during processing by the filter cascade n_1 through n_2 a failure does not occur is given by $\prod_{j=n_1}^{n_2} (p_j(C))^{\frac{1}{2d}} \prod_{j=n_1}^{n_2-1} p_{j,j+1}$. With probability q_2 this output is of acceptable quality requiring no further action, and with probability $(1-q_2)$ a third round of processing is necessary. Using the theorem of total probability, the probability that an output of acceptable quality is obtained at the l^{th} iteration is given by $(\prod_{i=1}^{l-1}(1-q_i))q_l$ for $2 \leq l < m-1$. The reliability of the cascade for the l^{th} iteration is given by $(W(C))^{1+\frac{1}{2d}+\ldots+\frac{1}{ld}} {W_p}^l$ where $W(C) = \prod_{j=n_1}^{n_2} p_i(C)$, and $W_p = \prod_{j=n_1}^{n_2-1} p_{j,j+1}$. The extreme case, namely, $l = m$ merits special consideration. In this case, the probability of obtaining acceptable quality output is $(1-q_1)(1-q_2)\ldots(1-q_{m-1})$. Since m is the maximum number of iterations that are possible with different filter configurations, $q_m = 1$. In other words, if an output of acceptable quality cannot be obtained after $m-1$ iterations by the filter cascade n_1 through n_2, then the output produced in the m^{th} iteration is considered acceptable if no failure occurs. Then the average reliability of the filter cascade n_1 through n_2, denoted W_{III} is given by Equation (12), where the first and the second terms consider the case of obtaining correct output of acceptable quality in the first and 2^{nd} through m^{th} iterations respectively.

$$W_{III} = W(C)W_p q_1 + \sum_{l=2}^{m}(\prod_{i=1}^{l-1}(1-q_i)q_l) \; (W(C))^{1+\sum_{i=2}^{l}\frac{1}{id}} {W_p}^l \quad (12)$$

The application reliability is then given by:

$$R = (\prod_{i=1}^{n_1-1}(p_i(C) * p_{i,i+1})(W_{III})(\prod_{i=n_2+1}^{n}(p_i(C) * p_{i-1,i}) \quad (13)$$

Since the conservative policy is employed, $p_j(C)$ is the same as $p_j(C|C)$ for all the filters.

4 Case study

Document understanding or "remastering" refers to the process of converting paper material such as books, magazines, journals, etc. into a searchable electronic form with information that is meaningful to both human beings and machines [13]. Typically, the system input is document pages in a raster format (TIFF or BMP) and the output is searchable Portable Document Format (PDF) or eXtensible Markup Language (XML) documents. The documents are remastered for use in the digital libraries of Web communities, such as the cognitive science community [1].

The architecture of the document understanding system under consideration is based on component-based software engineering principles. Components perform the document understanding functions, such as optical character recognition (OCR), layout analysis, and logical structure analysis. Each component is self-contained and fairly independent, and provides well-defined services and functions. We wrap each component such that the wrapper handles inputs, outputs, and errors, and provides the execution context. Components have no explicit knowledge of each other, whether they are on the same worker machine or not. Processes run in parallel over a distributed workstation cluster. The system is composed of multiple components and scripts that run in parallel with monitoring components, watchdogs, loggers, etc. We extract the pipeline responsible for the analysis of documents, shown in Figure 3, to illustrate our methodology. Our previous work has analyzed the performance of this pipeline [5]. Note that for this analysis, the

Figure 3. Application architecture, wo feedback

pipes are control pipes; that is, they carry a description of the data to be processed but not the data. The data in these applications is large (in MB), and hence is stored on a storage sever. The location of the data to be processed is carried in the task description which goes into the pipe.

Figure 4 depicts the same architecture when a feedback loop is added. The most critical component is the Content Analyzer / Converter filter which performs most of the

analysis. A feedback loop is added whose function is to maximize the possibility of getting a higher quality output. The feedback loop works as follows: the output from the PDF verifier is used to determine whether the quality is acceptable. If yes, the output is fed to the writer filter. If not, the feedback loop is used to reinvoke the converter to redo the analysis. The analysis is performed in every feedback turn with different parameters and settings. For this system, a set of 5 different types of analyzers are used; some of which use the same technology with different parameter values (for example changing the exposure in the input page) and others use different analysis technology (for example using polygonal region analysis or quadratic).

The conditional probabilities of the filters could be obtained from expert opinion, domain knowledge or estimated by statistical testing of the components. We elicited these probabilities through an extensive consultation with experts, and these are reported in Table 1. The rationale provided by the experts is as follows. The Tape Loader uploads the data from the archive tapes into the system, and encounters only one type of errors, namely, the component may not write the data correctly to the system disks, that is, $p(N|C)$. It will never produce correct output given incorrect input, hence $p(C|I)$ is set to zero. The Writer filter component has proven to be sufficiently reliable so its $p(C|C)$ was set to one. The filter reads output data and groups it into manageable writable disks. The Input Tiff filter reads the input and checks for its quality. It may decide that the input is correct while it is not and hence $p(C|I)$ is set to 0.5. Also, there is a likelihood that it will produce incorrect output given that the input was correct. The second case, however, is less likely and most of the errors observed from this filter are of the first type, so $p(I|C)$ is set to 0.025. The PDF Verifier is selected to be highly reliable and is based on a stable commercial component, because based on its output, a decision is made about output quality. The Content Analyzer is the most critical and error prone component, which encompasses sophisticated image processing and analysis algorithms. It is the most important component, but it is the one most likely to fail. The probabilities of obtaining an acceptable quality output for the five feedback loop iterations, namely q_1, q_2, q_3, q_4 and q_5 are set to 0.96, 0.97, 0.98 0.99 and 1.00 respectively. The value of d is set to 1.00. The pipe reliabilities are assumed to be 1.00.

Figure 4. Application architecture, w feedback

For these parameter values, the application reliability for the three cases is summarized in Table 2. These values indicate that for the topology without feedback, the opportunistic policy provides an improvement in the reliability over the conservative policy. For the topology with feedback, a significant degradation in the reliability does not occur due to the multiple processing iterations. In fact, the application reliability with feedback is close to the reliability of the conservative policy without feedback.

Table 1. Filter conditional probabilities

Filter	$C\|C$	$I\|C$	$N\|C$	$C\|I$	$I\|I$	$N\|I$
TL	0.95	0.00	0.05	0.00	0.00	0.00
IT	0.95	0.025	0.025	0.5	0.492	0.008
CA	0.95	0.03	0.02	0.5	0.492	0.008
PV	0.98	0.005	0.015	0.00	1.00	0.00
WF	1.00	0.00	0.00	0.00	1.00	0.00

Table 2. Summary of reliability estimates

Case	Reliability
Without feedback, conservative	0.8402
Without feedback, opportunistic	0.8519
With feedback (quality improvement)	0.8391

The reliability estimates reported in Table 2 were consistent with the expectations of the experts. Since the failure data from the actual application was never collected, a quantitative validation of these estimates could not be conducted. However, the most important use of these analytical expressions is at design time, where the focus is on analyzing the sensitivity of the application reliability to the filter/pipe reliabilities rather than obtaining a very accurate estimate. Sensitivity analysis can be used to guide resource allocation to improve the application reliability efficiently. Since the reliability estimates obtained from the methodology match the designer's expectations, it can enhance confidence in its use for such design-time analysis.

5 Related research

Prevalent architecture-based software reliability techniques can be mainly classified into path-based [8, 14] and state-based approaches [6, 9]. In the former, reliability is estimated by enumerating the execution paths through the application and this provides only an approximate estimate in the case of infinite loops. Ammar *et. al.* [3] consider error propagation in software architectures using this approach, but it does not connect error propagation probabilities to reliability. Cukic *el al.* [10] consider the impact of error propagation on reliability using this approach, but produces only an approximate estimate. In the state-space approach, a control flow graph represents the architecture,

and reliability is obtained analytically. It can account for the presence of infinite paths due to loops using analytical methods. Chen *et al*. [11] present a methodology to map the constraints of some architecture styles to the state space models. Abd-Allah [2] describe the use of reliability block diagrams to analyze the reliability of some architecture styles.

Existing analytical/state-space techniques cannot consider the impact of error propagation and possible downstream error correction. Secondly, the state-space models cannot account for a deterministic number of feedback iterations, with component reliabilities dependent on the number of iterations. The reliability analysis methodology proposed in this paper considers these two factors and hence can produce an estimate closer to the actual reliability.

6 Conclusions and future research

In this paper we presented a comprehensive reliability analysis methodology for the pipe and filter architecture style. We considered two variants of the style, namely, linear topology with and without feedback. We considered the impact of (i) error propagation and downstream error correction, and (ii) deterministic number of iterations of the feedback loop with filter reliabilities dependent on the number of iterations, on the application reliability. We illustrated the use of the methodology using a case study of a Document Analysis and Understanding Application.

The present research considers only point estimates of the pipe and filter reliabilities. In the future, we will develop a methodology to propagate the variances in the pipe and filter reliabilities to the variance in the application reliability. Our future work is also concerned with reliability analysis of other architecture styles.

Acknowledgments

This research is supported by EpSCoR and Workforce Development grants from Connecticut Space Grant Consortium, Large Grant from the University of Connecticut Research Foundation and the following grants from the US National Science Foundation (CNS-0406376 and CNS-SMA-0509271).

References

[1] http://cognet.mit.edu.

[2] A. Abd-Allah. “Extending reliability block diagrams to software architectures”. Technical Report USC-CSE-97-501, Dept. of Computer Science, University of Southern California, 1997.

[3] W. Abdelmoez, D. M. Nassar, M. Shereshevsky, N. Gradetsky, R. Gunnalan, and H. H. Ammar. “Error propagation in software architectures”. In *Proc. of Software Metrics Symposium*, pages 384–393, 2004.

[4] D. Garlan and M. Shaw. *Advances in Software Engineering and Knowledge Engineering, Volume 1, edited by V. Ambriola and G. Torotora*, chapter An Introduction to Software Architecture. World Scientific Publishing Company, New Jersey, 1993.

[5] S. Gokhale and S. Yacoub. “Performability analysis of a pipeline software architecture”. In *Proc. of Intl. Conference on Computer Science and Applications*, Edinburgh, Scotland, July 2005.

[6] K. Goseva-Popstojanova and S. Kamavaram. “Assessing uncertainty in reliability of component–based software systems”. In *Proc. of Intl. Symposium on Software Reliability Engineering*, pages 307–320, 2003.

[7] R. Kazman, G. Abowd, L.Bass, and P. Clements. “Scenario-based analysis of software architecture”. *IEEE Software*, pages 47–55, November 1996.

[8] S. Krishnamurthy and A. P. Mathur. “On the estimation of reliability of a software system using reliabilities of its components”. In *Proc. of Eighth Intl. Symosium on Software Reliability Engineering*, pages 146–155, Albuquerque, New Mexico, November 1997.

[9] J. C. Laprie and K. Kanoun. *Handbook of Software Reliability Engineering, M. R. Lyu, Editor*, chapter “Software Reliability and System Reliability”, pages 27–69. McGraw-Hill, New York, NY, 1996.

[10] P. Popic, D. Desovski, W. Abdelmoez, and B. Cukic. “Error propagation in the reliability analysis of component based systems”. In *Proc. of Intl. Symposium on Software Reliability Engineering*, pages 53–62, Chicago, IL, 2005.

[11] W. Wang, Y. Wu, and M. H. Chen. “An architecture-based software reliability model”. In *Proc. of Pacific Rim Dependability Symposium*, pages 143–150, Hong Kong, December 1999.

[12] L. G. Williams and C. U. Smith. “PASA: A method for the performance assessment of software architectures”. In *Proc. of the Third Intl. Workshop on Software and Performance*, pages 179–189, 2002.

[13] S. Yacoub. “Automated QA for document understanding systems”. *IEEE Software*, May 2003.

[14] S. Yacoub, B. Cukic, and H. Ammar. “A scenario-based analysis for component-based software”. *IEEE Trans. on Reliability*, 53(4):465–480, 2004.

SOPHIANN: A tool for extraction knowledge rules from ANN previously trained – A case study.

M. Song , L. Zárate, S. Dias, A. Alvarez, B. Soares, B. Nogueira, R. Vimieiro, T. Santos, N. Vieira
UNA – Centro Universitário
Belo Horizonte, Minas Gerais, Brazil
{mark, zarate}@una.br

***Abstract* - Artificial Neural Networks (ANN) are widely used with several purposes. Once trained, they are capable to solve unprecedented situations, keeping tolerable errors in their outputs. However, humans cannot assimilate the knowledge kept by those nets, since such knowledge is implicitly represented by their connection weights. Formal Concept Analysis (FCA) can be used in order to facilitate the extraction and understanding of rules described by ANN. In this work, a method to extract rules via FCA is discussed and a tool, SOPHIANN that implements all the stages necessary to the knowledge extraction process, is presented. The tool can be used to understand the relationship among the process parameters through implications rules. Also, a case study based on a solar energy system application is presented.**

I. INTRODUCTION

Understanding real world processes is not an easy task. Real problems, especially those related to industrial processes, are difficult to model and understand. They involve a high complexity level in their domains making the knowledge process very hard. In this context, Artificial Intelligence (AI) methods have been proposed as an alternative to represent human knowledge in computer systems. AI has two main ways: symbolism and connectionism. The symbolic field deals with symbolic or explicit knowledge. Even though this kind of systems presents numerous advantages (e.g. facility in explaining its knowledge), the knowledge acquisition is considered a bottleneck of such systems. Besides, symbolic systems are difficult to work with numerical data, known as implicit or sub-symbolic knowledge.

The connectionist field of AI has been proposed as an alternative to deal with sub-symbolic data. The main representative element is Artificial Neural Network (ANN). An ANN has the capacity to obtain the relationship as a function between the input and output parameters where input parameters are mapped to outputs. However, this behavior turns ANN a "black box" and no explanation can be attributed to it in the decision process. Thus, methods to extract knowledge from ANN are necessary. This knowledge can be extracted in form of problem domain rules. The rules extracted from neural networks promote the knowledge of the problem in its real context through the relation between the variables of its domain. Consequently, this rule extraction can be used in the training of human elements that are less experienced in relation to complex processes, specially the industrial ones.

Many researchers have been discussing the knowledge extraction from ANN [1], [2], [3]. Different methodologies have been presented for rule extraction. Some methods extract rules through the analysis of the net structure (i.e. weights, topology, etc), while others extract rules analyzing the data set used to train the net. There are also methods that use both the net structure and the data set to extract rules.

Formal Concept Analysis (FCA) has been proposed as a powerful technique in knowledge representation and extraction. In [4], FCA has been used to extract rules from previously trained neural networks. The method presented in [4] consists in a visual analysis of the line diagram in order to extract the rules. This method has not been considered a satisfactory approach, because the diagram can be complex. Then, the extraction of all the rules is a difficult task and, consequently, some important rules may be forgotten. In this work, an alternative approach to extract and represent the knowledge of previously trained neural networks is presented. The main idea of this new approach is join Formal Concept Analysis and the Next Closure algorithm to extract knowledge from ANN.

Besides, a computational tool (SOPHIANN), developed to support this approach, is presented. In SOPHIANN rules are extracted from neural networks that are representative to the domain of a process. The rules can be shown in "IF… THEN…" form or through a lines diagram representing the relation of the concepts involved in the process being studied. It is also shown an application of this method through use of SOPHIANN in an industrial process of solar energy.

This paper is organized in seven sections. In the second one, formal concept analysis is briefly described. In the third, some important considerations about closure systems are presented. In the fourth and fifth sections the approach and SOPHIANN tool to extract knowledge from ANN are respectively presented. After that, a case study is presented. Finally, some conclusions of this work are presented.

II. FORMAL CONCEPT ANALYSIS

Formal Concept Analysis (FCA) is a field of mathematics presented in the early eighties [5], [6]. Its main goal involves the knowledge representation by means of specific diagrams called line diagrams.

In FCA, Formal contexts are a primordial definition. They are represented by cross tables. Formal contexts have the notation (*G, M, I*), where *G* is a set of objects (rows

headers), M is a set of attributes (columns headers) and I, an incidence relation ($I \subseteq G \times M$). If an object $g \in G$ and an attribute $m \in M$ are in the relation I, this is represented by $(g, m) \in I$ or gIm and is read as "*the object g has the attribute m*".

Even though this definition of formal contexts is valid for many situations, mainly to represent objects that have the presence or absence of some properties (attributes), it is not a good representation for the major part of situations, where objects have attributes that can take on several values. For this case, attributes are called as many-valued attributes. So, contexts, where the set M of attributes is composed by many-valued attributes, are called *many-valued contexts*. Now, the notation of the formal context is given by the quadruple (G, M, V, I), where G is still a set of objects; M is a set of many-valued attributes; V the set of possible values for the attributes; and I is a ternary relation between G, M and V ($I \subseteq G \times M \times V$) [7]. In general, many-valued contexts can be transformed into single-valued contexts in order to obtain the formal concepts. A simple way to do such transformation is to replace each many-valued attribute by the corresponding attribute-value pair, as the example presented by Table I. In Table I T^i_j represents the pair attribute-value. Note that the number w_i of attribute-value pairs for each many-valued attribute is given by the cardinality of the set V (i.e. $|V|$). It is important to remember that the attributes can have continuous values, for example, V may be equals to the set $\Re$ of real numbers. In this case, there will be as many attribute-value pairs as the cardinality of the set $\Re$, i.e. infinity. So, for this case, may be interesting to choose intervals as values for the attributes. For example, considering V corresponding to the values of the Celsius scale and {*water temperature*} as the set M of attribute, it is more suitable to choose V as ranges of possible values in the Celsius scale (i.e. *value*#1 = [0ºC; 10ºC >; *value*#2 = [10ºC; 20ºC >, and so on).

Since a cross table, representing a formal context is given, algorithms can be applied to it in order to determine its formal concepts and its line diagram. Formal concepts are pairs (A, B), where $A \subseteq G$ (called extent) and $B \subseteq M$ (called intent). Every element of the extent (object) has all the elements of the intent (attributes) and, consequently, every element of the intent is an attribute of all objects of the extent.

TABLE I
MANY-VALUED CONTEXT

	Attribute #1			···	Attribute #n		
Objects	T^1_1	···	$T^1_{w_1}$	···	T^n_1	···	$T^n_{w_n}$
1	×			···			×
2			×	···	×		
···	···						

A. Derivation Operators

Given a set $A \subseteq G$ of objects from a formal context (G, M, I), it could be asked which attributes from M are common to all of those objects. Similarly, it could be asked, for a set $B \subseteq M$, which objects have the attributes from B. These questions defines the derivation operators, which are formally defined as:

$$A^{\uparrow} \leftarrow \{m \in M \mid gIm \; \forall \; g \in A\} \quad (1)$$
$$B^{\downarrow} \leftarrow \{g \in G \mid gIm \; \forall \; m \in B\} \quad (2)$$

III. CLOSURE SYSTEMS

The definition of closure systems is very simple, despite of not being very intuitive [4]: closure systems are sets of sets. For clarity, the notation of elements are given by small latin letters, the notation of sets are capital latin letters and for closure systems calligraphic letters.

Given a set M, a closure system C on it can be formally defined as:

- $C \subseteq \mathfrak{B}(M)$,
- $M \in C$,
- $D \subseteq C \rightarrow \cap D \in C$.

where, $\mathfrak{B}(M)$ is the power set of the set M and $\cap D \leftarrow \{x \mid \forall S \in D \; x \in S\}$.

A. Closure Operators

In [4] closure operators are defined as a map assigning a closure $\varphi X \subseteq M$ to each set $X \subseteq M$. Closure operators have the following characteristics:

Monotone if $X \subseteq Y$ then $\varphi X \subseteq \varphi Y$
Extensive $X \subseteq \varphi X$
Idempotent $\varphi\varphi X = \varphi X$

In this work, the derivation operators will be considered as closure operators, since they are idempotent ($X'''' = X''$), extensive ($X \subseteq X''$), and monotone ($X \subseteq Y \rightarrow X'' \subseteq Y''$). In this work, sets are called closed if $B'' = B$ ($B \subseteq M$).

IV. METHOD TO EXTRACT KNOWLEDGE FROM ANN

In this section, the steps to extract knowledge from neural nets are presented. It is needed to:

1) Select a process representative data set in order to train the ANN. It is defined as:

$$X = [x_{ij}]_{m \times n} \quad (5)$$

where n is the number of parameters; X_{ij} to $i = 1,\ldots,m$ and $j = 1,\ldots, n-1$ are the input parameters and X_{in} to $i = 1,\ldots,m$ is the output parameter. This output parameter presents a normal distribution $N_1(\bar{x}, S(x))$.

2) Define the structure of the multi-layers neural net (with N input parameters; H hidden layers and M outputs), and train it. In this work $M = 1$ for any situation.

3) Build a synthetic data set. To extract knowledge from the net, a synthetic data set has been generated considering the domain ranges of the input parameters. This synthetic data set is defined as:

$$Y = [y_{ij}]_{p \times n-1} \quad (6)$$

where *Y* has only elements generated with the purpose of knowledge extraction. Thus, no one of such elements has been collected from the process. Each input parameter has minimal and maximal values, which defines the domain range of each parameter. The minimum and maximum values are vectors that are respectively defined as:

$$Inf = [u_j], j = 1, ..., n-1 \quad (7)$$
$$Sup = [v_j], j = 1, \ldots, n-1 \quad (8)$$

The number of elements that will be generated in the interval between the minimum and maximum value is defined by the vector W:

$$W = [w_j]; j = 1, ..., n-1 \quad (9)$$

Hence, the variation of each parameter (in the interval) that will compose the synthetic data set is represented by the following expression:

$$Int = [I_j] = \frac{| v_j - u_j |}{w_j - 1} \quad (10)$$
$$w_j > 1 \text{ and } j=1,\ldots..,n-1$$

It could be observed that the number *p* of sets that will be generated depends on the number of intervals of each parameter. So *p* can be defined as:

$$p = w_1 \times w_2 \times \ldots \times w_{n-1} = \prod_{i=1}^{n-1} w_i \quad (11)$$

4) Present the synthetic set *Y* to the net in order to obtain the output parameter $Z = [z_{ij}]_{p \times 1}$ which has a normal distribution $N_2(\bar{x}, S(x))$. Verify the net generalization through the comparison of the distribution $N_1(\bar{x}; S(x))$ and $N_2(\bar{x}; S(x))$. If $e_x = |\bar{x}_1 - \bar{x}_2|$ and $e_{S(x)} = |S(x)_1 - S(x)_2|$ represent high errors, then, back to step 1.
5) Classify the parameters (columns) of the matrix $U = [Y,Z]_{p \times n}$ into intervals. As the data considered by this method has continuous values, the better context to represent this data is a many-valued context (section II). So data should be classified into ranges as presented in section II.
6) Build a *formal context cross table*. The classification in intervals of *n* variables of the objects of domain establishes a binary relationship between objects and attributes, named incidence, where an object has or not an attribute. The *formal context* is characterized by the set of these objects/attributes incidence relations, as shown in Section II. Due to these characteristics, formal contexts are easily visualized through *attributes vs. objects tables*, named cross table [6], where the incidence between an object A and an attribute B is represented by the correspondent cell w_{AB} marking.
7) Obtain the formal concept. It is possible to determinate the formal concepts, which are the ordered pairs (*Object, Attribute*), from the formal context. The object set is named *extension* and attributes set is named *intention*. Each extension element has all intention elements, with the reciprocal being true. In any formal concept, all objects of a same extension have all attributes from a same intention.
8) Apply the Next Closure algorithm in order to obtain the implication rules of the type: *If … Then …* .

V. THE SOPHIANN TOOL

In order to validate the methodology an open source tool, named SOPHIANN has been developed under the object-oriented paradigm. The tool presents an easy insertion of new algorithms and functionalities without the necessity of any system internal structure alteration. For example, SOPHIANN enables the user to incorporate new efficient algorithms in modules like Discretization, Formal Concept and Rules Extraction.

The proposed approach is entirely supported by SOPHIANN, from previously trained ANN information recovering (weights, topology, parameters minimum and maximum values, etc) to the rule extraction and visualization on a comprehensible way to the user, as shown in Fig. 1.

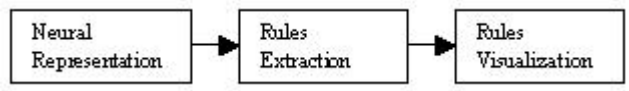


Fig. 1 Steps developed by SOPHIANN

Through the extracted rules visualization, the human element can reflect about them, building knowledge about the real process.

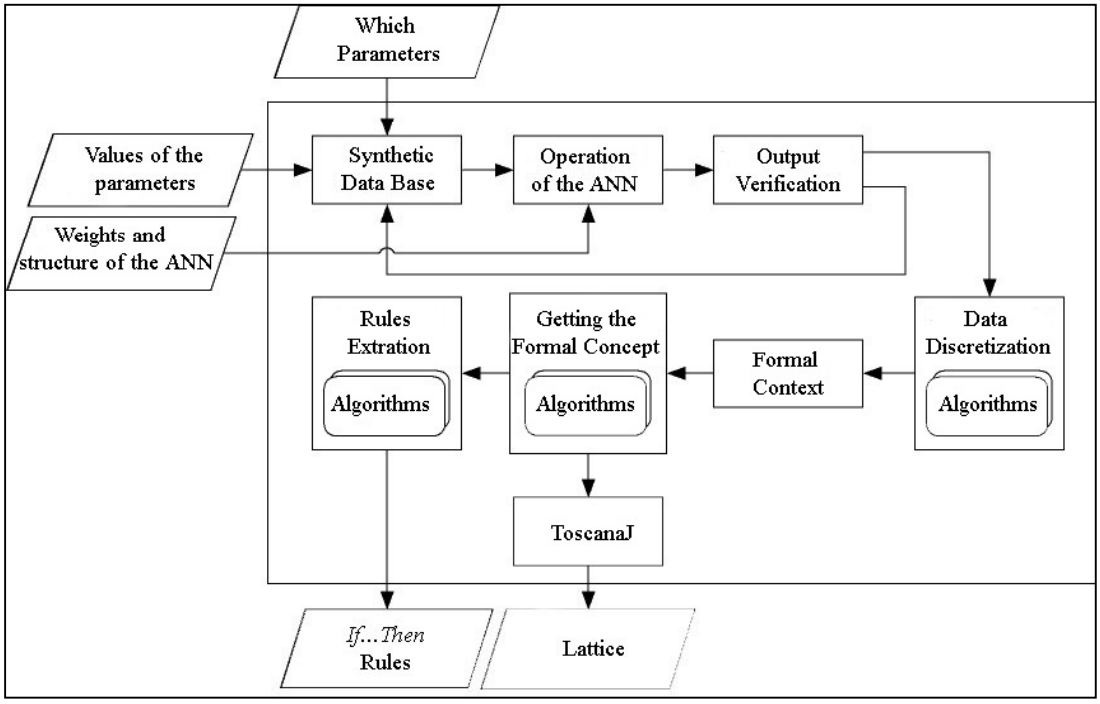


Fig. 2 SOPHIANN modules

The SOPHIANN tool, shown in Fig. 2, integrates all the resources used in the new approach. In SOPHIANN, rules are extracted from neural networks, which are representative to the domain of a process. The rules can be shown in the "IF… THEN…" form or through a line diagram representing the relation of the concepts obtained from the process parameters – in order to obtain the tool contact zarate@pucminas.br. Each module of SOPHIANN will be described in the next section as a case study is presented.

VI. Case Study – Solar Energy System

Solar energy systems, in particular water heaters, have considerable importance in the substitution of traditional electrical systems. The most widely used solar energy system, solar collector, has been explored in many researches [9], [10]. In this work, solar collectors are used as a practical example in the process of knowledge extraction.

Initially, the solar collector subsystem has been modeled using a ANN considering the input water temperature (T_{in}), solar irradiance (G) and environment temperature (T_{amb}) as inputs of the net, and output water temperature (T_{out}) as its output (see Table II) . For the training process, 631 data have been collected - 30 of them used in the validation stage where the trained net produced satisfactory results to validate the process. An important aspect in this stage is the normal distribution of values. In the training process it has been used normalized data from 0.2 to 0.8.

Table II – Minimum and maximum parameter values

	Tamb(ºC)	*Tin(ºC)*	*G(W/m2)*	*Tout(ºC)*
maximum	27,18	60,81	1084,95	64,73
minimum	22,48	20,29	731,71	26,42

After the training process concludes a synthetic data set is generated through the Synthetic Database module. Using these data the system rebuilds the ANN and generates an output set based on these data - the output is statistically validated before their discretization. Table III shows a synthetic data sample.

Table III – Synthetic Data Sample

Tamb(ºC)	*Tin(ºC)*	*G(W/m2)*	*Tout(ºC)(Net)*
22.48	20.29	731.71	26.38
22.82	28.97	933.56	35.33
23.15	20.29	958.79	28.42
24.16	40.55	933.56	46.79
24.159	40.55	958.79	46.96
25.17	31.87	857.87	37.70
26.17	40.55	984.02	47.26
27.18	46.34	731.71	51.29
27.18	46.34	756.94	51.47
27.18	60.81	1084.95	64.31

Note that the statistical distribution of both original and synthetic data are acceptable – the difference between the average and standard deviation values of the original and synthetic data follow approximately the same normal distribution ($ex = | x_1 - x_2 |$ e $eS(x) = |S(x)_1 - S(x)_2|$). Table IV compare the synthetic and original values of *Tout(ºC).*

Table IV – Data Sets Statistical Distribution

	Tout(ºC) original	*Tout(ºC) synthetic*
Average(X)	36,55	46,35
Standard Deviation (S(x))	8,77	11,46

Discretization Module

In this stage data values are divided in ranges. Although there are many researches in that area, the numbers of ranges is still an open question [12]. SOPHIANN allows the user to plug-in new algorithms in order to divide the ranges properly.

In this case study the algorithm has been implemented according to Stugart formula [5] - it has been used 2 ranges as shown in Table V.

Table V – Attributes Created According to Values Ranges

	Ranges
Tamb(ºC)	[22.48, 24.83 > [24.83, 27.18]
Tin(ºC)	[20.29, 40.55 > [40.55, 60.81]
G(W/m2)	[731.71, 908.33 > [908.33, 1084.95]
Tout(ºC)	[26.42, 45.575 > [45.575, 64.73]

Formal Context Module

In this module SOPHIANN determines the incidence relation according to the discretization values. As a result data sets are converted into a cross table or formal context (table VI).

Table VI – Cross Table

index	*Tamb(ºC)*	*Tin(ºC)*	*G(W/m2)*	*Tout(ºC)*
1	1	1	1	1
2	1	1	2	1
3	1	1	2	1
4	1	2	2	2
5	1	2	2	2
6	2	1	1	1
7	2	2	2	2
8	2	2	1	2
9	2	2	1	2
10	2	2	2	2

Formal Concept Module

In this module SOPHIANN generates formal concepts as described in [6]. In order to simplify the interpretation a lattice is also produced (Figure 3) - a lattice is a graph where vertex are formal concepts and edges are their hierarchical relationships [7].

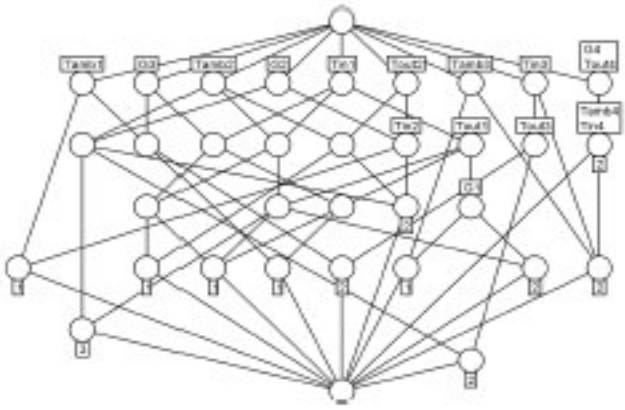


Fig. 3 – A lattice example

Rules Extraction Module

In order to extract the rules the Next Closure algorithm [8] has been implemented. Table VII shows some rules obtained in the process.

Table VII – Rules

1	***If***	Tout = 2	***Then***	Tin = 2			
2	***If***	Tin = 1	***Then***	Tout = 1			
3	***If***	Tin =2	***And***	G = 2	***Then***	Tout =2	
4	***If***	G =2	***And***	Tout =1	***Then***	Tin = 1	
5	***If***	Tin = 2	***And***	Tout =1	***Then***	G = 1	

In order to validate the rules an original data set has been used:

Tamb(ºC)	**Tin(ºC)**	**G(W/m2)**	**Tout(ºC)**	**RULE**
23,79	20,29	740,92	26,42	2
24,51	22,53	859,54	29,24	2
24,83	23,77	731,71	29,78	2
24,51	22,88	940,35	29,81	2,4
24,91	25,99	913,96	32,94	2,4
24,85	26,01	913,06	32,97	2,4
22,89	27,63	917,56	34,27	2,4
24,5	35,44	856,68	41,03	2
26,48	36,3	1084,95	43,80	2,4
27,18	38,12	1036,1	45,29	2,4
23,79	60,81	943,68	64,66	1,3
23,88	60,77	954,83	64,73	1,3
23,75	20,35	743,61	26,89	2

The rules presented in Table VII describe the solar collector. For example, it could be observed, according to rule 2, that when Tout is in interval 2, Tin will certainly belong to interval 2. Note that the number of ranges determines the number of rules extracted. The following example has been generated considering 4 ranges:

1 {Tin_[4]} ===> {Tout_[4]}
2 {Tout_[1]} ===> {Tin_[1]}
3 {G_[1] Tout_[4]} ===> {Tin_[4]}
4 {Tin_[1] G_[1]} ===> {Tout_[1]}
5 {G_[1] Tout_[2]} ===> {Tin_[2]}
6 {G_[2] Tout_[4]} ===> {Tin_[4]}
7 {Tin_[1] G_[2]} ===> {Tout_[1]}
8 {Tin_[3] G_[1]} ===> {Tout_[3]}
9 {G_[2] Tout_[2]} ===> {Tin_[2]}
10 {Tin_[3] G_[2]} ===> {Tout_[3]}
11 {Tamb_[1] Tin_[1] G_[3]} ===> {Tout_[1]}
12 {Tamb_[1] G_[3] Tout_[2]} ===> {Tin_[2]}
13 {Tamb_[1] Tin_[1] Tout_[2]} ===> {G_[4]}

As the number of rules increases it is easy to note that some of them might be included in others. For example rule numbers 3 and 6 (4 ranges) are related to rule number 1 (2 ranges) but defines a new condition based on a new process parameter. Considering the physical characteristic of the process one must not expect that all the rules obtained from 4 ranges can be obtained from 2 ranges. Note that most of the rules generate from 4 ranges where not obtained from 2 ranges. For example: Tamb_[2] Tin_[2] => Tout_[2] – *if Tamb = 2 And Tin =2 Then Tout =2*.

VII. CONCLUSIONS

Formal Concept Analysis has been presented as a new approach to knowledge representation and extraction. In this paper, FCA has been used as an alternative joint to extract rules from Artificial Neural Networks.

It has been also presented SOPHIANN, a computational tool that implements that approach. SOPHIANN is an open source tool, developed by LICAP [11] researchers under object-oriented paradigm on the Java language. This tool has modules responsible to implement each step of the knowledge extraction process. In some modules, SOPHIANN allows the user to aggregate newer algorithms.

On the other hand, the proposed approach is highly dependent of a satisfactory representation of the considered process through ANN. The method cannot compensate neural representation imprecision.

Another aspect that needs to be highlighted is about the used discretization method. A big interval number for each parameter influences on the obtained rules number. So, the specification of the intervals number can be considered as a regulator mechanism of the detail level of the obtained rules. However, this regulation is highly dependent of the neural representation quality of the details of data of the process being considered.

It is important to notice that the obtained rules depend on the synthetic data set generated, through which the output parameter is obtained via ANN. Therefore, this net, which is trained with few experimental data of the real process, is responsible by neural representation of the process and knowledge extraction.

ACKNOWLEDGEMENTS

The authors would like to thank the financial support of the National Research Council CNPq - Brazil. Project CT-INFO/ MCT/CNPq nº 031/2004, Process: 506512/2004-5.

REFERENCES

[1] M. Hilario, "An overview of strategies for neurosymbolic integration," in Hybrid Neural Systems, S. Wermter and R. Sun, Eds. Germany: Springer-Verlag, 2000.

[2] G. Towell and J. Shavlik, "The extraction of refined rules from knowledge-based neural networks," in Machine Learning Research Group Working, 1993, pp. 71–101.

[3] M. Craven, "Extracting Comprehensible Models from Trained Neural Networks", PhD Thesis, department of Computer Science, University of Wisconsin-Madison. 1996.

[4] R. Vimieiro, L. E. Zárate, J. P. D. Silva, E. M. D. Pereira, and A. S. C. Diniz, "Rule extraction from trained neural networks via formal concept analysis," Proceedings of the Eighth IASTED International Conference on Artificial Intelligence and Soft Computing, A. del Pobil, Ed., IASTED. ACTA Press, 2004, pp. 334–339.

[5] B. Ganter, "Formal concept analysis: algorithmic aspects," TU Dresden, Germany, Tech. Rep., 2002.

[6] B. Ganter and R. Wille, Formal concept analysis: Mathematical foundations. Springer Verlag, 1996.

[7] G. Gratzer, "General lattice theory", 2nd ed. Birkhäuser Verlag, 1998, ch. Applied lattice theory: Formal concept analysis, pp. 591–605.

[8] C. Carpineto and G. Romano, Concept Data Analysis: Theory and Applications. John Wiley & Sons, 2004.

[9] L. E. Zárate, E. M. D. Pereira, D. A. Soares, J. P. D. Silva, R. Vimieiro, and A. S. C. Diniz, "Optimization of neural networks training sets via clustering: Application in solar collector representation," in Proceedings of 6th International Conference on Enterprise Information Systems, I. Seruca, J. Filipe, S. Hammoudi, and J. Cordeiro, Eds., vol. 2, ICEIS. Porto, Portugal: INSTICC Press, April 2004, pp. 147–152.

[10] L. E. Zárate, E. M. D. Pereira, J. P. D. Silva, R. Vimieiro, and A. S. C. Diniz, "Neural representation of a solar collector with optimization of training sets," in Lecture Notes in Artificial Intelligence, B. Orchard, C. Yang, and A. Moonis, Eds., ISAI, AAAI, ACM/SIGART, INNS. Springer-Verlag, 2004.

[11] LICAP, "Applied Computational Intelligence Laboratory", December 2005, http://www.inf.pucminas.br.

[12] Yang, Y. and Webb, G. I. Discretization for data mining.

Proceedings

International Workshop on Agent-Oriented Software Development Methodology (AOSDM 2006)

Co-Editors

Hong Zhu, Oxford Brookes University, UK
Huaglory Tianfield, Glasgow Caledonian University, UK
Hongji Yang, De Montfort University, UK

The Dynamic Casteship Mechanism for Modeling and Designing Adaptive Agents

Xinjun Mao[1], Zhiming Chang[1], Lijun Shang[1], Hong Zhu[2] and Ji Wang[1]
[1]*Dept. of Computer Science, National University of Defense Technology, Changsha, China, 410073*
Email: xjmao@nudt.edu.cn, wj@nudt.edu.cn
[2]*Dept. of Computing, Oxford Brookes University Oxford OX33 1HX, UK*
Email: hzhu@brookes.ac.uk

Abstract. It is still a challenge to develop complex multi-agent systems with agents that typically exhibit adaptation to various behaviors in different situations. In this paper, we propose an extension of the dynamic casteship mechanism for modeling and designing adaptive agents. In our approach, caste is the modular unit in the design and implementation of multi–agent systems and provides an abstraction of the ways that agents adapt to different behaviors in different situations. The dynamic behaviors of agents are realized as the change of castes that agents bind during execution by "*join*" and "*quit*" operations on agent's casteships. The extended mechanism also enables agents to change the status of its castes to be either *active* or *inactive* at run-time. Accordingly, two new operations '*deactivate*' and '*activate*' on agent's casteships are introduced. In the paper, the formal semantics of these operations is rigorously defined. The properties of dynamic binding mechanism are investigated.
Keywords: adaptive agent, caste, dynamic binding

1. Introduction

Agent orientation has recently emerged as a new software development paradigm for developing complex software systems that operates in dynamic environment such as the Internet [6]. In the past few years, with the increasing acceptance and expectation of agent orientation as an emerging software engineering paradigm, there have been a great number of efforts in the research on development methodologies of complex multi-agent systems (MAS). Agent-oriented software engineering (AOSE) has become an active research area in agent-based computing. A number of methodologies, programming languages and CASE tools or environments have been proposed, such as Gaia[5], Tropos[11], CAMLE[3], etc.

Nevertheless, several challenges need to be faced before AOSE can deliver its promises and become a widely accepted and practically usable paradigm for the development of complex systems [9] that are typically unpredictable, open, dynamic, hierarchically structured but confined by global constraints [1]. We believe that, to be a successful general-purpose software engineering paradigm, language facilities must be developed to enable high level abstractions to be smoothly and naturally transformed into concrete language facilities and efficiently implemented in a systematic, robust, reliable and repeatable fashion [4]. Moreover, the methods must be universally applicable. This paper investigate the language facility caste proposed in [3] and [12] for the design and implementation of dynamic behavior adaptation in MAS. In the sequel, we will use *adaptive agents* to denote agents that are capable of adapting to different behaviors at runtime.

Agents executing in a dynamic environment often have to behave differently in different situations. For example, frequent role changes are common for agents in human societies and organizations. However, existing agent-oriented methodologies have not provided enough supports to develop such systems that naturally take such advantages, especially in the modeling, specification, design and implementation of such dynamic behaviors in a systematic way. It is no surprise that it is extremely difficult to develop MAS with adaptive behaviors [12]. This paper addresses this problem by the dynamic binding mechanism based on the language facility caste proposed in [4,12].

The remainder of the paper is organized as follows. Section 2 introduces an example of adaptive agents. In section 3, caste is defined and dynamic binding mechanism is informally discussed. Section 4 rigorously defines the formal model of MAS and the formal semantics of four atomic operations on castes. The properties of dynamic binding mechanism are also investigated in this section. Finally section 5 concludes the paper.

2. An example of adaptive dynamic MAS

In order to understand the requirements on the dynamic binding mechanism, in this section, we examine an example of information system to illustrate the basic characteristics of complex systems that consist of adaptive agents.

Suppose that an information system is to be developed for a university to support the management of the university's students, staff and related affairs such as registration, course selection, etc. and to provide personal assistances to the members of the university so that each member can participate in the operations of the university efficiently and effectively. Depending on the type of roles that a member plays in the university, a member can take

certain actions in the operation of the information systems, access certain subset of the information stored in the systems, and must obey certain set of rules that confines the member to fulfill his tasks. Each member of the university is then supplied with a 'personal assistant', which is a software agent that stores the personal information about the member, the permitted actions that the member can take according to his role, as well as the personal preferences and private information. The personal assistant agent must also support the collaborations between the members.

As in almost all organizations, a member of the university may change its roles while remaining as a member of the organization. For example, an undergraduate student, say Alex, who is going to graduate from the university. After graduation, he gets an offer from the university to study for a Master degree, which will take him two years to complete. The graduation does not mean that he will change his identity. It just manifests that he will play a new role in the organization of the university. By changing role, he will no longer be an undergraduate since then and start to be a postgraduate. This means he must give up certain access to the information, certain actions that he was granted as an undergraduate student as well as certain relationships with other members of the organization such as his personal tutor, etc. This will also give him some new capability of actions, such as access to the master degree student labs, new personal information, a master degree student ID number, and new relationships to other members, such as a professor as his supervisor, etc. The change of Alex's role in the university must also be reflected in his personal assistant software agent in terms of the changes of the restrictions on his access to the information system, and the set of permitted actions, etc. What is important is that the personal assistant agent must carry over all the personal information such as his academic records, personal details and personal preferences, etc. rather than start from scratch.

Role changes of occur not only as moving from one role to another, but also as in the form of temporarily leaving from a post and then returning to the role to continue the job. For example, after one year's study, Alex decides to take one year industrial placement job in order to gain some work experience. He thus leaves the university for one year and then comes back to school and carries on his study. This does not mean that Alex will retreat from the role of postgraduate student for ever. Instead, during the work placement year, he suspends his behavior as postgraduate temporarily. He is officially still a registered postgraduate during this year, but he will not go to the university to take courses or conduct researches. The information system should support Alex to suspend and resume his study by keeping all personal information such as the study program, scores of passed or failed modules, credits earned, etc. suspending the permitted activities as well as some restrictions on his performances, etc. while he is on leave, and resuming his normal postgraduate student behavior after returning to the university. The dynamic binding mechanism that supports the changes between roles must also be able to support such temporary suspension of a role and resumptions of a role after suspensions.

As in almost all organizations, a member of the university may also play multiple roles at the same time. For example, when Alex becomes a postgraduate, he also takes a teaching assistant (TA) job, which is a part of his offer of the postgraduate studentship. Therefore, in his first year, Alex is not only a postgraduate but also a TA. Alex's personal assistant agent must be able to help Alex on two different roles. The dynamic binding mechanism should be able to support the taking of new roles while stay in a role.

3. Caste and Dynamic Binding Mechanism

The notion of caste was originally introduced in SLABS. It helps to deal with the limitation of object technology [4]. We regard caste as basic abstraction to specify agents' behaviors and the modular unit to design and implement multi-agent systems.

In our meta-model of MAS, agent is defined as an autonomous and persistent computational entity situated in certain environment. Each agent consists of four parts: data, actions, behaviors and environments. That is, the structure of an agent is a 4-tuple, i.e., *Agent* = <*D*, *A*, *B*, *E*>, were *D* represents an agent's state space, *A* is the actions that agent can take, *B* is the behavior rules that determine how agent behaves in the context of its designated environment *E*. It is worth noting that these parts may change during the execution of the agent. Such changes are realized by the dynamic binding of an agent to castes.

A caste defines a set of structural features and behavior patterns for agents. A caste is a 4-tuple, i.e., *Caste* = <*D*, *A*, *B*, *E*>, where *D* defines a state space that the agents can have, *A* defines a set of actions that agents can take for the operation on the state space, *B* defines a set of behavior rules specifying how agents behave in terms of when to take an action and how to update its state in the context of their designated environment, and *E* defines an environment, which consists of a set of agents that have influence on the agent's behavior [12]. When an agent binds to a caste, it obtains the state space and the capability to take an action defined by the caste, becomes in collaboration with and influenced by the agents specified in the caste definition and must obey the behavior rules specified in the caste. For example, in the above example, we can define two castes to represent the roles of undergraduate and postgraduate students, respectively.

A state space defined by a caste is represented as a set of state variables. Each action consists of an action identifier and may contain a number of parameters. The state space and the set of actions are usually divided into two kinds: visible ones and invisible (or internal) ones. When an agent that binds to the caste takes a visible action, it generates an event that can be observed by other agents in the system. An agent taking an internal action generates an event that can only be perceived by its components, which are also agents. Similarly, the value of a visible data can be observed by other agents, while the value of an internal

state can only be observed by its components. For the sake of simplicity, in the sequel, we assume all the actions and state variables are visible because, in this paper, we are not concerned with the visibility issues.

Although there are similarities with the relationship between object and class and the relationship between data and type, none of the terms like *instance*, *member, classifier* and *type* accurately represents the relationship between agent and caste. The main difference is that an agent can take actions to join a caste or retreat from a caste at run-time. Consequently, it obtains (or lose) the structural and behavioral features defined by the caste. Hence, the relationship is called casteship.

It is worth noting that an agent's action of joining or quitting a caste is also determined by the behavior rules defined by the caste that the agent currently binds to. Therefore, the behavior rules specified in each caste explicitly declare what castes the agent can *join* and *quit* and in what situation to do so. In the above example, Alex graduates and becomes a postgraduate student must be the result of executing a behavior rule that enables an undergraduate to quit from the caste *Undergraduate* and to join the caste *Postgraduate*.

However, the existing mechanism of dynamic binding of agents to castes is still insufficient in the modeling of adaptive agents as discussed in section 2. For example, it can not distinguish the castes in active status from ones in inactive status. To overcome the drawbacks, this paper extend the dynamic binding mechanism by introducing the notion of active binding and inactive binding of an agent to a caste.

(1) Active binding

An agent's binding to a caste is active means that the agent obtains all the structural and behavioral features defined by the caste. In other words, the agent can take actions defined in the action part of the caste according to the behavior rules defined by the caste, which can be a change to the agent's internal state that belongs to the part of the state space defined by the caste.

(2) Inactive binding

An agent's binding to a caste can also be inactive, which means that the agent can not change the state variables and cannot take actions defined in an inactive caste while it still maintains its values of the state variables defined by the caste. The behavior rules of the inactive caste will not affect the agent's behavior. It does not observe the agents in its environment defined by an inactive caste either. However, the state variables defined by an inactively bounded caste are accessible. Moreover, when the agent becomes actively binding to the caste again, the values of the state variables are resumed to that when the agent's binding to the caste last changed to inactive. This is different from that agent retreats a caste.

An agent can change its casteship status by taking action *deactivate* to become inactive and *activate* to become active. When agent takes action to *join* a caste, the casteship will be in active state.

4. Formal model of dynamic casteship mechanism

In this section, we formally define the extended dynamic binding mechanism. We will first define the model of multi-agent systems.

4.1. Model of Multi-Agent Systems

In [12], the caste-centric formal model of MAS has been formally defined. The following extends the formal model to enable active and inactive dynamic binding of agents to be specified. For the sake of simplicity, we have omitted some aspects, such as the inheritance relationship between castes and scenario definition, of the original model so that we can focus on the dynamic casteship mechanism.

Definition1. A model M of MAS consists of two parts < MAS, CASTE>, where MAS = $\{a_1, a_2, ..., a_n\}$ is a finite set of agents and CASTE = $\{c_1, c_2, ..., c_m\}$ is a finite set of castes.

Agents in MAS behave continuously and autonomously. A time moment is an element in a time index set T, which is defined as the set of natural numbers. Let MAS_t be the set of agents in the system at time moment *t*. The casteship of agent *a* at time moment *t* to caste *c* is denoted by $a \in_t c$. We write CASTE(*a*, *t*) to denote the set of castes that agent *a* belongs to at moment *t*, i.e. CASTE(*a*, *t*) = $\{c \mid a \in_t c\}$. In our dynamic binding mechanism, the castes that agent binds may be either in active state or inactive state. Therefore, we write $CASTE^A(a, t)$ to denote the set of active castes that agent *a* belongs to at moment *t*, and $CASTE^I(a,t)$ to denote the set of inactive castes that agent *a* belongs to at moment *t*. Thus, CASTE(*a*, *t*) = $CASTE^A(a, t) \cup CASTE^I(a, t)$. We assumes that $CASTE^A(a, t) \cap CASTE^I(a, t) = \varnothing$, for all agents *a* in MAS and time moments *t*. This means that for at any moment *t*, a caste that an agent binds is either in active state or in inactive state.

Agent's state space at some moment depends on the casts that agent binds to at that time moment and their status. Let $D^A_{a,t} = \cup_{\{c \in CASTE^A(a, t)\}} Dc$ denote the state spaces of agent *a* at moment *t* based on active castes that it binds to at moment *t*, $D^I_{a,t} = \cup_{\{c \in CASTE^I(a, t)\}} Dc$ denote the state spaces of agent *a* at moment *t* based on inactive castes that it binds to at moment *t*. Similar to the above, we define $A^A_{a,t} = \cup_{\{c \in CASTE^A(a, t)\}} A_c$ the action set of agent *a* at moment *t* based on active castes that it binds to at moment *t*, $A^I_{a,t} = \cup_{\{c \in CASTE^I(a, t)\}} A_c$ the action set of agent *a* at moment *t* based on inactive castes that it binds to at moment *t*. In particular, we assume that for any caste *c*, A_c includes "*join*", "*quit*", "*activate*" and "*deactivate*" operations on castes. Similarly, let $B^A_{a,t} = \cup_{\{c \in CASTE^A(a, t)\}} B_c$ the behavior rule set of agent *a* at moment *t* based on active castes that it binds to at moment *t*, $B^I_{a,t} = \cup_{\{c \in CASTE^I(a, t)\}} B_c$ the behavior rule set of agent *a* at moment *t* based on inactive castes that it binds to at moment *t*; $E^A_{a,t} = \cup_{\{c \in CASTE^A(a, t)\}} E_c$ the environment of agent *a* at moment *t* based on active castes that it binds to at moment *t*, $E^I_{a,t} = \cup_{\{c \in CASTE^I(a, t)\}} E_c$ the environment of agent *a* at moment *t* based on inactive

castes that it binds to at moment t.

Definition2. The state of agent a at moment t is based on the castes that it binds to and is defined as $s_{a,t} = s^A_{a,t} \times s^I_{a,t,}$ where $s^A_{a,t} = D^A_{a,t} \times A^A_{a,t} \times B^A_{a,t} \times E^A_{a,t}$ denoting the state of agent a at moment t based on active castes that it binds to, and $s^I_{a,t} = D^I_{a,t} \times A^I_{a,t} \times B^I_{a,t} \times E^I_{a,t}$ denoting the state of agent a at moment t based on inactive castes that it binds to.

Let $S_a = \{s_{a,t} \mid t \geq t_0$ for any moment t in T where t_0 is the moment at which agent a is to be created$\}$ the set of all possible configurations of agent a. The state of MAS at moment t is $S_{MAS,t} = \prod_{(a \in MAS)} s_{a,t}$. We write $S_{MAS} = \cup_{(t \in T)} S_{MAS,t}$ the set of all possible configurations of MAS.

Definition3. A run r of a MAS is a mapping from time T to the set S_{MAS}. The behavior of a MAS is defined by the set R of all possible runs. For any given run r of MAS, a mapping r_a from T to S_a is a run of agent a in the context of r, where for any moment t, $r_a(t)$ is the restriction of r(t) on S_a In the sequel, we use $R_a = \{ r_a \mid r \in R \}$ to denote the behavior of agent a in the system.

We assume that agent takes actions step by step, which means if agent takes action *act* at moment t, then the action will be completed at $(t+1)$ moment.

Definition4. For any $c_1, c_2 \in CASTE$, if the behavior rules of c_1 permit an agent that binds to caste c_1 to *join* caste c_2, then we call c_1 can directly reach c_2, denoted as $c_1 \Rightarrow c_2$.

We assume that the directly reachable relationship between castes is irreflexive, which means any caste is not permitted to be bound again when it has already bound. We write $DReach(c) = \{c' \mid c \Rightarrow c' \}$ to denote the directly reachable castes set of c, and $DReach(a,t) = \cup_{\{c \in \text{CASTE}^A(a,t)\}} DReach(c)$ to denote the directly reachable caste set of agent a at moment t. For example, if the behavior rule of caste *undergraduate* explicitly declares that when undergraduate student passes the entrance examination, he will join the caste of postgraduate, then $postgraduate \in DReach(undergraduate)$. The directly reachable relationship between castes does not satisfy transitive property. If c_1 can directly reach c_2, the agent that binds to caste c_1 is possible to *join* caste c_2 when the scenario and the pre-condition specified in the behavior rule are satisfied.

Definition5. Let $c \in CASTE$, the reachable castes set $Reach(c)$ of caste c is recursively defined as follows.

(1) if $c_1 \in DReach(c)$, then $c_1 \in Reach(c)$.

(2) if $c_1 \in Reach(c)$ and $c_2 \in Reach(c_1)$, then $c_2 \in Reach(c)$.

Obviously, the reachable relationship between castes is transitive. It defines the possible castes that agent can bind during its run. If c_1 can reach c_2, then the agent that binds to caste c_1 is possible to *join* caste c_2 in its run. Let $Reach(a,t) = \cup_{\{c \in \text{CASTE}^A(a,t)\}} Reach(c)$ the reachable caste set of agent a at moment t.

Definition6. Let $c_1, c_2 \in CASTE$. If caste c_1 and c_2 are strictly not permitted for any agent a to bind to at the same time to govern the agent's behaviors simultaneously, then we say that caste c_1 and c_2 are conflict, written as $c_1 \uparrow c_2$. Let $V \subseteq CASTE$ be a subset of castes, if for all castes c_1, $c_2 \in V$, c_1 and c_2 are not conflict to each other, i.e. $c_1 \uparrow c_2$ is not true, then we say that the caste set V is consistent. For an agent a and moment t, if $CASTE^A(a,t)$ is consistent, we say that agent a is consistent on its casteships at moment t. If agent a is always consistent on its casteships at all time moments in its run, we say that agent a is coherent.

For example, if the university does not permit any student to be an *undergraduate* and *postgraduate* simultaneously, then the castes *undergraduate* and *postgraduate* are exclusive to each other. Hence, the caste set {*undergraduate*, *postgraduate*} is not consistent. It is obvious that the exclusiveness relationship between castes is reflexive, symmetric, but not transitive. As the casteship of an agent can change from time to time and agent is possible to *join* any caste in its reachable castes set, agents should avoid being inconsistent on its castes during its run. Therefore, since {*undergraduate*, *postgraduate*} is inconsistent, the agent that binds to *undergraduate* must firstly quit the caste *undergraduate* before it joins the caste *postgraduate*.

Definition7. For agent a in *MAS*, a is called *adaptive*, if and only if, there are time moments t_1 and t_2 in T $(t_1 \neq t_2)$ such that $\text{CASTE}^A(a,t_1) \neq \text{CASTE}^A(a,t_2)$ or $\text{CASTE}^I(a,t_1) \neq \text{CASTE}^I(a,t_2)$. Otherwise, agent a is called *static*.

In the above example, agent Alex is a typical adaptive agent.

Lemma1. Let t_0 be the moment that agent a is created, if $CASTE(a,t_0)$ is consistent and a is a static agent, then agent a is coherent. [1]

4.2. Formalizing Dynamic Binding Mechanism

In this section, we formally define the dynamic binding mechanism and investigate its properties. Formally, we write "$M \models_{r,t} \varphi$" to denote that the model M of MAS and its run r satisfies formula φ at moment t, and "$\models \varphi$" to denote that formula φ is valid for any model of MASs and their runs at all time. Let "$< >$" be the dynamic operator to represent action execution, intuitively, "$<a: act>$" indicates that agent a takes action *act*. "$\boldsymbol{U}$" is the *until* temporal operator and "$\psi \boldsymbol{U} \varphi$" means that φ will be eventually satisfied and before that ψ is satisfied. "$\bullet$" denotes the next temporal operator.

Definition8. Formally, the semantics of the above temporal operators are defined as follows.

- $M \models_{r,t} \psi \boldsymbol{U} \varphi$ iff $\exists t' \in T$: $(t \leq t')$ and $(M \models_{r,t'} \varphi)$ and $(\forall t'': t \leq t'' < t' \Rightarrow M \models_{r,t''} \psi)$
- $M \models_{r,t} \bullet \varphi$ iff $M \models_{r,t+1} \varphi$

Definition9. $M \models_{r,t} <a: join(c)>$ iff $c \notin \text{CASTE}(a,t)$ and $\text{CASTE}^A(a,t+1) = \text{CASTE}^A(a,t) \cup \{c\}$ and $\text{CASTE}^I(a,t+1) = \text{CASTE}^I(a,t)$

Definition 9 means that agent a executes action "$join(c)$" at moment t successfully, if and only if at moment t, c is not the caste of agent a, and at moment $(t+1)$ agent a actively

[1] For the sake of space, the proofs of the theorems and lemma in the paper are omitted.

binds to castes c, and the action execution will not change the inactive castes of agent a. A number of special predicates are introduced to specify the castes of agent and their status. "BindCaste(a, c)" means that agent a binds to caste c, "Active(a, c)" denotes that agent a binds to caste c and it is in active status. "Inactive(a, c)" means that agent a binds to caste c and it is in inactive status. Formally, their semantics are defined as follows.

Definition10. For all agents a, castes c and moments t,

- $M \models_{r,t}$ BindCaste(a, c) iff $c \in CASTE(a, t)$
- $M \models_{r,t}$ Active(a, c) iff $c \in CASTE^A(a, t)$
- $M \models_{r,t}$ Inactive(a, c) iff $c \in CASTE^I(a, t)$

Theorem1. *join* operation has the following properties.

(1) $\models \langle a: join(c) \rangle \rightarrow \bullet$(Active($a$, c))

(2) $\models \langle a: join(c) \rangle \wedge$ Inactive(a, c_1)$\rightarrow \bullet$Inactive(a, c_1)

Property (1) means that if agent a *joins* some caste c at moment t, then the agent will bind caste c actively in the next moment. Property (2) means that the *join* operation will not change the inactive castes of agent.

Definition11. $M \models_{r,t} \langle a: quit(c) \rangle$ iff $c \in CASTE^A(a, t)$ and $CASTE^A(a, t+1) = CASTE^A(a,t) \backslash \{c\}$ and $CASTE^I(a, t+1) = CASTE^I(a,t)$

Agent a *quits* caste c at moment t, if and only if, agent a binds to caste c actively at moment t, and at moment $(t+1)$ agent a unbinds to castes c and the action execution will not change the inactive castes of agent a.

Theorem2. *quit* operation has the following properties.

(1) $\models \langle a: quit(c) \rangle \rightarrow \bullet$ ($\neg$BindCaste(a, c))

(2) $\models \langle a: quit(c) \rangle \wedge$ Inactive(a, c_1)$\rightarrow \bullet$Inactive(a, c_1)

Property (1) manifests that if agent a *quits* some caste c, then the agent will unbind to the caste c when action is completed. Property (2) means that the *quit* operation will not change the inactive castes of agent.

Definition12. $M \models_{r,t} \langle a: deactivate(c) \rangle$ iff $c \in CASTE^A(a, t)$ and $CASTE^A(a,t+1) = CASTE^A(a,t) \backslash \{c\}$ and $CASTE^I(a,t+1) = CASTE^I(a,t) \cup \{c\}$

Agent a *deactivates* caste c at moment t, if and only if, agent a binds to caste c actively at moment t, and at moment $(t+1)$ the status of caste c will be changed from active to inactive.

Theorem3. *deactivate* has the following properties

(1) $\models \langle a: deactivate(c) \rangle \rightarrow$ Active(a, c)

(2) $\models \langle a: deactivate(c) \rangle \rightarrow \bullet$ (Inactive(a, c))

Property (1) means if agent a *deactivates* some caste c, then the caste c must be bound actively. Property (2) shows if agent a *deactivates* some caste c, then the state of caste c will be changed to inactive when the action is completed.

Definition13. $M \models_{r,t} \langle a: activate(c) \rangle$ iff $c \in CASTE^I(a, t)$ and $CASTE^I(a,t+1) = CASTE^I(a,t) \backslash \{c\}$ and $CASTE^A(a,t+1) = CASTE^A(a,t) \cup \{c\}$

Agent a *activates* some caste c at moment t, if and only if, agent a binds caste c inactively at moment t, and at moment $(t+1)$ the state of caste c will be changed from inactive to active.

Theorem4. *deactivate* has the following properties.

(1) $\models \langle a: activate(c) \rangle \rightarrow$ Inactive(a, c)

(2) $\models \langle a: activate(c) \rangle \rightarrow \bullet$(Active($a$, c))

Property (1) means that if agent a *activates* caste c, then the caste c must be bound to inactively. Property (2) shows that if agent a *activates* caste c, then the state of caste c will be changed from inactive to active. We assume that there is no other actions in agents except *join*, *quit*, *deactivate* and *activate* that can change the casteships of an agent to any caste. Formally, for any agent a, caste c, action *act* and time moment t, if $M \models_{r,t} \langle a: act \rangle$ and $act \notin \{join, quit, deactivate, activate\}$, then $CASTE^I(a, t) = CASTE^I(a,t+1)$ and $CASTE^A(a, t) = CASTE^A(a,t+1)$).

Definition14. An agent a is *rational* about caste operations, if and only if, it satisfies the following properties.

(1) $M \models_{r,t} \langle a: join(c) \rangle \rightarrow M \models_{r,t} \neg$BindCaste($a$, c)

(2) $M \models_{r,t} \langle a: quit(c) \rangle \rightarrow M \models_{r,t}$ Active(a, c)

(3) $M \models_{r,t} \langle a: deactivate(c) \rangle \rightarrow M \models_{r,t}$ Active(a, c)

(4) $M \models_{r,t} \langle a: activate(c) \rangle \rightarrow M \models_{r,t}$ Inactive(a, c)

Formula (1) means that when an agent intends to *join* a caste, then the caste should not be bound by the agent. Formula (2) states that when an agent intends to *quit* a caste, the agent should have already actively bound to the caste. Formula (3) means that when an agent intends to *deactivate* a caste, the agent should have already actively bound to the caste. Formula (4) states that when an agent intends to *activate* a caste, then the caste should be already bound to inactively by the agent.

Definition15. An agent a is faithful, if and only if, agent a intending to *join* caste c at some moment t implies that the caste c is directly reachable for agent a at moment t.

Note that, an agent can only take actions defined by the castes that the agent actively binds, i.e., for any agent a, action *act* and moment t, $M \models_{r,t} \langle a: act \rangle \Rightarrow \exists c$: $c \in CASTE^A(a,t)$ and $act \in A_c$.

Lemma2. For any faithful agent a, caste c and moment t, if $c \notin Reach(a,t)$ and $c \notin CASTE(a,t)$, then for any $t' > t$: $c \notin CASTE(a, t')$.

The lemma states that if a caste is not reachable for agent a at some moment t, then the agent is impossible to bind the caste in its future run.

Definition16. An agent a is consistent about caste operations, if and only if, it satisfies the following conditions. (1) if agent a intends to join caste c at moment t, then there is no caste $c' \in CASTE^A(a,t)$: $c \uparrow c'$; (2) if agent a intends to activate caste c at moment t, then there is no caste $c' \in CASTE^A(a,t)$: $c \uparrow c'$.

Lemma3. For any faithful agent a and moment t, if *Reach(a,t)* is consistent, then the agent will be consistent in its future run.

Lemma4. For any agent a, caste c and moment t, if agent a execute action "*join(c)*" and there is no caste $c' \in CASTE^A(a,t)$: $c \uparrow c'$, then agent a is consistent when the "*join*" action is completed; if agent a executes action "*activate(c)*" and there is no caste $c' \in CASTE^A(a,t)$: $c \uparrow c'$, then agent a is consistent when the "*activate*" action is completed.

The lemma shows that if an agent intends to *join* or *activate* a caste and the caste to be joined or activated does *not* conflict with any castes that agent *a* has already been actively bound to, then when the operation is completed the agent is consistent.

Definition17. An agent follows the dynamic binding mechanism in its run, if and only if, the agent is rational, faithful and consistent about the caste operations. If all agents in MAS follow the dynamic binding mechanism, then we say that the MAS follows the dynamic binding mechanism, such MAS is abbreviated as MAS_{DBM}.

Definition18. We call that an agent *a* is reachable at moment *t*, if and only if, for any caste $c \in$ CASTE*(a, t)*, $\exists t' < t$: $M|=_{r,t'} <a: join(c)>$ and $c \in DReach(a,t')$.

Definition19. For any caste operation $act \in \{join, quit, activate, deactivate\}$ and an agent *a*, if agent *a* are always consistent and reachable based on the *act* operation, then we say that the caste operation *act* is safe for agent *a*.

Theorem5. If agent *a* follows the dynamic binding mechanism, then the *join*, *quit*, *activate* and *deactivate* caste operations are safe for agent *a*, i.e., for any caste operation $act \in \{join, quit, activate, deactivate\}$, caste *c* and moment *t*, (1) if $M|=_{r,t} <a: act(c)>$ and agent *a* is consistent at moment *t*, then when *act* is completed agent *a* is still consistent ; (2) if $M|=_{r,t} <a: act(c)>$ and agent *a* is reachable at moment *t*, then when *act* is completed agent *a* is still reachable.

5. Conclusion

Dynamic agents that typically exhibit various behaviors in their lifetime are widespread in complex MASs. In the past years, many attempts have been made to support the development of such agents. However, it is still a challenge and open problem in the literature of AOSE. In this paper, we present an approach of dynamic binding mechanism to model and design dynamic agents. We adopt caste as abstraction to specify agents' behaviors and as modular unit to implement dynamic agents. Our approach permits agents to bind multiple castes and the caste that agent binds can be in active or inactive state. The dynamic behaviors of agents are interpreted and realized as the change of agents' casteships in their lifecycles, which can be specified and implemented by four atomic operations on castes. The semantics of dynamic binding mechanism and caste operations are rigorously defined based on the temporal logic integrating with dynamic operators. Some important properties of dynamic binding mechanism are formally specified and discussed. Our approach to dynamic agents differs from the approach proposed in [2] as we permit multiple castes to be bound at a moment and the *join* operation actually integrates with the *enact* and *activate* operations. There is no explicit gap between caste specification and agent design.

Acknowledgements. The authors acknowledge supports from Natural Science Foundation of China (60373022, 90612009), 973 project of China (2005CB321802), 863 project of China (2005AA113130), and science foundation of Huawei Corporation.

References

1. Xinjun Mao, Eric Yu, Organizational and Social Concepts in Agent oriented Software Engineering, Proceedings of AOSE, LNCS 3382, 1-15, 2005.
2. Mehdi Dastani, M. Birna van Riemsdijk, Joris Hulstijn, Frank Dignum, John-Jules Ch. Meyer, Enacting and Deacting Roles in Agent Programming, Proc. of AOSE V, LNCS 3382, 189-204, 2005.
3. Shan, L. and Zhu, H., CAMLE: A Caste-Centric Agent-Oriented Modelling Language and Environment, Proc. of Software Engineering for Multi-Agent Systems III: Research Issues and Practical Applications, LNCS 3390, 144-161, Springer, 2005.
4. Zhu, H., and Lightfoot, D., Caste: A step beyond object orientation, Proc. of JMLC, LNCS 2789, 59-62, 2003.
5. F. Zambonelli, N. R. Jennings, and M. Wooldridge, Developing Multiagent Systems: The Gaia Methodology, ACM Transactions on Software Engineering Methodology, 12(3), 317-370, 2003.
6. N.R.Jennings, An agent-based approach for building complex software systems, Communication of ACM, 44(4), 35-41, 2001.
7. E. Yu, Agent-Oriented Modelling: Software Versus the World, Proc. Of AOSE, LNCS 2222, 206-225, 2001.
8. J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems, Proc. of 3rd International Conference on MultiAgent Systems, 128-135, 1998.
9. Franco Zambonelli, Andrea Omicini, Challenges and Research Directions in Agent-Oriented Software Engineering, Autonomous Agent and Multi-Agent Systems, 9, 253-283, 2004.
10. Michael Luck, Peter McBurney and Chris Preist, Agent Technology: Enabling Next Generation Computing, A Roadmap for Agent-based Computing, http://www.agentlink.org, 2005.
11. F Giunchiglia, John Mylopoulos and A Perini, The Tropos Development Methodology: Processes, Models and Diagrams, Proc. Of AAMAS, 35-36, 2002.
12. Zhu Hong, A formal specification language for agent-oriented software engineering, Proc. of AAMAS, 1174 -1175, 2003.
13. Thomas Juan, Adrian Pearce and Leon Sterling, ROADMAP: Extending the Gaia Methodology for Complex Open System, Proc. of AAMAS, 3-10, 2002.
14. Bauer, B., Muller, J.P., and Odell, J., Agent UML: a formalism for specifying multiagent software systems, Proc. Of AOSE, LNCS 1957, 91-103, 2001.
15. J. Odell, H. V. D. Parunak, S. Brueckner, J. Sauter. Temporal aspects of dynamic role assignment, Proc. Of AOSE, LNCS 2935, 201-213, 2003.
16. Wang, J., Shen, R., Zhu, H. Agent Oriented Programming based on SLABS, Proc. of COMPSAC, 2005.

An Ontology Based Multi-Agent System Conceptual Model

Walid Chainbi
ENIS B.P.W –3038- Sfax – Tunisia-
Walid.Chainbi@lycos.com

Abstract

Ontologies have established themselves as a powerful tool to enable knowledge sharing, and a growing number of applications have benefited from the use of ontologies as a means to achieve semantic interoperability among heterogeneous, distributed systems. This paper is about the use of ontologies in multi-agent systems conceptual modeling. By coming up with ontologies related to communication and organization, we deduce a multi-agent system conceptual model

1. Introduction

Computational intelligence research originally focused on complicated, centralized intelligent systems with expertise in certain domains. However, the increasing demand for distributed problem solving led to the development of multi-agent systems which are formed from a collection of independent software entities whose collective skills can be applied in complex and real-time domains [1]. These software entities exhibit characteristics like autonomy, responsiveness, pro-activeness and social ability [2]. Their functionality and effectiveness has proven to be highly depended on the way they are represented in conceptual models [3]. The way intelligent agents work, how the software seems to act in intelligent ways on its own, whether over computer networks or not, can be difficult to be perceived from people.

Actually, the long tradition in Artificial Intelligence (AI) suggests modelling an agent as having mental states such as beliefs and desires [4, 5, 6]. This approach has been influenced by the folk psychological explanation of human behavior. This is also the perspective taken in the philosophical explanation of rational behavior, and generally referred to as an *intentional stance* [7]. In this approach, interaction is masked which gives the impression that agents are isolated.

This paper proposes a multi-agent system conceptual model based on the development of a cooperation ontology. Cooperation constitutes one of the key concepts which differentiates multi-agent systems from other related disciplines such as distributed computing, object-oriented systems, and expert systems.

The term *ontology* has been originally used in philosophy where it indicated the systematic explanation of existence. More recently, the term has been used in various areas in AI and more widely in computer science, such as knowledge engineering, knowledge representation, information retrieval and extraction, e-commerce, ...(for more details see [8]). An *ontology* is a kind of knowledge representation describing conceptualization of some domain. It specifies a vocabulary including the key terms, their semantic interconnections, and some rules of inference. Formally, an ontology is an agreement about a shared conceptualization, which includes frameworks for modeling domain knowledge and agreements about the representation of particular domain theories. Definitions associate the names of entities in a universe of discourse with human readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these names [9].

The remainder of this paper is organized as follows. Section 2 deals with a cooperation conceptualization. In section 3, we deduce our agent concepts. By using labeled transitions systems in section 4, we describe the behavior of a single-agent system as well as the behavior of a multi-agent system. Section 5 discusses the merits of the presented study. We conclude in section 6 with a summary of the main points relevant to this study, and we give directions for future work.

2. Cooperation conceptualization

2.1. Communication conceptualization

Communication in multi-agent systems as for human beings, is the basis of interaction and social organization. Communication consists of a set of psychological and physical processes relating the sender to the addressee(s) in order to reach some goals [10]. This definition emphasizes three underlying dimensions to communication: *physical* dimension, *psychological* dimension, and *social* dimension. The physical dimension deals with all the required physical means to support communication such as physical connexions. The psychological dimension takes mainly into account mental factors which can be in the beginning or in the end of the communicative action. Finally, the social dimension deals

with the set of conventions adopted as a guiding line by a group (e.g., communication protocols). Next, we outline the underlying concepts to communication.

2.1.1. Beliefs. The concept of belief is an essential element of a communication model. Indeed, beliefs are a subset of the propositions that agents exchange. Beliefs correspond to the notion of knowledge in more traditional AI terms. It can be *true* knowledge if the intended theory is *internal* i.e., the formalisms are described according to the observations of individual agents on their proper internal states, their environment and the other agents. Otherwise, it depends on the judgment of an external observer. Beliefs change due to the external environment of the agent, his communication with the others and his proper reasoning.

2.1.2. Goals. A goal is an underlying concept to the psychological level of a communication. It refers to a potential state of the world to which the agent belongs. Goals and beliefs are the exchanged propositions between agents.

2.1.3. Actions. A communication is defined as a speech act [11] and a-fortiori as an action. Accordingly, an action is the basic element of a communication model. An agent can perform at a point of time one of the following actions: *physical* actions (i.e., interactions between agents and the spatial environment), *communicative* actions (i.e., interactions between agents and can be emission or reception actions), *private* actions (i.e., internal functions of an agent and correspond to an agent exploiting his internal computational resources), *decision* action (i.e., can generate communicative, physical and private actions, and can also update the agent's beliefs). We assume that the agent's goals are modified only after a negotiation with the other agents.

The actions to execute are determined by the resolution methods and communication protocols. Concerning the actions execution, an agent use an interaction paradigm (e.g., an agenda). The interaction paradigm implicitly defines the metaphors used by the agents to interact and cooperate among each other [12].

2.1.4. Message. The transition step between the psychological and physical dimensions is message production. In a multi-agent universe, a message is a specification of a speech act to which we can add the physical processes of communication supporting message transmission. Practically, it can be considered as a four attributes structure: elocution force, propositional content, sender, and routing. Next is an example of this structure according to the well known prey/predators problem in which four predators try to capture a prey on a bi-dimensional grid.

Force : *inform*
Propositional content *: the prey's position is* (x_0,y_0)
Sender : *predator 1*
Routing : *predator 2*

2.2. Organization conceptualization

Whenever we have to deal with task distribution and well-knit interaction between agents in a multi-agent system, the basic problem is an organizational issue i.e, deciding who will do what and when [13]. Organization in animal or human societies deals with task distribution and work division according to the proper skills of each agent member. Next, we draw the key concepts which are the basis of an organization in cooperation framework.

2.2.1. The problem. A problem is defined by a set of tasks. Each task can be optional or compulsory according to a resolution strategy. A task is an ordered set of actions allowing to reach a goal. For example, the prey/predators problem includes the following compulsory tasks: "follow the prey", "communicate the prey's position once it is seen". To this set, we can add the following optional tasks: "communicate the predator's position at each step", or "communicate the distance covered by the predator".

2.2.2. Role. The role characterizes the responsibilities which would be attributed to an agent i.e., the tasks he is charged with. An agent's role is defined by a subset of compulsory tasks and a subset of optional tasks. The study presented in this paper doesn't deal with coordination. However, we think that coordination is related to cooperation as follows: coordination deals mainly with the additional activities which are not directly productive (usually, we use the term control knowledge) but are used to improve the achievement of productive activities. The achievement of productive tasks involve coordination tasks without which the formers could not be implemented.

2.2.3. Cooperation mode. The concept of cooperation mode has been previously stated in [14]. A cooperation mode specifies the responsibilities attributed to the partners taking part in a collective activity. A cooperation mode is defined by a tuple representing the respective roles of the agents. The details concerning the nature of relations between the different roles are out of the scope of this paper. The relevance of a cooperation mode is measured by the cooperation strategy which will be described in the following paragraph.

2.2.4. Cooperation strategy. In a distributed problem solving, each agent member of the system is able to process a set of tasks by applying some resolution methods within his competencies. Agent competencies can

be complementary or redundant. For the same tasks, an agent may be more competent than another (he puts into practice a resolution method more efficaciously). For other tasks, the agents can have different methods but there is no way to determine a priori which method is the most efficient. Accordingly, to select an agent to achieve a task, the resolution context should be taken into account. Hence, a dynamic distribution of tasks seems to be more appropriate than a static distribution. If the abilities of the agents are redundant, then they can play the same role. In this case, an agent should be selected to accomplish each role. According to the context, we can specify different indications concerning the way the choice is done. For example, in the case of system-human cooperation, we can indicate that the user must play all the decision roles even if the system is competent to accomplish such roles. We call a cooperation strategy, a process which determines the appropriate cooperation mode to a given situation. In this case, we talk about dynamic organization. In certain kinds of problems, a static structure may be more appropriate than a dynamic one. Such discussion is beyond the scope of this paper ; it can be addressed apart in another study with more details. Generally, and according to the essence of multi-agent systems, a dynamic structure can respond better to complexity. A cooperation strategy can be implemented by using a refereeing function. For example, in the prey/predators problem, this function can take the current players' positions and their goals as input and return the new players' goals. Accordingly, new roles have to be achieved to reach the new goals.

In order to guarantee the coherence of an organizational model, some properties should be verified. We think that they are closely related to the application domain.

3. Agent concepts

A conceptualization of cooperation has been presented so far. Indeed, two aspects have been mentioned as mainstays of cooperation: communication and organization. Concerning communication, the terminology includes the concepts of beliefs, goals, actions and message. Organization includes the concepts of roles, cooperation mode, and cooperation strategy.

3.1. Agent's state

Multi-agent systems put into practice a set of techniques and concepts allowing some concrete (e.g., robots) or abstract (e.g., software) entities called "agents" to cooperate according to some cooperation modes. By focusing on interaction and individual satisfaction, multi-agent systems ban thinking in a centralized or global way. Accordingly, it is forbidden to talk about cooperation mode (see § 2.2) whenever we talk about agent systems. We keep from our previous study the following concepts for an agent: *beliefs* and *goals* as communication concepts and *roles* as concepts related to organization. These concepts define an agent's state and indicate the aspects that may change over time.

We think that our beliefs correspond to the notion of knowledge in more traditional AI terms. The concept of role is very similar to "Intention". The concept of role as stated here (an organizational concept) denotes the importance of the organization dimension whenever we talk about agent systems (which has not been stated for intentions in the literature). We think that the term "role" is better (more expressive) whenever we talk about organization. Besides, intentions and roles represent both the deliberative state of an agent but they come from different approaches : the former was the fruit of the adoption of the intentional stance, whether the latter is the result of what is adopted in this paper as an ontology based approach. Concerning the relation role-plan, a role is not a plan. Indeed, a role represents dynamic if transient behavior whether a plan is a schema which can be executed to get something done.

3.2. Agent's activity

The types of activity an agent can perform may be classified as either external (in which case the domain of the activity is the environment outside the agent i.e., the other agents and the spatial environment) or internal (in which case the domain of the activity is the agent himself).

An agent interacts with the other agents by executing communicative actions (emission actions and reception actions). Generally an emission action does not modify the set of beliefs of an agent, whereas a reception action may have an impact on his beliefs. An agent can also execute physical actions on his spatial environment. Each action from the spatial environment (a stimulus) on an agent will be interpreted. The results of the interpretation activity, agent's beliefs and goals, resolution methods, and the communication protocols form the parameters of the decision activity. The latter can update an agent's beliefs. It can also modify the agent's role by generating any action (physical, private or communicative) which will be integrated into an interaction paradigm (for example an agenda). The internal activity of an agent includes also his private actions which act on his beliefs.

As we have dealt with conceptualization, next we deal with the formal aspects of an ontology.

4. Semantics

In this section, we use labeled transition systems [15] to capture the conceptualizations described in the previous section. Our choice is mainly justified by the fact that notions that appear with different and complex forms in the literature such as Petri nets [16], process calculus [17] can be described in the same formal framework namely transition systems. Indeed, these notions amount to specify a set of states, and a set of transitions between these states. For example, the states of a Petri net are its markings, and the state transitions are implemented with the simultaneous (or not) firing, depending on the semantics given to the net, of net transitions. In process algebra, the system states are terms, and the transitions are defined by the calculus operational semantics which indicates how and under which conditions, a calculus term can be transformed into another term [15].

The proposed model represents a multi-agent system by a triplet < **S**, **A**, **Ω** > consisting of:

- a set **S** of states where each element describes the complete instantaneous state of the system ;
- a set **A** of actions ;
- a set **Ω** of all possible system executions which are sequences of the form

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \cdots$$

where each s_i is a state and each a_i is an action.

4.1. A single-agent system model

An agent state consists of his beliefs, goals and roles. To represent beliefs, we adopt the idea of Wooldridge [18] who represented beliefs by ground atoms of first-order logic (i.e., atomic formulae containing no variables).

We represent an agent's goal by a first-order formula. We assume that the sets of all possible beliefs and goals are finite since we deal with agents that have limited resources (i.e., non omniscient). For instance, according to the prey/predators problem, getting closer to a prey from the western side is a predator's goal that can be represented by the following formula :

$\forall X\ \forall Y$ *Position (prey, X, Y)* $\Rightarrow$ *Position (self, X-1, Y)*

where *self* is the predator's identifier and *Position (prey, X,Y)* is an atom representing a predator's belief

A role is represented by an ordered set of actions where each action can be physical, private, or communicative. Notice that the decision action doesn't belong to the action sequence representing the role.

Definition 4.1 [An agent state]
An agent state is a tuple <*B, G, R*> defined as follows: *B* is the set of beliefs of the agent, *G* are his goals, and *R* is his role.

We denote by *PHA* the set of physical actions, *PRA* the set of private actions, *COA* the set of communicative actions such that $COA = COEA \cup CORA$ where *COEA* is the set of emission actions and *CORA* is the set of reception actions, τ a decision action, which an agent can execute.

Definition 4.2 [The state transition due to an action]
Let s=<*B, G, R*> be a state of an agent and a an action ($a \in PHA \cup PRA \cup CORA \cup COEA$). The state transition due to a is described by the following axiom :

$$< B, G, R > \xrightarrow{a} < B', G, R' >$$

Such that:

1. (i) If $a \in PHA \cup PRA \cup CORA$ then $B' = (B - Bels_a(s)) \cup Bels'_a(s)$ where:
   $Bels_a(s)$ denotes the set of beliefs of the agent at a state s and on which the action a has an impact. This impact corresponds to the application of functions that is, if $Bels_a(s) = \{R_1(...), ..., R_i(...),..., R_k(...)\}$ where R_i is a relation symbol, then there are functions symbols $f_1, ..., f_i, ..., f_k$ such that $Bels'_a(s) = \{R_1(f_1(...)), ..., R_k(f_k(...))\}$[1].
   (ii) If $a \in COEA$ then $B' = B$.
2. R' is the result of suppressing a from R i.e., $R=a.R'$ where "." is the operation of concatenation.

For example, in the prey/predators game, *move_up* is a physical action which a predator is able to do. The ground atom *Position(self,* v_1, v_2*)* is an agent's belief which indicates his position on the grid. Let $Bels_{move\_up}(s) = \{$ *Position (self,* v_1, v_2*)*$\}$ and G a predator's goal (e.g. getting closer to a prey from the western side). Let the predator's role = <*move_up, move_right, move_right*> = R. Then, we have

$$< B, G, R > \xrightarrow{move\_up} < B', G', R' >$$

Where: $B' = (B - \{Position\ (self,\ v_1,\ v_2)\}) \cup \{Position\ (Inc\_Ord(self,\ v_1,\ v_2))\} = (B - \{Position\ (self,\ v_1,\ v_2)\}) \cup \{Position\ (self,\ v_1,\ v_2+1)\}$ such that *Inc_Ord* is a function which increments the second component of agent coordinates on the grid (i.e., the second term of the relation symbol *Position*) ; $G'=G$; R'=<*move_right, move_right*>.

Definition 4.3 [The state transition due to a decision action]
Let <*B, G, R*> be a state of an agent and τ a decision action. The state transition due to the execution of a decision action is described by the following axiom:

[1] The application of a function symbol on terms of first-order logic results in possibly different terms - see the example with the use of the function *Inc_Ord.*

$$< B, G, R> \xrightarrow{\tau} <B', G, R'>$$

1. $B' = (B - Bels_\tau(s)) \cup Bels'_\tau(s)$
2. $R' = mod(R)$ where *mod* is an update function of the role resulting in the following operations :
   - The generation of an action (physical, private or communicative). In this case the action will be integrated in the role and *mod (R) = Insert (R, a)*. -- *Insert* is a function which integrates the action *a* in the role *R*
   - The suppression of an action (physical, private or communicative). In this case *mod(R) = Delete*(*R*, *a*). -- *Delete* is a function which deprives the role of one of its actions.
   - The repetition of the two operations mentioned above.

4.2. A multi-agent system model

In this section, we extend the behavioral semantics of an agent presented in the previous section to deal with the behavioral semantics of a multi-agent system.

Definition 4.4 [A multi-agent system state]
A multi-agent system state *SA* is a tuple $<<B_1, G_1, R_1>, ..., <B_i, G_i, R_i>, ..., <B_n, G_n, R_n>, GG>$ where : $<B_i, G_i, R_i>$ is the state of the agent *i*, and *GG* is the global goal of the system such that $\mathbf{C}(G_1, G_2, ..., G_n) \Rightarrow GG$ where **C** is a relation which combines the local goals of the agents.

C is closely related to the system to be modelled. For instance, in the prey/predators problem, *GG* is the capture of the prey, G_i is getting closer to the prey from one side (north, south, east or west) and **C** is the conjunction of all the local goals.

Definition 4.5 [the state transition due to an action]
Let $<<B_1, G_1, R_1>, ..., <B_i, G_i, R_i>, ..., <B_n, G_n, R_n>, GG>$ be a state of our system, $A_i = <B_i, G_i, R_i>$ a state of an agent *i* (*i* : 1..n), and *a* an action. The state transition due to *a* is described by the following inference rule:

1. $a \in PHA \cup PRA$

$$\frac{<B_i, G_i, a.R_i> \xrightarrow{a} <B'_i, G_i, R_i>}{<A_1, ..., A_i, ..., A_n, GG> \xrightarrow{a} <A_1, ..., <B'_i, G_i, R_i>, ..., A_n, GG>}$$

If an agent is ready to execute a physical or private action, then the whole system is ready to execute the same action.

2. $a \in COEA, \ \overline{a} \in CORA$

$$\frac{<B_i, G_i, a.R_i> \xrightarrow{a} <B_i, G_i, R_i>; \quad <B_j, G_j, \overline{a}.R_j> \xrightarrow{\overline{a}} <B'_j, G_j, R_j>}{<A_1, ..., A_i, ..., A_n, GG> \xrightarrow{a\,\overline{a}} <A_1, ..., <B_i, G_i, R_i>, ..., <B'_j, G_j, R_j>, ..., A_n, GG>}$$

If an agent is ready to send a message *a* and another agent is ready to receive the same message, then the whole system is ready in this case to execute this interaction. We note it by $a\,\overline{a}$. One can see that in this rule, we model communication in a synchronous way[2].

3. τ a decision action

$$\frac{<B_i, G_i, R_i> \xrightarrow{\tau} <B'_i, G_i, R'_i>}{<A_1, ..., A_i, ..., A_n, GG> \xrightarrow{\tau} <A_1, ..., <B'_i, G_i, R'_i>, ..., A_n, GG>}$$

Each decision action executed by an agent changes the state of the system by changing the corresponding state of the decision-maker.

4. negotiation action: it is a finite sequence of communicative actions initiated to update the goals of the agents (actually, the communicative actions used to negotiate are different from the usual communicative actions mentioned above. The latters cannot modify the individual goals whereas the formers are designed to update these goals). Indeed, under certain circumstances, it can be useful for an agent to modify his proper goal. We assume that the global goal can change if the agents perceive that it can't be achieved any longer. Let η be a negotiation action, we use the following axiom to describe its impact:

$$<A_1, ..., A_i, ..., A_n, GG> \xrightarrow{\eta} <A'_1, ..., A'_i, ..., A'_n, GG'>$$

Where $A_i = <B_i, G_i, R_i>$; $A_i' = <B'_i, G'_i, R'_i>$ and $\mathbf{C}(G'_1, G'_2, ..., G'_n) \Rightarrow GG'$.
The negotiation action is represented in the above definition at a high level of abstraction. Therefore, we don't discuss how a negotiation takes place in this paper.

5. Discussion

The presented study is about the use of ontologies in multi-agent systems conceptual modeling. Ontologies have established themselves as a powerful tool to enable

[2] Actually, agents can also communicate in an asynchronous way i.e., an agent has the possibility to do something else (e.g., a physical action) while he is waiting for a communication.

knowledge sharing, and a growing number of applications have benefited from the use of ontologies as a means to achieve semantic interoperability among heterogeneous, distributed systems [8]. Ontologies are used in this paper as an argumentation tool to what is presented as belief-goal-role agents. Unlike most previous work on multi-agent systems, which have focused on the principled construction of individual agents following an agent-centered view (see [4, 5, 6] for example), this study has followed a global view by stressing the study of interaction (which is probably the most important single characteristic of complex systems) to deduce a multi-agent system conceptual model. The main problem with the previous studies (those based on the intentional stance) is that the interaction aspect is masked (or very difficult to perceive). An ontology based approach is particularly useful for understanding the rationale behind the existing practices and structures. In this paper, an ontology is used to design a multi-agent system conceptual model. Design is a creative process, and there is an inherent tendency in any creative process to be neither precise nor accurate, but rather to follow the inspiration of the moment in an unstructured manner. The fundamental problem with this process (initially mentioned by Jones in [19]) is that designers are obliged to use current information to predict a future state that will not come about unless their predictions are correct. The final outcome of designing has to be assumed before the means of achieving it can be explored: designers have to work backwards in time from an assumed effect upon the world to the beginning of a chain of events that will bring the effect about. Ontologies can be used to tame this difficulty, and to guide and discipline creativity.

6. Conclusion

In this study, we have presented a multi-agent system conceptual model based on the development of an ontology for cooperation. One main hypothesis is developed in this work, considering that communication and organization constitute, at least in a pragmatic point of view, two complementary ways to implement cooperation. Agent metaphor comes packaged with a large number of powerful psychological abstractions such as beliefs, desires and intentions. The presented study is an argumentation endeavor to what is presented as belief-goal-role agents. Starting from the fact that not all conceptualizations of a domain are equally suitable, we think that the presented ontology-based approach is crucial for achieving an adequate conceptualization.

In future work, the following issues can be investigated. We need to deal with conflicts that arise from the use of ontological arguments in conceptual modeling. Actually, ontologies can result from the efforts of many domain experts. Thus, they can be designed and maintained independently. In such a situation, we may obtain two heterogeneous views, and then we need to know which view is more sound. Obviously, the soundness should be defined clearly according to fixed criteria. Then, two behaviors can be envisaged : choosing the ontology which is more sound than the other, or reconciling those different views by merging or integrating the diverse ontologies.

7. References

[1] M. d'inverno, M. Luck, *Understanding Agent Systems* (Second Edition), Springer, 2004.

[2] N.R. Jennings, M. Wooldridge, "Intelligent Agents : Theory and Practice", *The Knowledge engineering review*, 10(2), 115-152, 1998.

[3] S. Wang, D. Yang, K. Tanaka, F. Grandi, Z. Shuigeng, E. Mangina, T. Ling, I. Song, G. Jihong, H. Mayr, *Proc. First International Workshop on Conceptual Modelling for Agents* (CoMoA2004), Shanghai, China, November 2004, LNCS 3289, 457-496, 2004.

[4] P.R. Cohen, H.J. Levesque, "Intention is choice with commitment", *Artificial Intelligence*, 1990, pp. 213-261.

[5] M. Georgeff, A. Rao, "An Abstract Architecture for rational Agents", proc. of KR & R-91, 1991, pp. 473-484.

[6] N.R. Jennings, M. Wooldridge, "Agent Theories, Architectures and Languages: A Survey", *LNAI 890*, Springer Verlag, 1994.

[7] D. Dennett, *The Intentional Stance*, MIT Press, 1987.

[8] V. A.M. Tamma, *An Ontology Model supporting Multiple Ontologies for knowledge sharing*, PHD Thesis, University of Liverpool, 2001.

[9] M.P. Singh, M.N. Huhns, *Service-Oriented Computing*. John Wiley & Sons, Ltd, 2005.

[10] D. Anzieu, J.Y. Martin, La Dynamique Des Groupes Restreints, Puf, 1968.

[11] J.L. Austin, *How to do things with words*, Oxford University Press, 1962.

[12] S. Bandinelli, E. Di Nitto, A. Fuggetta, "Supporting Cooperation in the SPADE-1 Environment", *IEEE Transactions on Software Engineering*, Vol. 22, No 12, 1996.

[13] L. Gasser, "Distribution and Coordination of Tasks Among Intelligent Agents", *Scandinavian Conference on Artificial Intelligence*, 1988, pp. 189-192.

[14] J.L. Soubie, A.H. Kacem, "Modèles de coopération homme/système intelligent", *systèmes coopératifs de la modélisation à la conception,*Octares, 1994.

[15] A. Arnold, "Systèmes de transitions finis et sémantique des processus communicants", *Techniques et Sciences Informatiques journal*, Vol.9, no 3, 193-216, 1990.

[16] T. Murata. "Petri Nets: Properties, Analysis and Applications". Proc. IEEE, vol.77, no 4, 1989.

[17] R. Milner, *A Calculus of Communicating Systems*. LNCS 92, Springer-Verlag, 1990.

[18] M. Wooldridge, "This is MYWORLD: Th Logic of An Agent-Oriented DAI Testbed", *LNAI 890*, Springer Verlag, 1994.

[19] J.C. Jones, *Design Methods : seeds of human futures*, 1970.

A Hierarchical Agent-oriented Knowledge Model for Multi-Agent Systems

Liang Xiao and Des Greer
School of Computer Science,
Queen's University Belfast
Belfast, BT7 1NN, UK
email: { l.xiao, des.greer }@qub.ac.uk

Abstract

Software systems used in a business environment must consider knowledge about business processes, business rules, business terms, and their manageability, adaptability, and maintainability due to changing environment over time. In this paper, from a Software Engineering perspective, we propose hierarchical structure for business knowledge that is then made available to agents running in systems. Such knowledge is sourced from the business requirements and becomes a knowledge base of agents, the basis for agent behaviour. Because the knowledge is externalised from the system, system behaviour is easily maintained. As a result, the hierarchical agent-oriented knowledge model is advanced and advocated. The work extends the framework of Agent-Rule-Class (ARC) [2] and contributes a further development of the Adaptive Agent Model (AAM) [3] and is illustrated using the case of a rail track management system.

1. Introduction and Background

Software must change if it is to remain useful [1]. When knowledge of business requirements is structured in a manageable and maintainable manner, the effort in re-delivery by developers is reduced. Better still, when software agents are used to manage and apply the configurable knowledge, software adaptivity is facilitated [2] [3]. The authors of the paper view agents from a Software Engineering perspective, where agents are computational entities that have dynamic behaviour, being situated in a changing environment.

In practice, common agent-oriented software development approaches, as reviewed in [4], extend two basic approaches: object-oriented (OO) approaches, where agents are considered as active objects, and knowledge engineering (KE) approaches, where agent knowledge is modelled. Most current approaches only focus on one of them. An example approach of the former type includes agent-oriented programming (AOP) [5], an analogy of object-oriented programming. Another typical example is an agent-oriented methodology for enterprise modelling [6], which combines OO methodologies and enterprise modelling methodologies. In the case of the latter approach, the knowledge acquisition process is the focus as in the case of agent knowledge modelling, CommonKADS [7] being a European standard. Other related approaches include Agent Oriented Abstraction [8], agents being used in corporate knowledge modelling, and DIAMS [9], where agents are used to address knowledge management and information access issues within distributed business environments.

Inheritance and extension in either direction is useful and promote methodology reuse, but insufficient in our opinion. In answering an open question of common interest, "why are agent-oriented methodologies necessary", we argue, using the proposed Adaptive Agent Model (AAM), such a methodology benefits both communities. From one perspective, it provides a higher level of abstraction than an OO methodology on its own. Thus, knowledge can be externalised for easier management rather than fixed in objects. From another perspective, the deployment of modelled knowledge is supported by an underpinning object layer.

In AAM we aim at an operational development paradigm, by which developers of traditional OO systems can easily transfer and adapt their skills, while the resultant agent-oriented systems are easier to manage and deploy domain knowledge. For this purpose, knowledge is encapsulated in business models, being captured requirements structured to be executable by agents. The running agent systems make use of a lower layer of object components, the use of which is specified in the knowledge model in an adaptable form. By using the knowledge in a structured hierarchy, agents easily know what, when, and how components are to be used in various levels. The ultimate goal is to produce reusable and executable models external to agents, AAM being a concrete instantiation of Model Driven Architecture [10].

2. Hierarchy Overview

This section briefly investigates typical business systems and how their requirements, if structured as knowledge for agent systems, give rise to distinct layers emerge. Figure 1 illustrates the three layers of business knowledge.

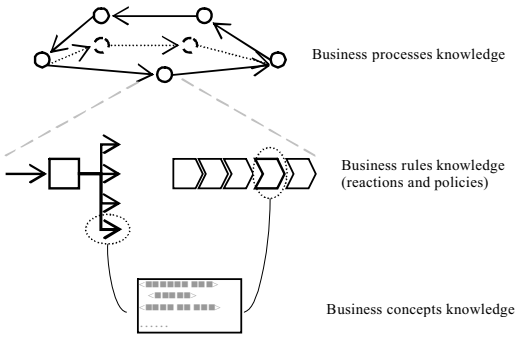


Figure 1: AAM knowledge hierarchy

As shown, in the *business processes knowledge* layer, agents act and react in collaboration one another, relying on their belief or interaction protocols. This first layer is built upon the architectural collaboration patterns of agents. In this layer, inter-agent message exchanges and the results of these satisfy the goals found in the business requirements. A business process involving human or computer agents describes the knowledge in this layer. The next layer of the knowledge model is the *business rules knowledge*. This is found in individual agents and captures how each agent individually performs business tasks in a business process. This involves internal decision making process and execution of proper business policies in that context. By implication, this knowledge must be modelled and known to the agents. Figure 1 shows reaction strategies and policies at this level. These are both categorised as business rules, according to which agents must comply to behave. The third layer of the knowledge model is *business concepts knowledge*. Here, the vocabularies agents use to communicate with each other in the systems, which also might be used in the policy specifications, are defined as building blocks at the bottom of the hierarchy. These building blocks are called business concepts in AAM. For explanatory purposes, AAM components and their features are further described in Table 1.

Table 1: AAM components

Component	Description
Agent	• carries out activities required by AAM knowledge database
AAM knowledge database	• machine understandable, and thus can be translated by agents • built from business requirements
Business process	• principle component of knowledge database • involves other components, can be decomposed into sub-processes as specified in rules
Business rule	• statements, actions, and procedures that should be enforced in business environment • business requirements in a form suitable to be assigned to individual agents • refers to business concepts
Business concept	• essential and irreplaceable atomic business units truly exist in business environment • referred by business rules and spoken by agents

3. Case Study

An actual national railway system specification has been investigated as our case study. The system is mainly responsible for the running of a railway on a daily basis, monitoring train running with regard to incidents and ensuring the safety of the train services by conveying issues to relevant parties for resolution. Being a very complex system, the specification document has more than 250 pages and contains a large number of functions with detailed descriptions. A selected excerpt of the specification follows in Figure 2, with a concern about fault management of the railway system.

Case background: The specification has a main area of **Train Running**, and another **Infrastructure Management**, both are sub-divided into *Business*, *Incident*, and *Execution* domains. Domains relevant with fault management include: Infrastructure Management - Incident (abbreviated **IMI**), being responsible for passing of information about faults between the system and contractors; Infrastructure Management – Execution (abbreviated **IME**), being responsible for granting of isolations; Train Running - Incident (abbreviated **TRI**), being responsible for refinement and corrections of planned train journeys. External entities: **Train Operators**, who initiate train running requests, and have to be consulted when dealing with perturbations, and **Contractors**, who carry out maintenance.

Case terminology: The infrastructure of the railway system consists of the assets necessary to run the trains. An *infrastructure asset* is any identifiable item of interest within the infrastructure. An infrastructure asset may have a number of asset *faults*. Asset faults may either cause an *incident* or may be caused by an incident. An incident may cause a *track restriction*. Under a *contract* or a variation to a contract with a contractor, infrastructure assets are maintained and asset faults are fixed.

Case description: An asset fault is either reported to the system (Requirement: *IMI-AcceptFaultReport*) or detected directly by the system (Requirement: *IMI-NoticeFault*). The handling of both cases is the same (Requirement: *IMI-HandleFault*). If the fault has already been cleared no further action is needed immediately. Otherwise the system notifies the Contractor responsible for the fault and agrees a priority for fixing the fault. The fault may not require immediate attention and may have no immediate impact, in which case nothing further is done. If the fault does have some impact an incident is recorded. It may be necessary to put in place immediate track restrictions (Requirement: *IME-ImposeSuddenRestrictions*), and this will involve changes to forecast train journeys (Requirement: *TRI-RespondToIncident*). Affected train journeys are amended for re-scheduled services to the Train Operator.

Figure 2: Excerpt from rail track case study

4. Layered Knowledge Modelling

The three layers of the knowledge model described in Section 2 are given more details in this section.

4.1 Conceptual Model & Fact Model

The use of metadata and ontologies has become a predominant element in Semantic Web research. These concepts prove useful for management of meta-information separately from the applications that gave rise to much of that information in the first place [11].

For the purpose of easy agent knowledge maintenance, and to ease interoperability, we use a *Conceptual Model* (CM), which externalises business concepts from the applications that use them. This provides a way for agents to understand available terms and their relationships within documents or messages. Concepts are also used to construct meaningful business rules, applicable in various situations.

It is a common practice in the object-oriented community to begin the modelling process with a conceptual model of the domain space relevant for the application being designed. One may use a grammatical analysis of natural language description of a system to discover objects to build the system. This technique is also plausible for the identification of business concepts and the building of a taxonomy. However, here they are not turned into system components, rather, they become knowledge of agents and vocabularies of conversations.

Example business concepts found in the case study description are "fault", "incident", and "restriction". A "fault" has properties indicating its location, impact, and priority, these themselves being business concepts. All business concepts and their relationships in an object-oriented structure form the ontology of our business models. These must be registered in the CM before being referred to by business rules and business processes. Relationships between business concepts may be enforced as required by business rules for business needs. An implication of a relationship between "fault" and "incident" is such a case. Business concepts are represented in XML. The example of "fault" and its related properties is shown in Figure 3.

```
- <concept>
    <name>fault</name>
  - <properties>
      <property>type</property>
      <property>location</property>
      <property>impact</property>
      <property>priority</property>
      <!-- …
    <
  <
```

Figure 3: XML representation of a business concept

At runtime, concrete facts are established in a *Fact Model* (FA) as instantiations of abstract concepts. For example, when a report about an asset fault arrives, one fact may be established and states that a fault has occurred in London, and is a class of "rail broken", and so on. Properties of this business object are thereafter populated with values. One dedicated agent, the *Fact Manager Agent* (FMA) is responsible for the management of all facts at that moment. It interacts with a *Policy Rule Manager Agent* (PRMA) to deduce new facts from existing facts by application of *Policy Rules* (PR), and make available to all agents the known facts. Once facts are established as a result of information brought by event messages, agents have knowledge to reason about their reactions using *Reaction Rules* (RR). Agent knowledge gets dynamically updated as message exchange continues and facts are added or removed. The PRMA inherits the original BRMA of its mechanism [2] [3]. Section 4.5 describes how these special management agents and ordinary agents work together.

This methodology is supplemented by a lower layer class facility which enables the use of an existing OO infrastructure. The facility is based on an agent-class hierarchy [2], where dynamic agents invoke static class methods as determined by business needs. At runtime established facts are mapped to business objects instantiated from business classes, the schemas of which are as defined in the CM. Methods defined on the business objects are invoked for the manipulation of facts as business rules permit or require. This leads to the availability of additional knowledge, supporting agents in their reasoning and behaviour. Because the business concepts that comprise the business rules are separated from the classes which are associated with them, it is only at the time when they are used that the specific matched class methods are bound. Therefore, classes to be invoked at runtime are exchangeable and new behaviour can be achieved by the replacement of classes or their methods.

4.2 Policy Rule Model

Business policies change over time and so externalisation of them as executable rules is justified. Business policies structured in an IF-THEN format is presented in [3] as global rules that all agents in the system should obey. A Policy Rule captures a constraint or invariant. PR assertions on the logical relationships between entities must always be true to reflect the required policies. The compositional entities of PR are business objects, attributes, associations, operations, and the compositional operators of a PR are: IF, THEN, AND, OR, and so on. The IF condition of a PR returns a boolean expression over relationships between entities or values. The THEN action of a PR is an assignment of entity values, concrete values, or acts to other entities or actors. Typical PRs arising from the scenario of fault management are categorised as follows and illustrated with examples.

i). Policies defined for classification of business objects, based on facts.

Rule1.
If fault is located at the capital cities
Then it has "*immediate impact*"

This knowledge is stored as shown in Figure 4.

```
- <policy>
    <id>100</id>
    <condition>
      f                              O
"       "
    </condition>
    <action>
      fault.impact = true
    </action>
    <priority>5</priority>
```

```
</policy>
```

Figure 4: XML representation of a PR

ii) Policies defined on the relationships of (classified) business objects and their attributes, facts deduced.

Rule2.
If fault has "*immediate impact*"
Then it has a high priority of 10

iii) Policies defined on the relationships of (classified) business objects/attributes and behaviour of system, behaviour triggered. These contribute to the formation of RRs. Rule3&4 are given here and are part of the requirement *IMI-HandleFault.*

Rule3.
If fault has no "*immediate impact*"
Then IMI-HandleFault does nothing
Rule4.
If fault has "*immediate impact*"
Then IMI-HandleFault establishes a new incident associated with the fault AND request IME to place track restrictions

The PR form of IF-THEN essentially states if the IF antecedent clause is true then the THEN consequent clause becomes true as well. New facts are increasingly known and this may satisfy other PRs, which leads to additional facts. Recursively, new knowledge is derived from existing knowledge as PR chains are formed. The process proceeds until no more PR has its antecedent satisfied. A PRMA is responsible for the application of PR wherever appropriate as described previously.

4.3 Reaction Rule Model

In AAM, we define an event-driven agent architecture, agents being responsive to events according to rules. The PRMA is responsible for the application of all Policy Rules when triggered by request events, and thus deduced facts are known and used by ordinary agents. Reaction Rules in contrast represent reactive processes that agents individually follow in response to events. Event-oriented decomposition, based on events that the system must handle, is a design strategy that enables modular architecture, easy design, independent unit development, and eventually effective maintenance [12]. In many agent-oriented systems, agents monitor the occurrence of interesting events and respond to them [13]. The agent response might generate new events to other agents. Many systems view event information as a string without any semantic meaning. In contrast, we consider events as providing the context for agents to react and through sequences of successive events, business processes are executed. Agreements are bound between agents for their interactions using. RRs.

The first step in the agent-oriented development process is agent identification and requirements assignment to agents. Note in case study, IMI, IME, and TRI are labels for business domains with some required functions organised per domain. Suppose one business domain is delegated to one agent, who has the knowledge concerned with that domain. When the domain is required for different purposes, the corresponding agent responds and plays several roles to realise several aspects of domain functions, in doing so it fulfilling its responsibilities. Interactions among domains are delegated to message passing among respective agents. Such cross-domain interactions require collaboration of agents, and the collaboration pattern of agents is decided by the interactivity of functions of the involved domains. Some agent-oriented methodologies let designers choose to aggregate related roles into a single agent for convenience [14]. In contrast, we believe this should be done at the specification level, where domain division is the most appropriate criteria that decides the nature of agents and their responsibilities.

In the case study, a set of functions are required in the specification related to how the system manages faults. One function of special concern is reconstructed in Figure 5. *IMI-HandleFault*, belongs to the IMI business domain, and constrains IMI in its handling of faults in reactions. This function describes a constrained system behaviour. The used keyword of "*is informed by*" in Figure 5, followed by the name of another function indicates the source of an event, and "*inform*" or "*use*" followed by the name of another function indicates the target of an action. In this sense, a RR specifies the cause and result of an agent behaviour.

IMI-HandleFault *is informed by* **IMI-AcceptFaultReport** or **IMI-NoticeFault** about an asset fault,
IF the fault has been cleared *THEN* DO_NOTHING, *ELSE*
Inform the responsible **Contractor** about the fault with an agreed priority,
IF the fault has no immediate impact *THEN* DO_NOTHING, *ELSE*
Create an incident related with the fault AND
Create and put in place track restrictions *using* **IME-ImposeSuddenRestrictions**

Figure 5: Reconstructed function specification becomes a Reaction Rule

In knowledge modelling of intra-agent reaction processes, we define a RR structure of: {event, processing, {condition, action}$_n$}. Agents follow RRs for event processing, decision making, and action selection in various conditions. Informative event messages have business objects encoded in them, conforming to pre-defined structures with concrete instantiations at runtime. Specific business objects are decoded and used by the recipient to make corresponding decisions and respond accordingly at the time of running. While different actions are chosen by agents after the decision making, different collaborative agents are chosen, leading to the formation of dynamic business processes, if required.

Essentially, a RR determines on receipt of an event message, after the processing of the message, if certain conditions are evaluated to be satisfied, the actions that agent should perform. Figure 6 shows the XML representation of RR "*IMI-HandleFault*". Details about steps and the mechanism for executing a RR using such an XML schema have been described in a previous paper

[2]. Supporting tools for viewing, addition, and edition of RRs have been developed in previous work [3].

```
- <reaction>
    <name>HandleFault</name>
    <business-process>Fault Management</business-process>
    <owner-agent>IMI</owner-agent>
  - <global-variable>
    - <var>
        <name>asset</name>
        <type>Asset</type>
      </var>
    - <var>
        <name>fault</name>
        <type>Fault</type>
      </var>
    </global-variable>
  - <event>
    - <message>
        <from>IMI.AcceptFaultReport</from>
      - <content>
        - <report>
            - <reporter>Henry</reporter>
            - <fault>
                <type>rail_broken</type>
                <location>London</location>
              - <asset>
                  <id>10015</id>
                  <type>rail</type>
                  <contractor>Contractor_A</contractor>
                  ...
                </asset>
              </fault>
            </report>
          </content>
      </message>
    </event>
    <processing>
      asset = new Asset (reportMsg)
      fault = new Fault (reportMsg)
    </processing>
    <condition>
      fault.cleared () == false
    </condition>
  - <action>
    - <message>
        <to>Contractor.FixFault</to>
      - <content>
        - <fault>
            ...
          </fault>
        </content>
      </message>
    </action>
    <condition>
      fault.immeImpact () == true
    </condition>
  - <action>
    - <message>
        <to>IME.ImposeSuddenRestrictions</to>
      - <content>
        - <asset>
            ...
          </asset>
        </content>
      </message>
    </action>
    <priority>5</priority>
  </reaction>
```

Figure 6: XML representation of a RR

4.4 Business Process Rule Model

Given the RR structure in the previous section, each RR has an internal processing component, and an external interface of event message receiving and action message sending, through which agents interact. The execution of collections of RRs following message flow sequentially and conditionally forms business processes, and thus is the blueprint of systems. The inter-connected RRs collectively constrain business processes and form higher level rules for system goals, called Business Process Rules (BPRs). Thus, one RR is about how one task is to be performed following a process, a goal internal to one agent, while one BPR is about how one business is achieved by a compositional process gathered by a whole collection of RRs, a goal shared by many agents. *IMI-HandleFault* is one of the many RRs that comprise a BPR for managing new asset faults, called "Manage New Fault", shown in Figure 7. Only default conditions are considered as true for simplification.

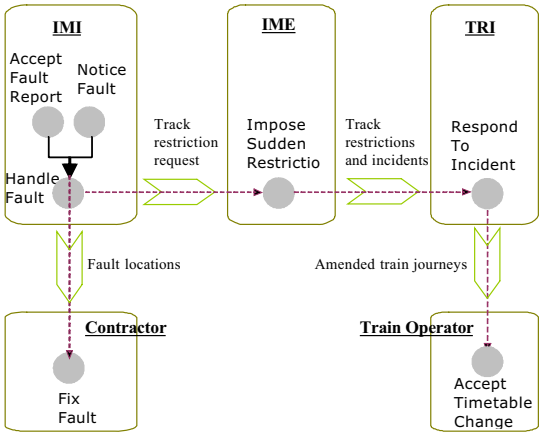


Figure 7: BPR "Manage New Fault" for case study

Agent IMI initialises the above BPR using either of its two RRs: "*IMI-AcceptFaultReport*" or "*IMI-NoticeFault*", in the interest of solving newly detected faults. The agents finalise the BPR are: *Contractor* and *Train Operator*, the completion of whose functions fulfils the goal of managing new faults. We define Initial Agents (IAs) as those initiate the BPR and Final Agents (FAs) as those finalise BPR. IAs act spontaneously without request by other agents and FAs complete the BPR and request no further action from other agents. Intermediate agents participate in BPRs between the activities of IA and FA during the execution of BPRs. BPRs must be ensured of completeness and consistency to satisfy business requirements of business processes and goals. This means for every input to IAs, results can be expected from FAs, indicating goals of respected business processes are accomplished. As long as the input, output and goals of BPRs are all met, the selection of intermediate agents in participation of BPRs, or their individual decision making and other acts are not concerned. Figure 8 shows the BPR "Manage New Fault" in XML, expecting a new fault as input, and "fault fixed" and "train service re-scheduled" as results.

```
- <process>
    <name>Manage New Fault</name>
    <goal>a new fault is managed</goal>
  - <IAs>
      <IA>IMI</IA>
    </IAs>
```

```
- <FAs>
    <FA>Contractor</FA>
    <FA>Train Operator</FA>
  </FAs>
  <cause>a new fault is reported</cause>
- <effects>
    <effect>
      A Contractor will fix the fault
    </effect>
    <effect>
      Train Operator will re-schedule train services
    </effect>
  </effects>
</process>
```

Figure 8: XML representation of a BPR

4.5 Complementary Nature of Rule Types

RRs are not fully functional without supplying additional knowledge from which to reason and make decisions. Equally PRs cannot work independently without a context. In fact they do complement each other in the overall view of our integrated knowledge models, underpinned by the Conceptual Model, and supporting the realisation of BPR and corresponding goals. Collaborative agents perform recursively in three levels aiming at business goals: BPR-RR-PR. When each agent realises its responsibilities in a BPR, it applies relevant RRs and PRs. The fragment of the BPR in Figure 7 carried out by IMI is conducted in practice as follows.

1. A fault is reported to IMI.
2. The "fault" structure encoded in the incoming message matches with the one defined in CM.
3. A fact about a "fault" is established in FM with its location of "London" as well as other information.
4. A business object "fault" is constructed using the schema defined in CM, as well as an "asset".
5. The RR "*IMI-HandleFault*" is selected by IMI in this context as its <event> section is specified to handle reported faults.
6. Facts in FM are looked for in relation with the conditions of the RR, to assist evaluation.
7. FMA interacts with PRMA and a Class Manager Agent to seek additional knowledge either by applying relevant PR or invoking related class methods. The fault is known as having impact as a result of its location, indicated by a PR (R1 in Section 4.2).
8. The business objects of "fault" and "asset" established previously are retrieved and encoded in messages. The messages are prepared to be sent to responsible agents to fix faults and impose restrictions as defined in <action> of the RR.
9. IMI sends the message and FMA demolishes invalid facts.

5. Conclusions

A hierarchical agent-oriented knowledge model is introduced, collectively termed the Adaptive Agent Model (AAM). The AAM is built upon two building blocks: a Conceptual Model (CM) used for vocabulary definition and referred to by agents, and a Fact Model (FM), conforming to the CM constructed at runtime according to agents' current knowledge. Layered rule knowledge supports agent behaviour.

1. BPR: a business process is initialised by the IAs at the beginning, causing a series of agents to react using various RRs, the process being finished at the FAs.
2. RR: an agent chooses a RR to react to after an event in a particular context, makes a decision, selects collaborators, and requests them to carry on the BPR.
3. PR: while a RR is functioning, a set of PRs may become relevant, so forming PR chains which are applied to assist the RR to make the decision or reflect business policies must be enforced in that context.
4. BC: BCs make up structured terms that are used to construct rules, mapping to objects that are invoked by agents.

AAM contributes a complete framework for building agent-oriented business models not only of pragmatic value for applications that require manageable and maintainable specifications, but also can be tailored and adapted for specific use (ARC framework, business models, rule hierarchy, etc.) due to its layered structure.

6. References

[1] M. Lehman & L. Belady, "Program Evolution: Processes of Software Change", Acad. Press, London, 1985.

[2] L. Xiao & D. Greer, "Modelling Agent Knowledge with Business Rules", Proceedings of the Seventeenth International Conference on Software Engineering and Knowledge Engineering (SEKE'05), Taipei, Taiwan, Republic of China, 14-16 July, 2005.

[3] L. Xiao & D. Greer, "The Adaptive Agent Model: Software Adaptivity through Dynamic Agents and XML-based Business Rules", Proceedings of the Seventeenth International Conference on Software Engineering and Knowledge Engineering (SEKE'05), Taipei, Taiwan, Republic of China, 14-16 July, 2005.

[4] C. A. Iglesias, M. Garijo, and J. C. Gonzalez, "A survey of agent-oriented methodologies", In J. P. M¨uller, M. P. Singh, and A. S. Rao, editors, Intelligent Agents V — Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98), LNAI, Springer-Verlag, Heidelberg, 1999.

[5] Y. Shoham, "Agent-oriented programming", Artificial Intelligence, 60:51–92, 1993.

[6] E. Kendall, M. Malkoun, and C. Jiang, "A methodology for developing agent based systems for enterprise integration", In D. Luckose and Zhang C., editors, Proceedings of the First Australian Workshop on DAI, LNAI, Springer-Verlag: Germany, 1996.

[7] G. Schreiber, B. Wielinga, R. Hoog, H. Akkermans, and W. Velde, "CommonKADS: A Comprehensive Methodology for KBS Development", IEEE Expert: Intelligent Systems and Their Applications, 9(6) 28-37, 1994.

[8] P. Maret & J. Calmet, "Modeling corporate knowledge within the agent oriented abstraction", Proceedings of the 2004 International Conference on Cyberworlds, pp. 224-231, 2004.

[9] J. Chen, S. Wolfe, and S. D. Wragg, "A distributed multi-agent system for collaborative information management and sharing", In Proceedings of the 9th ACM International Conference on Information and Knowledge Management (CIKM), pp. 382-388. ACM Press, New York, NY, USA, 2000.

[10] Object Management Group, Inc., 250 First Ave. Suite 100, Needham, MA 02494, USA.

[11] J. Pollock & R. Hodgson, "Adaptive Information: Improving Business Through Semantic Interoperability, Grid Computing, and Enterprise Integration", Wiley-Interscience, 2004.

[12] A. Wasserman, "Toward a Discipline of Software Engineering", IEEE Software 13(6) 23-31, 1996.

[13] N. Jennings, K. Sycara & M. Wooldridge, "A Roadmap of Agent Research and Development", Autonomous Agents and Multi-Agent Systems, 1, pp. 7-38, 1998.

[14] M. Wooldridge, N. Jennings, and D. Kinny, "A Methodology for Agent-Oriented Analysis and Design", Proceedings of the Third International Conference on Autonomous Agents (Agents-99) Seattle, WA. 28, 1999.

A Comparative Analysis of *i**Agent-Oriented Modelling Techniques

Gemma Grau, Carlos Cares, Xavier Franch, Fredy J. Navarrete
Universitat Politècnica de Catalunya (UPC)
C/Jordi Girona 1-3, UPC-Campus Nord (C6), Barcelona (Spain)
{ggrau, ccares, fjnavarrete, franch}@lsi.upc.edu

Abstract

Agent-oriented software engineering has become an extended practice. The autonomy and flexibility provided by agents makes it appropriate either for the development of agent-based systems and for the development of complex and distributed software systems. Due to this wide applicability there are many modelling languages and methodologies for representing and developing systems using the agent-oriented paradigm. Among them, the i language is very appropriate, not only for the agent concepts that it models, but also for its capabilities in the disciplines of requirements engineering and organizational process modelling. Nowadays, i* is one of the most widespread notations used for these purposes, and agent-oriented methodologies may take advantage of its existence in the requirements phase. However, due to the degree of freedom inherent to the i* language, the construction of the models may not be an easy task. To give light to this subject, we present several i* agent-oriented modelling techniques and we compare them according to a certain set of criteria.*

1. Introduction

The difficulty involved in the design and development of a software system increases with the complexity of such a system. To solve this problem, several software engineering paradigms have been proposed, being one of them agent-oriented software engineering [1]. The agent-oriented paradigm provides the means for representing the organizational structure of any kind of system and the interactions between the entities (computational or human) involved. Agents exhibit properties such as autonomy, reactivity, pro-activeness and social ability, which allow representing, analysing and designing software solutions for agents and multi-agents systems, but also for other kinds of complex systems that involves distribution of data and control, legacy systems or open systems [2].

The success in using the agent-oriented paradigm is enforced by the great amount of existing agent-oriented methodologies. As most of them address different issues, a crucial step is to understand the relationships between these various methodologies and to understand the main properties addressed by them [3]. With this aim, there is a great amount of literature for comparing and evaluating agent-oriented methodologies [3, 4, 5, 6, 7, 8, 9, 10, 11].

Agent-oriented methodologies are mainly composed of four different phases, namely the analysis, design, implementation and verification phases [2], which are done at the level of abstraction more adequate to the problem to be solved. It is during the analysis phase that the acting entities of the problem domain are identified and modelled as agents. On the other hand, the agents and their actions (or behaviour) are refined and specified in the design phase. In agent-oriented methodologies these two phases have to include a requirements gathering stage that is not always supported by the proposed processes [12]. For instance, GAIA [1] views the requirements capture phase as being independent of the paradigm used for analysis and design and, so, it only focus on the late requirements phase. A similar problem is detected in Prometheus [13], where a simple version of KAOS is used to describe the goals of the system, but description of business models is not supported. On the other hand, TROPOS [12] uses the *i** framework [14] as its modelling technique, and because of this, it is one of the methodologies that better supports early and late requirements analysis [12, 7]. Further work on Prometheus [15], proposes to use the facilities provided by *i** for refining the methodology and improve its early requirements analysis phase. As *i** is an agent-oriented modelling language, most of the constructs that it defines can be adapted to other agent-oriented modelling languages. This is the reason why it is possible to use it for obtaining and modelling early requirements in agent-oriented methodologies and, then, defining transformation guidelines between *i** and other agent-oriented modelling languages such as proposed in [15].

Despite the usefulness of *i** models, their construction may be difficult because of the degree of freedom provided by the language. In order to address this issue, we have studied those *i** methods that directly address the construction of *i** models and we have compared them following a certain set of criteria. More precisely, the only methods that provide precise guidelines for constructing *i** models are, as far as we know:

- The TROPOS methodology [12];
- The *goal-based business modelling oriented towards late requirements generation* method [16];
- A methodology for mapping activity theory diagrams into organizational models based on *i** [17];

- A methodology for building *i** models from BPEL process descriptions [18];
- The R*i*SD methodology [19]; and
- The PR*i*M methodology [20].

The objective of this paper is to present and compare the existing set of *i** modelling methods. As similar work has already been done for agent-oriented methodologies, related work is presented in section 2. Section 3 presents the *i** framework in more detail. In section 4, an overview of the different *i** model construction methods is provided. Section 5 contains the comparison criteria, which are used in section 6 for analysing and comparing the different methods. Finally, section 7 presents the conclusions.

2. Related work

In order to successfully apply the agent-oriented paradigm we have first to decide if an agent-oriented approach is suitable and, second, to choose the most appropriate agent-oriented methodology. The adequacy of the agent-oriented paradigm against other paradigms is discussed in [1] and [21]. In [1], the agent-oriented paradigm is balanced against the object-oriented paradigm, whilst [21] provides guidelines for evaluating the benefits of an agent-oriented approach against the adoption of other methodologies such as the object-oriented ones. Once the decision of adopting an agent-oriented paradigm is assumed, the already existing agent-oriented methodologies have to be analysed and, as most of them address different issues, the most appropriate has to be chosen by analysing, evaluating and comparing them.

Agent-oriented methodologies are generally built by extending other existing methodologies such as object-oriented or knowledge oriented methodologies in order to include the relevant aspects for agents [4]. As all the methodologies can be categorized according to its origin, it is possible to establish a genealogy of agent-oriented methodologies, such the one presented in [11]. Therein, the TROPOS methodology [12] comes from *i**; and *i** inherits both from the field of Requirements Engineering and the techniques related to the Object-Oriented patterns for agents.

There exist several proposals whose only aim is to compare and evaluate agent-oriented methodologies. For instance, in [4] several methodologies are described, stating its origin, the models they propose and the processes or phases undertaken to construct the models. Although this work does not explicitly compares the methodologies, there are other proposals that do it according to a set of established criteria [3, 5, 6, 7, 9, 10, 11].

The most common categorization of the criteria includes concepts, notation, process and pragmatics [9, 10, 11]. The **concepts** deal with those aspects related on how a methodology adheres to the basic notions of agents-based systems and, so, includes criteria such as *proactiveness*, *reactiveness*, and *adaptability*. Those criteria can be grouped into those that are internal to the agent such as *autonomy* or *mental notions*, and the ones that refer to its interaction with other agents such as *social ability*, *protocols* or *capability to deal with multiple interests*. The **notation** category is related to the modelling techniques that the methodologies exhibit. It includes criteria for evaluating the *syntax* and *semantics* of the formalism used; the *adequacy* and *expressiveness* of the language; and the *refinement*, *modularity* it provides. The **process** category groups all those aspects dealing with the process development aspect of a methodology, including criteria such as the *lifecycle coverage*, the *development context* and the *activities* proposed in the methodology. Finally, the **pragmatics** refers to the management criteria and technical issues that are provided to the users when adopting it. It contains criteria such as *target domain*, *quality*, *cost estimation*, *resources* or *tool support*.

Although not all the comparison frameworks propose the same criteria, overlapping exist in most of the categories because all them share an interest for evaluating both the agent concepts and their representation, and the methodological process itself. For instance, [6] establishes two categories of concepts: *internal attributes* and *interaction attributes*; and [5] groups the criteria of notation, process and pragmatics into the category *software engineering attributes*. On the other hand, the evaluation frameworks in [7, 11] consider the platform in their evaluation. In particular, [7] considers a *technology dimension* in order to include aspects such as *mode of processing*, *human-machine interface* and *development environment*; whilst [11] distinguishes between platform independent and platform dependent attributes and evaluates the methodologies according to a concrete platform.

The evaluations are completed by applying the criteria individually in each methodology [6, 10] or by applying a certain attribute over a set of methodologies [5, 7, 9, 3]. Most of the evaluations provide a qualitative evaluation of the proposed criteria, but there are also some different approaches. For example: [3] compares the agent-oriented methodologies by using the NFR approach; [6] and [10] adopt a quantitative approach by evaluating the methodologies with numerical values and, subsequently, obtaining quantitative results; and, finally, the proposals [3, 7, 8] define an exemplar as well as attributes for analysing, comparing and evaluating the methodologies. The agent-oriented methodologies that are evaluated more often in the proposals are: MaSE, evaluated in [7, 9, 11]; Agent Modelling Technique for Systems of BDI Agents, evaluated in [4, 6, 7]; GAIA, evaluated in [4, 7, 10]; and TROPOS, evaluated in [3, 9, 11].

As we have already mentioned, agent-oriented and goal-oriented requirements engineering are approaching because goals help to abstract the intentionality of agents, such as it is revealed by *i**. Regarding comparison frameworks on goal-oriented software engineering, most of the proposed methods introduce and analyze other proposals in their related work. However, as far as we know, only [22] specifically compare goal-oriented methods. The criteria used include the usage of the methods, the subject addressed, the kind of representation and the development support. Among the compared methods, there are *i**, GRAM and KAOS.

3. The *i** framework

The *i** framework proposes the use of two types of models for representing systems, each one corresponding to a different abstraction level: a Strategic Dependency (SD) model represents the intentional level and the Strategic Rationale (SR) model represents the rational level.

A SD model consists of a set of nodes that represent actors and a set of dependencies that represent the relationships among them. Dependencies expresses that an actor (*depender*) depends on some other (*dependee*) in order to obtain some objective (*dependum*). Thus, the *depender* depends on the *dependee* to bring about a certain state in the world (goal dependency), to attain a goal in a particular way (task dependency), for the availability of a physical or informational entity (resource dependency) or to meet some non-functional requirement (softgoal dependency).

A SR model allows visualizing the intentional elements into the boundary of an actor in order to refine the SD model with reasoning capabilities. The dependencies of the SD model are linked to intentional elements inside the actor boundary. The elements inside the SR model are decomposed accordingly to two types of links:

- *Means-end* links establish that one or more intentional elements are the means that contribute to the achievement of an end. The "end" can be a goal, task, resource, or softgoal, whereas the "means" is usually a task. There is a relation OR when there are many means, which indicate the different ways to obtain the end. *Contribution links* are *Means-end links* with a softgoal as an *end*, which allows stating explicitly if the contribution of the *means* towards the *end* is negative or positive.
- *Task-decomposition* links state the decomposition of a task into different intentional elements. There is a relation AND when a task is decomposed into more than one intentional element.

For more details about *i**, we refer to Yu's seminal work [14] and also to [23] for an *i** reference model in UML.

4. Overview of *i** modelling techniques

The **TROPOS methodology** [12] addresses agent-oriented development and it is intended to support all analysis and design activities in the software development process. TROPOS models information systems as social structures by means of a collection of social actors and the social dependencies among them. The modelling of these concepts are associated to different activities. Therefore, the TROPOS methodology consists of five phases: early requirements analysis, late requirements analysis, architectural design, detailed design, and implementation. We remark that requirements analysis is split into two phases: early requirements and late requirements analysis, both sharing the same conceptual and methodological approach. However, in the early requirements phase the domain stakeholders are identified and modelled as social actors, stating the *why* behind the system functionalities. As a last result in the overall process, the stated early requirements support the verification of how the final implementation matches the initial needs. In the late requirements phase the conceptual model is extended including the system as a new actor and its dependencies with the other actors of the environment. These dependencies define all the functional and non-functional requirements of the system-to-be.

The objective of the ***goal-based business modelling oriented towards late requirements generation*** method [16] is to use a business model for constructing a software requirements specification. The method proposes to use a goal-based elicitation method in order to capture the organizational context in a so-called Goal-Refinement Tree, which contains the successive decomposition of goals into subgoals. Such decomposition is done by means of classification and elicitation strategies. Once the Goal-Refinement Tree is constructed, several steps are applied in order to map its goals into the *i** SD elements. The *i** SR model is created afterwards by representing the actors internal goals. The final model is used to perform business improvement analysis by means of inserting the system actor into the system. This analysis leads to strategic models that can be used to derive functional (use case) specifications with their corresponding scenarios, which can be done by using the guidelines provided in [24].

The methodology presented in [17] proposes the **mapping of activity theory diagrams into organizational models based on *i****. Thus, it seeks to build *i** models based on activities theory in order to document functional and non-functional requirements and, so, provide a better understanding of the process. Taking the activity theory models as a starting point, the activities and their actions are analysed in order to construct the model. This analysis is done by decomposing the activities into actions and the actions into sub-actions until the actions are kept simple enough. The method provides concrete guidelines for mapping the activity theory concepts to an SD *i** model, providing rules for distinguish the *depender* and the *dependee*, as well as the different kinds of dependencies. SD models are then used to build the SR by following analogous guidelines.

The **methodology for building *i** models from BPEL process descriptions** [18] provides several guidelines to guide the mapping between BPEL Web Services descriptions into *i** diagrams. The SD and SR models are developed simultaneously by applying the guidelines, which map the concepts provided in the BPEL descriptions to the *i** constructs. Once the models are generated, they can be automatically translated into the action language ConGolog in order to run simulations and analyse the model properties. This method relies on a very formal basis and its main goal is to represent and evaluated agent-based designs for inter-organizational networks.

The **R*i*SD methodology** [19] is aimed at building Reduced *i** SD models for software systems. R*i*SD is defined as a set of activities structured in two phases, one for constructing the

social system and the other for constructing the socio-technical system. The social system model does not include the software system and focuses on the stakeholder needs. Its construction is iterative and begins with the identification of the initial set of social actors involved, their main goals and their strategic dependencies. The process iterates until all the actors and dependencies are obtained. The socio-technical system model incorporates the software system and the SD model dependencies are reassigned around this new actor. The system may be further decomposed into subsystems which are modelled as new actors and, therefore, the existing dependencies are reassigned again. New subsystems may depend on each other and these dependencies are also established.

The **PR*i*M methodology** [20] addresses *i** modelling from the business process reengineering perspective, where the specification of the system-to-be starts from the observation of the current system and uses the *i** framework for modelling this process. The *i** model is used for generating several process alternatives that can be evaluated using structural metrics [25]. The final model can be used for the specification of the new system, taking advantage of the possibility of connecting strategic reasoning with information system development. Based upon this business process reengineering point of view the PR*i*M methodology is composed of five phases. In the first one, the current process is analysed and the information obtained is summarized into Detailed Interaction Scripts (DIS). Every activity produces a DIS, which contains the activity preconditions, postconditions and triggering events, and a description of the actions and the resources involved in the activity. In the second phase, an *i** model is constructed by using a set of prescriptive guidelines in order to obtain the process view of the current system and its rationale. In the third phase the systematic generation of process alternatives is done by means of the addition of new actors and the reallocation of responsibilities between them. Those different alternatives are evaluated in the fourth phase and one of them is selected as the solution. Finally, the specification of the new system is generated based on the chosen process alternative.

There are some other approaches that address the construction of *i** models. However, as their main goal is not the construction of the models but their use they are not included in this analysis, and we just briefly summarise the three most widespread. First, the seminal proposal of *i** [14], which clearly introduces the *i** constructors and their uses, but provides small guidance on how models are constructed. Second, the organizational patterns provided in [26] which are based on organizational structures than can be used as a basis to generate *i** models. This approach is suitable when a specific organizational architecture is modelled, but there is little guidance on how to use the patterns and no description on how exactly this patterns are defined. Finally, the RESCUE process [27] provides several guidelines for applying *i** modelling at the requirements specification stage with the aim of discovering and eliciting requirements. However, its main goal is to get the system specification rather than the *i** model.

5. Comparison criteria

The analysis and comparison of agent-oriented methodologies has been widely studied [3, 4, 5, 6, 7, 8, 9, 10, 11]. As we have already mentioned in section 2, some of these proposals state a set of comparison criteria for comparing agent-oriented methodologies according to the concepts, the notation, the process and the pragmatics they exhibit. However, in the context of the *i** modelling techniques, all the methodologies use the same concepts and graphical constructors, which are the ones provided by *i**. Consequently, the concepts and notation are not relevant for comparing them and, so, the focus is on the methodological process. The criteria used for evaluating the i* modelling methods are based upon the ones proposed on the agent-oriented methodology comparison frameworks and have been grouped into the following categories: the **development process**, the **prescriptiveness** of its steps, the **resources** involved and the ***i** issues** addressed.

According to [10], a **development process** is a series of actions, changes, and functions that, when performed, result in a working computerized system. Thus, for evaluating the process we take into account the **development context** where the method is applied (e.g., creating new software, reengineering of existing software, prototyping, designing for reusing components). For establishing the **lifecycle coverage** of the method we consider the phases proposed in TROPOS [12]: early requirements, late requirements, architectural design, detailed design and implementation. As mentioned in [1], there are three different approaches for identifying the behaviour of the agents: top-down approaches, which are based on progressive decomposition of behaviour; bottom-up approaches, which begin by identifying elementary agent behaviours; and mixed approaches. Thus, the **development approach** adopted when building the *i** models is also analysed. Some of the methods present **process restrictions** that constraint the situations where they can be applied.

Prescriptiveness is concerned with guidelines and rules for a correct and unambiguous use of the *i** constructors. To deal with these issues, we compare the degree of detail provided when describing each of the stages proposed by the methods. This includes the **method decomposition** in smaller parts (e.g. steps) in order to tackle the problem complexity, the existence of precise **guidelines or heuristics** for constructing the models and the existence of **examples**.

Resources are crucial when using a method, in the analysis two different kinds of resources are studied: resources generated by the method and resources available for applying the method. The **resources generated** by the methodology are mainly SD and SR models which can be developed at different levels of detail. If the application of the method generates **intermediate artefacts**, they are also considered. On the other hand, the **resources provided** are those that are available for applying the methodology such as templates, patterns or software tools.

The nature and objectives of the seminal proposal of *i** [14] has provided the language with a certain degree of freedom (see

[23] for a comparative analysis on *i**-based agent-oriented languages). This flexibility when applying the language results into new constructors that appear on behalf of the methods needs. In order to evaluate those ***i* issues*** we consider how each method addresses the *i** framework by means of the addition of new *i** constructors. As *i** models become difficult to manage when the complexity of the system grows, we also analyse how the **scalability** of the models is tackled.

6. Applying the comparison criteria

Based on the criteria proposed in the previous section, we have compared the methodologies and we have obtained the analysis results that are summarized in Table 1. In order to facilitate to reference them, some of the methods names are abbreviated as follows: GBM stands for the *Goal-based Business Modelling oriented towards late requirements generation method* [16]; ATM stands for the methodology for mapping Activity Theory diagrams into organizational Models based on *i** [17]; and, BPD stands for the methodology for building *i** models from BPEL Process Descriptions [18].

Context: Addressing the development context, the methods GBM, ATM and R*i*SD are aimed at supporting the specification of a new software system. BPD addresses the representation and dynamic evaluation of agent-based designs and, thus, aims at prototyping. PR*i*M adopts a process reengineering perspective which includes the option of designing the new system by reusing software components. TROPOS is the only method that aims at developing new software and, thus, its lifecycle coverage goes from early requirements to implementation. Both early requirements and late requirements are addressed by GBM, R*i*SD and PR*i*M; whilst ATM main focus is on early requirements and BPD main focus on the detailed design. The development approaches adopted by TROPOS, GBM and R*i*SD are top-down; ATM and BPD are bottom-up; whilst PR*i*M has a mixed approach. The restrictions applied over the methods deal with the starting situation from where they have to be applied: TROPOS, GBM and R*i*SD do not need auxiliary information and, so, they can be built from the scratch. On the other hand, activity diagrams, BPEL descriptions and the complete documentation of the existing process are needed as a starting point for ATM, BPD and PR*i*M, respectively.

Prescriptiviness: All the methods decompose the problem into steps or stages for a better application of their guidelines.

Comparison Criteria		TROPOS [12]	GBM [16]	ATM [17]	BPD [18]	R*i*SD [19]	PR*i*M [20]
Process	Development context	- Creating new software	- Specifying new software	- Specifying new software	- Prototyping	- Specifying new software	- Reengineering Design for components reuse
	Lifecycle coverage	- From early requirements to implementation	- Early Requirements - Late Requirements	- Early Requirements	- Detailed Design	- Early Requirements - Late Requirements	-Early Requirements - Late Requirements - Arch. Design
	Development approach	- Top-down	- Top-down	- Bottom-up	- Bottom-up	- Top-down	- Mixed
	Restrictions	-	-	- Activity Diagrams as starting point	- BPEL descriptions as starting point	-	- Current process as starting point
Prescriptiveness	Problem decomposition	- Different steps for early requirements and late requirements	- Different steps for the construction of the SD and the SR	- Different steps for the construction of the SD and the SR	- SD and SR models are build in parallel in different steps	- Different steps for early requirements and late requirements	- SD and SR models are build in parallel in different steps
	Guidelines and heuristics	- Heuristics for the *i** element identification - Heuristics to decide the *dependum* kind	- Mapping guidelines from the Goal-Refinement Tree to *i**	- Mapping guidelines from activity diagram - Heuristics to decide *dependum* kind	- Mapping Guidelines from BPEL	- Heuristics for the *i** element identification - Heuristics to decide the *dependum* kind	- Mapping Guidelines from the DIS - Consistency checks
	Example	- eCulture System	- Conference Review Process	- Project-based Learning activities	- Loan Service	- Information Reliability	- Meeting Scheduler
Involved Resources	Resources produced	- Complete SD and SR models	- Complete SD and SR models - Considers alternative paths	- Complete SD and SR models - Do not considers alternative paths	- Complete SD and SR models - Do not considers alternative paths	- Complete SD model - Partial SR model - Considers alternative paths	- Complete SD model - Partial SR model
	Intermediate Artifacts	- Capability diagrams - Plan diagrams - Agent interaction diagrams	- Goal-Refinement Tree	- Decomposition of the activities in actions - Decomposition of actions into subactions	-	-	- DIS: Detailed Interaction Scripts for each activity
	Method Resources	- JACK tool [12]	-	-	- SNet Tool [28]	-	- REDEPEND-REACT [29]
*i** Issues	Constructors Used	- Basic *i** [14]	- Basic *i** [14] Without softgoals and means-end link	- Basic *i** [14]	- Basic *i** [14] - Without means-end - Adds the constructor *task precondition*	- Basic *i** [14] - Adds the constructor *support*	- Basic *i** [14] - Restricts the SR decomposition
	Scalability	- By goal-decomposition	- By goal-decomposition	- By decomposing the problem in activities	- Not addressed	- Using the supports decomposition	- By decomposing the problem in activities

Table 1. Comparison of *i** methodologies according to the established set of criteria

This decomposition divides the process into late requirements and early requirements specification in TROPOS and R*i*SD, into the construction of SD or SR models in GBM and ATM and in several steps for a parallel SD and SR construction in BPD and PR*i*M. All the methods propose concrete guidelines for creating the models. For instance, in ATM and BPD, guidelines determine the mapping between the elements of the starting artefacts and the final *i** model elements. GBM and PR*i*M generate intermediate artefacts and provides mapping guidelines between their elements and the final *i** model elements. TROPOS and R*i*SD provide identification heuristics in order to facilitate the identification the *i** elements without any artefact as a starting point. On the other hand, in GBM, BPD and PR*i*M, the guidelines strongly determine the kind of the dependum and other strategic elements used. More freedom is provided in TROPOS, ATM, and R*i*SD, where additional heuristics are provided to help to decide the dependency kind (goal, resource, task or softgoal). Additionally, PR*i*M provides consistency checks in order to ensure the correct application of the guidelines. Every method is demonstrated by means of an illustrative example in order to clarify the explanations: TROPOS models an eCulture System; GBM a Conference Review Process; ATM models Project-based Learning Activities; BPD a Loan Service; R*i*SD models information reliability; and PR*i*M a Meeting Scheduler.

Involved Resources: TROPOS is the only method that builds complete SD and SR models. GBM, ATM, BPD, and PR*i*M develop a complete SD model but the SR does not allow alternative analysis because only task-decomposition is used. Alternatives are developed in PR*i*M as alternative models. R*i*SD SD models are also complete and the SR model contains both *task-decomposition* and *means-end* links for allowing the representation of alternatives. However, R*i*SD's SR model is not considered complete because it is built as an auxiliary model in order to support the development of the SD model. According to the order in which SD and SR models are developed, GBM and ATM fully develop the SD model before developing the SR model, whilst in the other methods the construction of the two can be intertwined. The intermediate artefacts produced by each of the methodologies are the following : TROPOS permits to generate capability diagrams, plan diagrams and agent interaction diagrams; GBM generates an intermediate Goal-Refinement Tree; ATM provides the decomposition of activities into actions, and the decomposition of these actions into subactions; and PRiM produces Detailed Interaction Scripts (DIS). We remark that some of the methods provide specific tools for supporting its processes. In particular, TROPOS proposes the JACK tool [12], BPD proposes the SNet Tool [28] and PR*i*M is partially supported by REDEPEND-REACT [29].

i* **issues:** All the methods are based on the seminal proposal of *i** [14], however some of them do not use all the constructors, whilst others have added their own constructors. As we have already mentioned, GBM, ATM and BPD do not use means-end links in the SR models and, so, do not consider alternative analysis. We also remark that GBM does not use the softgoal construct in any of the produced models. On the other hand, BPD extends *i** with the constructor *Task precondition* in order to describe task preconditions and effects. R*i*SD also adds a new constructor, *supports*, which establish a relation between related dependencies. This constructor tackles scalability in R*i*SD, because it helps to address the problem incrementally by decomposing the already existing *i** elements. A similar approach is adopted in TROPOS and GBM where scalability can also be handled by goal-decomposition. In BPD this issue is not mentioned. Finally, ATM and PR*i*M deal with scalability by decomposing the problem in activities and the activities into actions.

7. Conclusions

The *i** framework is agent-oriented and thus, is suitable in the context of agent-oriented software engineering. As it is also successfully applied in requirements engineering and organizational modelling, *i** can also support the requirements engineering phase in those agent-oriented methodologies where requirements gathering or organizational rational is not explicitly addressed. In order to construct *i** models a specific methodology has to be applied. We have presented six different methodologies and we have analysed them in order to inform their use. The comparison focuses on the process, the prescriptiveness of the provided explanation, the resources involved and the *i** issues addressed by each method. The criteria used in each of these categories had been defined upon the ones already defined in other agent-oriented methodologies comparison frameworks. As a result, and this is the main contribution of our work, these criteria show under which circumstances one method may be more valuable than the others and therefore their selection may now rely on objective criteria rather than on a subjective belief.

Further work includes a deeper analysis of the *i** modelling methods by using an exemplar such as proposed in [7, 9, 11] and by evaluating the comparison framework from different points of view (e.g., the authors, students, industry practitioners). Also a quantitative approach would be adopted by refining the evaluation criteria into new ones in order to allow a numerical evaluation and, then, applying a quantitative approach [6, 9]. Case studies on using and selecting the methodologies would also be completed.

Acknowledgements

This work has been done in the framework of the research project UPIC, ref. TIN2004-07461-C02-01, supported by the Spanish Ministerio de Ciencia y Tecnología. Some authors have grants that partially support their work: C. Cares, by the MECE-SUP FRO0105 Project of the Chilean government and Universidad de la Frontera; and G. Grau, by a UPC research scholarship.

8. References

[1] M. Wooldridge, N. R. Jennings, and D. Kinny. "The Gaia methodology for agent-oriented analysis and design". Journal on Autonomous Agents Multi-Agents Systems, Vol. 3, No. 3, 2000.

[2] N.R. Jennings. "On Agent-Based Software Engineering". Artificial Intelligence, Vol. 117, No. 2, 2000. pp: 277–296.

[3] C.T.L. Silva, P.C.A.R. Tedesco, J.F.B. Castro, R.C. Silva. "Comparing Agent-Oriented Methodologies Using the NFR Approach". In *Proceedings of the Third International Workshop on Software Engineering for Large-Scale Multi-Agent Systems*, SELMAS 2004. Vol. 1, pp: 1-9.

[4] C.A.. Iglesias, M. Garijo, J.C. González. "A Survey of Agent-Oriented Methodologies". In *Proceedings of the Fifth International Workshop on Intelligent Agents V. Agent Theories, Architectures, and Languages*, ATAL 1998. pp: 317 – 330.

[5] O. Shehory and A. Sturm. "Evaluation of modeling techniques for agent-based systems". In *Proceedings of the Fifth International Conference on Autonomous Agents*, 2001.

[6] L. Cernuzzi, G. Rossi. **"**On the evaluation of agent oriented modeling methods". In *Proceedings of the First International Workshop on Agent-Oriented Methodologies*, held at OOPSLA 2002. pp. 21-30.

[7] A. Sabas, S. Delisle, M. Badri. "A Comparative Analysis of Multiagent System Development Methodologies: Towards a Unified Approach", In *Proceedings of the Third International Symposium "From Agent Theory to Agent Implementation*", held at the *16th European Meeting on Cybernetics and Systems Research*, 2002.

[8] E.S.K. Yu, L.M. Cysneiros. "Agent-Oriented Methodologies - Towards a Challenge Exemplar". In *Proceedings of the Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems*, AOIS 2002.

[9] K.H. Dam, M. Wnikoff. "Comparing Agent-Oriented Methodologies". In *Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems*, AAMAS 2003.

[10] A. Sturm, O. Shehory. "A Framework for Evaluating Agent-Oriented Methodologies". In *Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems,* , AOIS 2003. pp: 94-109.

[11] J. Sudeikat, L. Braubach, A. Pokahr, W. Lamersdorf. "Evaluation of Agent - Oriented Software Methodologies - Examination of the Gap Between Modeling and Platform". In *Proceedings of the Fifth International Agent-Oriented Software Engineering Workshop*, AOSE 2004. pp: 126-141.

[12] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos. "TROPOS: An Agent-Oriented Software Development Methodology". *Journal of Autonomous Agents and Multi-Agent Systems*. May 2004. Volume 8, Issue 3.

[13] L. Padgham, M. Winikoff. *Developping Intelligent Agent Systems - A practical Guide*, John Wiley & Sons, 2004.

[14] E. Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD. thesis, University of Toronto, 1995.

[15] G. Cysneiros, A. Zisman. "Refining the Prometheus Methodology with *i**". In *Proceedings of the Third International Workshop on Agent-Oriented Methodologies*, held at OOPSLA 2004.

[16] H. Estrada, A. Martínez, O. Pastor. "Goal-based business modeling oriented towards late requirements generation". In *Proceedings of the 22nd International Conference on Conceptual Modeling*, 2003. Springer-Verlag, LNCS 2813. pp: 277-290.

[17] G.C. Neto, A.S. Gomes, J.B. Castro. "Mapeando Diagramas da Teoria da Atividade em Modelos Organizacionais Baseados em *i**". In *Proceedings of the 7th Workshop em Engenharia de Requisitos*, WER 2004. pp: 39-50.

[18] D. Schmitz, G. Lakemeyer, G. Gans, M. Jarke. "Using PBEL Process Descriptions for Building up Strategic Models for Inter-Organizational Networks". In *Proceedings of the Workshop on Modeling Inter-Organizational Systems*, MIOS 2004. Springer-Verlag, LNCS 3294. pp: 520-532.

[19] G. Grau, X. Franch, E. Mayol, C.P. Ayala, C. Cares, M. Haya, F. Navarrete, P. Botella, C. Quer. "R*i*SD: A Methodology for Building *i** Strategic Dependency Models". In *Proceedings of the 7th International Conference on Software Engineering and Knowledge Engineering*, SEKE 2005. pp: 259-266.

[20] G. Grau, X. Franch, N.A.M. Maiden. "A Goal Based Round-Trip Method for System Development". In *Proceedings of the 11th International Workshop on Requirements Engineering: Foundation For Software Quality*, REFSQ 2005. Essener Infromatik Beiträge. pp: 71-86.

[21] S.A. O'Malley, S.A. DeLoach. "Determining when to use agent-oriented software engineering". In *Proceedings of the Second International Workshop on Agent-Oriented Software Engineering*, AOSE 2001. pp. 188-205.

[22] E. Kavakli, P. Loucopoulos. "Goal Driven Requirements Engineering: Evaluation of Current Methods". In *Proceedings of the 8th CAiSE/IFIP8.1 Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*, EMMSAD 2003.

[23] C.P. Ayala, C. Cares, J.P. Carvallo, G. Grau, M. Haya, G. Salazar, X. Franch, E. Mayol, C. Quer. "A Comparative Analysis of *i**-Based Goal-Oriented Modelling Languages". In *Proceedings of the 17th International Conference on Software Engineering and Knowledge Engineering,* SEKE 2005. pp: 43-50.

[24] V.F.A. Santander, J.F.B. Castro. "Deriving Use Cases from Organizational Modeling". In *Proceedings of the 10th IEEE International Requirements Engineering Conference*, RE 2002. pp: 32-39.

[25] X. Franch, G. Grau, C. Quer. "A Framework for the Definition of Metrics for Actor-Dependency Models". In *Proceedings of the 12th IEEE International Requirements Engineering Conference*, RE 2004. pp: 348-349.

[26] M. Kolp, P. Giorgini, J. Mylopoulos. "Organizational Patterns for Early Requirements Analysis". In *Proceedings of the 15th International Conference on Advanced Information Systems Engineering*, CAISE 2003. Springer-Verlag, LNCS 2681. pp: 617-632.

[27] S. Jones, N.A.M. Maiden. "RESCUE: An Integrated Method for Specifying Requirements for Complex Socio-Technical Systems". Book chapter in *Requirements Engineering for Sociotechnical Systems*, Idea Group Inc., 2004.

[28] G. Gans, D. Schmitz, M. Jarke, and G. Lakemeyer. "SNet Reloaded: Roles, Monitoring, and Agent Evolution." In *Proceedings of the 6th Workshop on Agent-Oriented Information Systems*, AOIS 2004. pp: 2-16.

[29] G. Grau, X. Franch, N.A.M. Maiden. "REDEPEND-REACT: an Architecture Analysis Tool". In *Proceedings of the 13th IEEE International Requirements Engineering Conference*, RE 2005. pp: 455 - 456.

A Negotiation Model for the Process Agents in an Agent-Based Process-Centered Software Engineering Environment*

Nao Li[1,3], Mingshu Li[1,2], Qing Wang[1], Shuanzhu Du[1,3]
[1] *Laboratory for Internet Software Technologies, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China*
[2] *Key Laboratory for Computer Science, The Chinese Academy of Sciences, Beijing 100080, China*
[3] *Graduate University of Chinese Academy of Sciences, Beijing 100039, China*
{linao,wangqing,dusz }@itechs.iscas.ac.cn, {mingshu}@admin.iscas.ac.cn

Abstract

In our agent-based process-centered software engineering environment (named ISPMS, Intelligent Software Process Management System) a software process is described as the cooperative process agents that represent the developers, teams and organizations having software process knowledge and being capable of performing the activities in a software process. Negotiation is applied to build up the cooperative relationships between the process agents. However, current most negotiation models are designed with a fixed negotiation protocol, fixed and single decision-making strategy applied to all the negotiation participants. The negotiation process of the process agents in ISPMS is changing and multiform, requiring a certain extent of flexibility and dynamism, to which these inflexible models are not applicable. We present a flexible negotiation model for the process agents. The model specifies the possible changing factors in a negotiation process as rules. The process agents can define and maintain these rules based on their own demand, the changes of their environment, etc. The rules are stored and evolved independently, interact with the other part of the model in a low-coupling way and thus provide the flexibility and dynamism for the negotiation of the process agents in a lower cost. The model also defines the negotiation primitives to support the communicative acts of the process agents.

1. Introduction

Most software process modeling approaches model a software process as pre-defined sequential interactions of activities or tasks, which result in inflexibility problems since a real software process requires agile adaptation to changes, such as the changing requirement, design, environment, etc. Our agent-based software process modeling approach (ASPM) and the corresponding developed process-centered software engineering environment (named ISPMS, Intelligent Software Process Management System)[1][2] apply agent technologies to software process modeling, where a software process is modeled as cooperative process agents that represent the developers, teams and organizations involved in software processes, have software process knowledge, are capable of performing the activities of a software process and build up the cooperative relationships each other through active negotiation. ISPMS is closer to the human-centered essence of software processes and provide the higher flexibility and adaptability for software processes.

Negotiation is applied to build up the cooperative relationships between the process agents in ISPMS because it is a flexible way and close to the real software process. The negotiation between the process agents to build up the cooperative relationships is a process where the process agents alternate in making values of some parameters, such as workload, code quality, etc., until reach an agreement about them. It is also a contracting process where a contract moves from a template form to an instance with agreed parameter values, and the resulting contract can be (or part of) a real business contract between organizations, or a commitment from team members to a team leader. The process is multiform and changing, dependent on the market environments, business objectives, project characteristics, etc. For example, the negotiation between organizations about software outsourcing could be one-many auction or one-one bargaining, and the negotiation between organizations is obviously different from that of between a team leader and team members.

A well-designed negotiation model to model the above

* This research is supported by the Science-Technology Project of the National 'Tenth Five-Year-Plan' of China under Grant No.2005BA113A01

process is very important in ISPMS. A negotiation model generally considers three basic issues: a negotiation protocol, negotiation objectives and decision-making strategies[3]. The negotiation protocol determines message flows between participant agents, specifying when an agent can send a message and what messages it can send as valid responses to specified incoming messages. All participant agents need a shared understanding of it. Negotiation objectives could be seen as parameters and a negotiation process is a process to set them to agreed values. The decision-making strategies are used by the participant agents to privately determine which actions they make in an effort to achieve their own goals. Current most negotiation models are designed with a predefined and fixed negotiation protocol[4], fixed number of negotiation objectives[5], or even a single decision-making strategy applied to all participants[5][6]. These fixed factors are usually implicitly represented in agent's design. When they change, the maintenance cost is very high. The negotiation process of the process agents in ISPMS is changing and the process agents need to dynamically decide which negotiation protocol they want to use, how many objectives they want to negotiate and what kind of decision strategies they want to apply, to adapt to the changing factors such as different market environments, business objectives, project characteristics, etc. The inflexible models mentioned above cannot be applied to the process agents.

To achieve the flexibility of negotiation, some researches separate the rules controlling some aspects of a negotiation process from the negotiation process. They have made some progress. Lochner[7] designs a rule script language to specify the time control in auctions and thus time control is changeable and flexible. Su[8] uses ETR (Event-Trigger-Rule) rules to represent decision-making models and the rules can be dynamically added and modified at running time. Bartolini[9] insists traditional negotiation protocols ignore some rules for a negotiation (e.g., obliging a participant to improve on a previous offer), and presents a taxonomy of such rules, which provide the flexibility for a negotiation protocol.

However, when considered to be applied in ISPMS, these methods have the flowing problems: (1) they only achieve the flexibility in some aspects of a negotiation model, but the process agents in ISPMS need a full solution, covering the three basic issues of a negotiation model; (2) most of them aim at the e-commerce domains, and thus can not completely adapt to software process.

In order to solve the problems, we present a flexible negotiation model for the process agents in ISPMS (named NM-PA). In NM-PA, the traditional negotiation protocol is separated into two parts: a static generic protocol and some rules. Decision-making strategies are also represented as rules. These explicitly defined rules compose as a whole the dynamic control and decision-making of message flows, taking the form of if-then, like if x>a then y=b. The process agents can define and change them, based on their knowledge, the changes of environment, etc. The rules are stored and evolved independently, interacting with the other part of the model in a low-coupling way. Therefore, they can provide the flexibility and dynamism for the negotiation of the process agents in a lower cost. The generic protocol serves as the static control of message flows (compared with the traditional models, it is also fixed but necessary for the balance of flexibility and efficiency). It can support most of multiform negotiation processes among the process agents.

Against to those models with fixed negotiation objectives, NM-PA gives an alternative approach. It organizes the negotiation objectives in a structural cooperation contract. The negotiation objectives and the structure serve as the domain knowledge of the negotiation and all the negotiation participants have a sharing understanding of them. Therefore, the negotiation objectives can be added even at running time. NM-PA also defines some negotiation primitives for the communicative acts of the process agents, referred to FIPA ACL messages[10].

2. Structure of the negotiation model

The negotiation process of the process agents in ISPMS is an interactive process where a cooperation contract with some parameters moves from a template form to an instance with agreed parameter values, under some controls and constrains. The negotiation model for the process agents structurally defines the elements involved in the process and the relations between them.

Definition1. The negotiation model for the process agents in ISPMS NM-PA is a 7-tuple $(A, T, NPrim, CCT, MF, GProtocol, NR)$, where,

(1) A is a set of potential participant process agents: $A = \{a_1, a_2, ..., a_n\}$.

(2) T is a set of distributed sequential time points, as the system clock: $T = \{t_1, t_2, ..., t_n\}$.

(3) NPrim is a set of negotiation primitives:
NPrim = {Propose_N, Accept_N,Reject_N,Terminate_N
Propose_CC,Accept_CC,Reject_CC,Modify_CC}.

(4) CCT is a cooperation contract template.

(5) MF is a set of message flows: $MF = \{mf_1, mpf_2, ..., mf_n\}$.

(6) GProtocol is a generic protocol controlling MF.

(7) NR is a set of negotiation rules, $NR = \{nr_1, nr_2, ..., nr_n\}$, nr_i controls mf_i.

In the following further description of the element in Definition1 and the relations between them are given.

2.1. Process agents

The process agent a_i has three kinds of knowledge. One is about the process agent itself (denoted by K), one is about its environment (denoted by E) and the other is about the interaction with other process agents (denoted by H). In the flowing we introduce only the knowledge of the three kinds that are closely related with the negotiation (for full definition of the process agent, refer to [1][2]):

(1) K includes the knowledge such as the capabilities a process agent has (e.g., coding, quality control level), the resources available to it (e.g., human, time), etc., $K = \{k_1, k_2, ..., k_n\}$.

(2) E includes the knowledge about the existence of other process agents, the capabilities of them, etc., $E = \{e_1, e_2, ..., e_n\}$.

(3) H includes negotiation histories with other process agents (denoted by Nh, which would be defined in 3.1).

2.2. Negotiation primitives

Negotiation primitives are communicative acts of participant process agents. Table1 lists those we define with their explanations.

Table 1. Negotiation primitives

Propose_N	Propose a negotiation request
Accept_N	Accept a negotiation request
Reject_N	Reject a negotiation request
Propose_CC	Send a CC (see Definition2) or a counter-CC
Accept_CC	Accept the current CC without further modifications
Reject_CC	Reject the current CC
Modify_CC	Modification to the CC received last
Terminate_N	Terminate the current negotiation process

2.3. Cooperation contract template

The cooperation contract template is the structure specifying how the negotiation objectives organize.

Definition2. A cooperation contract template CCT is a 5-tuple (CT, CD, O, V, OT), where,

(1) CT is a set of contract types: $CT = \{ct_1, ct_2, ..., ct_n\}$.

(2) CD is a set of contract descriptions: $CD = \{cd_1, cd_2, ..., cd_n\}$.

(3) O is a set of negotiation objectives: $O = \{o_1, o_2, ..., o_n\}$.

(4) V is a set of values of O: $V_i = \{v_1, v_2, ..., v_n\}$, v_i is the value of o_i.

(5) OT is a set of types of negotiation objectives: $OT = \{ot_1, ot_2, ..., ot_n\}$, ot_i is the negotiation type of o_i, $ot_i = \{N, NN\}$.

CT specifies the contract type, which can be an outsourcing, a task assignment, etc. CD describes what the contract is mainly about, a J2EE or a C++ project, or others. OT denotes whether an objective or a contract rule is negotiable or not, with N referring to yes and NN referring to not. All the participants have a shared understanding of the content of a cooperation contract.

An instance of CCT is a cooperation contract $CC_i = (ct_i, cd_i, O_i, V_i, OT_i)$, where $ct_i \in CT, cd_i \in CD, O_i \subseteq O, V_i \subseteq V, OT_i \subseteq OT$, and thus CC is a set of cooperation contracts, $CC = \{CC_1, CC2, ..., CC_n\}$.

2.4. Message flows

A message flow is a message sent by a participant process agent to another, under the control and decision-making of GProtocol (2.5) and NR (2.6). It is defined as the following:

Definition3. A message flow mf_i is a 5-tuple (a_{is}, a_{ir}, $NPrim_i$, CC_i, t_i), where,

(1) a_{is} is the agent sending $NPrim_i$ and CC_i, $a_{is} \in A$.

(2) a_{ir} is the agent receiving $NPrim_i$ and CC_i, $a_{ir} \in A$.

(3) $NPrim_i$ is the primitive sent, $NPrim_i \in NPrim$.

(4) CC_i is a cooperation contract sent, $CC_i \in CC$.

(5) t_i is the time point when $NPrim_i$ and CC_i are sent, $t_i \in T$.

2.5. Generic protocol

GProtocol is a generic negotiation protocol controlling and determining the message flows MF. It is specified by a state transition diagram (Figure1), which gives the legal states that a process agent may be in during a negotiation process and thus the legal transitions between states a process agent is allowed to take.

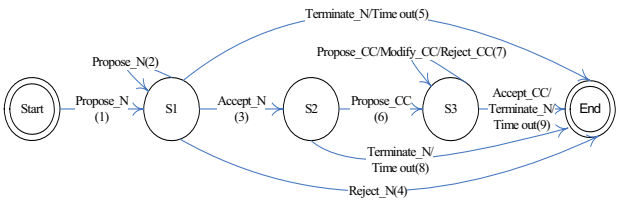


Figure 1. The generic negotiation protocol

In Figure1, Start is an initial state, S_1 is a state where a negotiation proposer has proposed a negotiation request and waits for reply, S_2 is a state where a negotiation relation is formed between the negotiation proposer and another process agent, S_3 is a state where the negotiation proposer has proposed the initial CC and the negotiation participants alternate in making values of the negotiation objectives in CC, End is an end state indicating a negotiation process ends.

In Figure1, each state transition is explained as the following: (1) A negotiation proposer proposes a negotiation request; (2) The negotiation proposer may make a second negotiation request without waiting for a reply; (3) An agent accepts the negotiation proposer's negotiation request; (4) An agent rejects the negotiation proposer's negotiation request; (5) The negotiation process ends if time is out, or one of the participants terminates the process. (6) The negotiation proposer proposes an initial cooperation contract; (7) The participant agents alternate to send the cooperation contract by proposing a new, modifying or rejecting the most recent; (8) The negotiation process ends if time is out or one of the participants terminates the process; (9) The negotiation process ends if time is out or one of the participants accepts the cooperation contract, or one of the participants terminates the process.

2.6. Negotiation rules

Negotiation rules also control and determine the message flows MF.

Definition4. The set of negotiation rules nr_i consists of two subsets, $nr_i=\{CR_i, DR_i\}$, where:

(1) CR_i is a set of control rules, $CR_i = \{cr_{i1},cr_{i2},...,cr_{in}\}$.

(2) DR_i is a set of decision rules, $DR_i = \{dr_{i1},dr_{i2},...,dr_{in}\}$.

In Definition4, cr_{ij} and dr_{ij} are both single negotiation rule, whose definition is based on Horn clauses:

Definition5. A negotiation rule r is a 2-tuple (Conds, Concl), where:

(1) Conds is a set of conditions, Conds={$condition_1$, $condition_2$,…,$condition_n$}.

(2) Concl is the conclusion inferred from the Conds, Concl={conclusion}.

A negotiation rule r indicates that when all $condition_i$ are true, the conclusion is true. Both $condition_i$ and conclusion are Horn atoms.

CR_i and DR_i control and determine the message flows mf_i . CR_i takes charge of public controls on message flows. Its main difference from the GProtocol lies on that it may change with different environments or goals. It is usually public to all a_{is} and a_{ir}. DR_i is used by a_{is} to privately decide the primitive and values of the negotiation objectives contained in the cooperation contract to send. DR_i usually keeps private and invisible to a_{ir}.

3. Execution of the negotiation model

Having specified the structure of NM-PA, it is now to explain how it is executed.

3.1. Execution mechanism

In NM-PA, A, T, NPrim and CCT are the most basic elements. After a negotiation process starts, they gradually constitute MF under the control and decision-making of both GProtocol and NR, which can be seen as a state transition function and an output function respectively, driving the execution of the negotiation process. To explain the functions, we give the following denotations:

(1) S is a finite state set of the process agents: $S=\sum_{i=1}^{n}\{Start_i,S1_i,S2_i,S3_i,End_i\}$, where Start is the initial state and End is the end state (Figure1).

(2) Nh is the negotiation histories, $Nh = \{Nh_1,Nh_2,...,Nh_n\}$, where Nh_i is the subset of MF (Definition3), $Nh_j=\sum_i mf_i(a_{is}=a_j, a_{ir}=a_j)$.

(3) I is a finite input set of states: $I=NPrim \times CC \times K \times E \times Nh$ (see 2.1 for K and E, see Definition1 and Defintiaion2 for NPrim and CC respectively).

(4) Out is a finite output set of states: $Out=NPrim \times CC \times Nh$.

Thus the functions are defined as:

(1) The state transition function GProtocol: $S \times NPrim \rightarrow S$.

(2) The output function OF: $I \rightarrow Out$. OF has two sub functions: $CR : I \rightarrow Out$ and $DR : I \rightarrow Out$. The final output is $CR(I) \cap DR(I)$ $(\neq \varnothing)$.

In any time, a participant process agent is in one of the state of S. When it receives an $NPrim_i$ $(\in NPrim)$, it is transit to another state under GProtocol (see Figure1). Furthermore, it outputs an $NPrim_j$ $(\in NPrim)$ and a CC_j $(\in CC)$ complying with OF.

CR and DR are negotiation rules as defined in Definition4. Notice that CR's input H is the negotiation history of all process agents, but since DR are private and process agents may not have the negotiation histories of

all the others, $\{h_i\}(\subseteq H)$ is DR's exact input in most cases.

From an analysis of Figure1 and the execution process, once a negotiation process starts, each of its three mid states (S_1, S_2, S_3) can reach its end state within a finite time period. We then conclude that an NM-PA based negotiation process can terminate in a finite time period after it starts.

3.2. An example

NM-PA is implemented in ISPMS. In order to validate it, we simulate a real negotiation process about an outsourcing project. In the real negotiation process, another organization and us alternate in making values of some parameters of the project we interest in and sending them by emails, until reach an agreement about the values. We print that agreement, sign it and start the cooperation with each other to complete the project. In the following we explain how negotiation rules function in the execution of the process in ISPMS.

Initially the negotiation proposer (A1) defines the control rules and original cooperation contract, and sends them and the negotiation request to all the other process agents. The control rules are defined as:

(1) if *start(a negotiation)*, then *timeout(3 days)*.

(2) if *acknowledge(a message)*, then *ack_time_limit(2 hours)*.

(3) if *over_ack_time_limit(an ack)*, then *terminate(the negotiation).*

The original cooperation contract has the flowing content:

(1) The contract type is *Outsourcing*.

(2) The contract description is *J2EE*.

(3) One negotiation parameter is *the number of developers* and its proposed value is *4*; another is *how many months* and the value is *3* (indicating the project needs 4 developers to complete within 3 months).

(4) Both types of the two parameters are *N* (indicating they are negotiable).

The other process agents need define their decision rules, which would decide whether to accept a negotiation request and how to make values of the negotiation parameters once they decide to accept the negotiation request. In our simulation, we initialize two process agents and define the same decision rule for their decision-making of whether to accept a negotiation request:

If *confidential_partner(the negotiation proposer)*, then *true (accept the negotiation request).*

For simplification, we only add the fact *confidential_partner(A1)* into one process agent's database (this indicates that when the negotiation process is started, only one process agent would accept the negotiation request). We denote this process agent A2.

A2 needs the decision rules to decide whether to accept the proposed contract by A1, and if not, make new values of the parameters and send them to A1. The decision rules are defined as the following (*p1* denotes the negotiation parameter of "*the number of developers*", *p2* denotes "*how many months*", *max(d)* denotes the maximum number of developers provided, *max(m)* denotes the maximum number of months the maximum developers can provides):

(1) if *received_value(p1)<=max(d)* and *received_value(p2)<= max(m)*, then *accept(the contract)*.

(2) if *(received_value(p1) × received_ value(p2)) > (max(d) × max(m))*, then*_reject(the contract)*.

(3) if *received_value(p1)>max(d)* and *received_value(p1) × received_value(p2) = max(d) × max(m)*, then *modify(the contract)*, *set_value(p1,max(d)), set_value(p2, max(m))*.

In A2's fact repository: *max(d)=3 and max(m)=4*. This indicates that when the negotiation process is executed, the third rule would be matched and A2 would modify the contract proposed by A1 and the modified contract has *p1* with value *3* and *p2* with value *4*.

A1 also needs the decision rules to decide whether to accept the proposed contract by A2, and if not, make new values of the parameters and send them to A2. The decision rules are as the following (*catch_deadline(received_value(p2))* denotes that it is true that the project is completed before the project deadline within the *value(p2)* months):

(1) if *received_value(p1) × received_value(p2)= last_sent_value(p1) × last_sent_value(p2)* and *catch_deadline(received_value(p2))*, then *accept(the contract)*.

(2) if *received_value(p1) × received_value(p2)= last_sent_value(p1) × last_sent_value(p2)* and *not_catch_deadline(received_value(p2))*, then *reject(the contract)*.

A1's data matches the first rule. This indicates that when the negotiation process is executed, A1 would accept the contract. After specifying the above negotiation rules, we start the process execution. The execution result is that A1 and A2 successfully build up the cooperation relationship (as we explain above). The parameter values are recorded as cooperation constrains in the agreed cooperation contract.

The negotiation process of the process agents is changing and multiform. In the negotiation process above, the first control rule indicates that the timeout of the process is 3 days. But market environment is often

changing. For example, some outsourcing project has to been completed in a tight time and 3 days is too long for the negotiation. In this case, A1 can simply modify the rule to adapt to the change. For example, modify the timeout from 3 days to 1 day. Aiming at different projects, A1 can also modify or redefine the whole control rules. Such flexibility happens to the decision rules as well. For example, A2 can change one of its decision rules to:

If *(received_value(p1)× received_value(p2)) > (max(d) × max(m))*, then *modify(the contract), set value(p1) to max(d), set value(p2) to max(m)*.

A1 can change one of its decision rules to:

If *(received_value(p1) × received_value(p2))/ last_sent_value(p1) × last_sent_value(p2)>=value(an accepted proportion)*, then *accept(the contract)*.

A1's changed rule indicates if A2 cannot undertake the whole workload but an accepted proportion, A1 can accept it. This often happens when two organizations have a good cooperation history and are willing to build the relationship. Dependent on different negotiation participants, A1 can define different decision rules.

In terms of the negotiation objectives, when the negotiation is processing, they can be added into the cooperation contract. For example, if in an alternation A1 adds the "bug rate" into the cooperation contract with its value, A2 is aware of the addition by structurally analysis of the cooperation contract since all the negotiation objectives are structurally organized in it and then decides whether to accept, reject or modify the value of the "bug rate" by corresponding decision rules.

4. Conclusion

This paper proposes a rule-based negotiation model for the process agents in an agent-based process-centered software engineering environment (NM-PA). It has the following main characteristics and advantages:

(1) It separates the traditional negotiation protocol into two parts: a generic protocol and some rules. The generic protocol is static. It can support most multiform negotiation processes for building the cooperative relationship of the process agents. The rules are alterable and result in a flexible and changeable negotiation protocol. The process agents can change and modify them adaptive to environments, objectives, and so on.

(2) It also represents decision-making strategies as rules. In the same way as the negotiation protocol, decision-making strategies can be defined and changed to adapt the changes.

(3) Negotiation objectives are organized in a structural cooperation contract. They can be added at running time. The contract-like structured negotiation objectives can be part of the real contract or commitment, which can be further used as the process and quality controlling of software development.

(4) Negotiation rules are stored and evolved independently, interacting with the other part of the model in a low-coupling way. Therefore they provide the flexibility and dynamism for the process agents in a lower cost.

5. References

[1] Zhao Xin-Pei, Li Ming-Shu, Wang Qing, Keith Chan, Harenton Leung. An Agent-Based Self-Adaptive Software Process Model. Journal of Software, vol.15(3), 348-359, 2004

[2] X. Zhao, K. Chan, M. Li. Applying Agent Technology to Software Process Modeling and Process-Centered Software Engineering Environment. Proceedings the 2005 ACM Symposium on Applied Computing (SAC'05), 1529-1533.

[3] N.R. Jennings, P. Paratin, A. R. Lomuscio, S. Parsons, C. Sierra, M. Wooldridge. Automated Negotiation: Prospects, Methods and Challenges. Int. Journal of Group Decision and Negotiation, vol.10 (2), 199-215, 2001

[4] Wang Li-Chun, Chen Shi-Fu. A Negotiation Model for Multi-Agents and multi-Objectives. Journal of Software, vol.13(8), 1637-1643, 2002

[5] Vidya Narayanan, Nicholas R.Jennings. An Adaptive Bilateral Negotiation Model for E-Commerce Settings. In Proceedings of 7th International IEEE Conference on E-Commerce Technology, Munich, Germany: 34-39,2005

[6] Mokdong Chung, Honavar, V.A. Negotiation Model in Agent-Mediated Electronic Commerce. Multimedia Software Engineering. In Proceedings of International Symposium: 403 – 410, Dec. 2000.

[7] Kevin M.Lochner, Michael P.Wellman. Rule-Based Specification of Auction Mechanisms. In Proceedings of the Third International Joint Conference On Autonomous Agents And Multi-Agent Systems: 818-825, 2004.

[8] Stanley Y. W. Su, Chunbo Huang, Joachin Hammer. A Replicable Web-Based Negotiation Server For E-commerce. In Proceedings of the 33rd Hawaii International Conference on System Science, IEEE Computer Society, 2000.

[9] Claudio Bartolini, Chris Preist, Nicholas R.Jennings. Architecting For Reuse: A Software Framework for Automated Negotiation. In Proceedings of 3rd International Workshop on Agent-Oriented Software Engineering, Bologna, Italy: 87-98, 2002

[10] Foundation for Physical Agents. FIPA ACL Message Structure Specification, 2000.

A Formal Architectural Model For Mobile Service Systems

Zuohua Ding
Center of Math Computing and Software Engineering, Zhejiang Sci-Tech University
Hangzhou, Zhejiang 310018, P.R. China
and
National Institute for Systems Test and Productivity, University of South Florida
Tampa, FL 33620, U.S.A
zouhuading@hotmail.com

Abstract

This paper describes how agents migrate from one place to another place to collect services or offer services. The service architecture is based on two level model: the agent level and system level. Both agent and system behaviors are formalized by Labeled Coloured Petri nets. Possible agent migration cases have been discussed such as choice, sequential, and clone. Our architecture model reclaims the features for agent-based mobile service systems: the agent location and strong mobility.

Keywords: *Software architecture, mobile service, multi-agent system, formalism.*

1 Introduction

Service Oriented Architecture(SOA) is a new paradigm in distributed systems aiming at building lossely-coupled systems that are extendible, flexible, safer and fit well with existing legacy systems. Mobile agents are usually used to model mobile services because they can migrate from machine to machine, and execute at different locations during their life time. This paper combines these two paradigms together to model mobile service systems.

There are two levels in our model: agent level and system level. In agent level, each agent must have at least two processes: one for communicating with local resources and the other for self autonomy task to achieve service computing goal. Once the service computing is completed or more information is required, the agent is ready to migrate to other places. In system level, the system must have the ability to transmit agents from one location to another location, plus, the system needs to fit a sequence of activities such as suspending execution, transmitting agent, and resuming execution, when agents are in migration.

The rest of this paper is organized as follows. Section 2 is to create agent net and the agent behavior is described by Labeled CPN. In Section 3, we create system net based on Labeled CPN, which has agent nets as its tokens. We also discuss possible migration cases such as choice, sequential, and clone. In Section 4, we indicate that our model has two features: agent location and strong mobility, which are essential to mobile service systems. Section 5 reviews some related work, and Section 6 is the conclusion of this paper.

2 Creating Agents

In service oriented engineering, an agent must be capable of providing/requesting service to/from other agents or local hosts and reasoning about its own behavior towards new service design and computing, thus committing towards service accomplishment. Formally, we have

Definition 1.1 An agent is a software object with a knowledge base that contains non-empty sets of self service and public services, external communication primitives, internal service functions and service designing methods.

Here *service functions* include two aspects. One aspect is that an agent has ability to provide services when other agents make service requests. The agent may accept or refuse requests depending on the environment information occupied by the agent. The other aspect is that an agent requests services from other agents in case the knowledge is not rich enough for the agent to compute new selected services. The knowledge base may be updated after the agent accomplishes a service.

For the *service designing methods*, our argument is based on Belief-Desire-Intension(BDI) module [10] for agents.

The BDI module is based on the study of mental attitudes where an agent's mental attitudes are characterized by the notions of belief, desire, goals, and

intensions. Belief represents the information state of an agent which comprises of what the agent knows about itself and the world including all kind of services. Desire represents an agent's motivational state, i.e., what goal an agent is trying to achieve, in other words, what services an agent is planning. The Intension represents the deliberative state of the agent. In this state an agent choose a goal(service) and becomes committed to achieving it, i.e., service computing.

An agent's service planning may change over time because of the priorities(as specified by its user) or as a result of change when knowledge base gets updated. Thus the phase of service planning can be shifted to the phase of service computing when the agent believes that the planned service is achievable. If this is the case, the agent does not need any additional knowledge to achieve the goal only by assuming the existing knowledge it contains.

Therefore, in addition to external events, the internal service planning can also make an agent's state change in the process to accomplish the services.

Let $\mathcal{M} = \{Message, Service\}$ denote the data types from external events (message) and internal events (service), and S denotes all states including:

Knowledge-Updating(KU),
Service-Designing(SD),
Service-Computing(SC),
Message-Communicating(MC),
Event-Managing(EM).

Let T denote all those mappings that map one state to another state. If $a \wedge b$ means that conditions a and b must be true at the same time, then agent's state changes can be represented by the following mappings($t \in T$):

For Internal Events:

Mapping 1: From Knowledge-Updating to Service-Designing:

$t : s_1 \wedge m_1 \rightarrow s_2 \wedge m_2,\ s_1 \in KU, s_2 \in SD,$
$m_1, m_2 \in Service.$

Mapping 2: From Service-Designing to Service-Computing:

$t : s_1 \wedge m_1 \rightarrow s_2 \wedge m_2,\ s_1 \in SD, s_2 \in SC,$
$m_1, m_2 \in Service.$

Mapping 3: From Service-Computing to Knowledge-Updating:

$t : s_1 \wedge m_1 \rightarrow s_2 \wedge m_2,\ s_1 \in SC, s_2 \in KU,$
$m_1, m_2 \in Service.$

Outgoing Event Control:

Mapping 4: From Service-Computing to Event-Managing:

$t_p : s_1 \wedge m_1 \rightarrow s_2 \wedge m_2,\ s_1 \in SC, s_2 \in EM,$
$m_1, m_2 \in Service.$

Mapping 5: From Service-Designing to Event-Managing:

$t_r : s_1 \wedge m_1 \rightarrow s_2 \wedge m_2,\ s_1 \in SD, s_2 \in EM,$
$m_1, m_2 \in Service.$

Mapping 6: From Event-Managing to Message-Communication:

$t : s_1 \wedge m_1 \rightarrow s_2 \wedge m_2,\ s_1 \in EM, s_2 \in MC,$
$m_1 \in Service, m_2 \in Message.$

Incoming Event Control:

Mapping 7: From Message-Communicating to Event-Managing:

$t : s_1 \wedge m_1 \rightarrow s_2 \wedge m_2,\ s_1 \in MC, s_2 \in EM,$
$m_1 \in Message, m_2 \in Service.$

Mapping 8: From Event-Managing to Service-Computing:

$t : s_1 \wedge m_1 \rightarrow s_2 \wedge m_2,\ s_1 \in EM, s_2 \in SC,$
$m_1, m_2 \in Service.$

Mapping 9: From Event-Managing to Knowledge-Updating:

$t : s_1 \wedge m_1 \rightarrow s_2 \wedge m_2,\ s_1 \in EM, s_2 \in KB,$
$m_1, m_2 \in Service.$

Mapping 10: If a state has silent moving, we may use ϵ to denote the state mapping.

Naturally, we may use a Coloured Petri net to describe all the state changes of agents. There are a few definitions about Coloured Petri nets(CPN). For our purpose, we choose the definition in [3]. With a little modification, we get the following Labeled CPN which is also an extension of the Labeled Petri net [7].

Definition 1.2 A bag (or multiset) 'bg', over a non-empty set A, is a function $bg : A \rightarrow N$, sometimes denoted as a formal sum $\sum_{a \in A} bg(a)'a$. Bag(A) denotes the set of all bags over A. Extending set operations to Bag(A) we define the $sum(+)$ and the $difference(-)$ as follows:

If bg, bg_1 and bg_2 are bags over A, then:

- $bg_1 + bg_2 := \sum_{a \in A}(bg_1(a) + bg_2(a))'a$
- $bg_1 \leq bg_2 :\Leftrightarrow \forall a \in A, bg_1(a) \leq bg_2(a)$
- $bg_1 - bg_2 := \sum_{a \in A}(bg_1(a) - bg_2(a))'a$ if $bg_2 \leq bg_1$ and
- $|bg| := \sum_{a \in A} bg(a)$ is the size of bg and $\emptyset$ denotes the empty bag (with $|bg| = 0$).

Definition 1.3 A Labeled CPN is defined by a tuple

$$\mathcal{N} =< P, T, \mathcal{C}, cd, A, M_0 >$$

where

- P is a finite set of places,
- T is a set of transition relations, i.e., $\forall t \in T, t \subseteq 2^P \times 2^P$,
- $\mathcal{C}$ is the set of colour classes,
- $cd : P \cup T \rightarrow \mathcal{C}$ is the colour domain mapping,
- A is a set of action label functions on transition relation T, and $A : t \rightarrow Bag(cd(t)), \forall t \in T$,
- m_0 is the initial or current marking:

$$M_0 = \cup_{p \in P} M_0(p),$$

where $M_0(p) \in Bag(cd(p))$.

Definition 1.4 A marking of a Labeled CPN

$$\mathcal{N} =< P, T, \mathcal{C}, cd, A, M_0 >$$

is a vector M such that $M(p) \in Bag(cd(p))$ for each $p \in P$. A transition relation $(p, t, q),\ p, q \in P,\ t \in T$ can "fire" in a marking M iff $M(p) \geq a(t),\ a \in A$ (denoted by: $M \xrightarrow{a}$). The firing of a transition leads to the next marking:

$$M'(p) = M(p) - a(t),\ \text{if } p \in (p, t, q),$$

$$M'(p) = M(p) + a(t),\ \text{if } p \in (q, t, p,)$$

$$M'(p) = M(p),\ \text{otherwise.}$$

Such a state transition is denoted as (M, a, M'). Given a Labeled CPN $\mathcal{N}$, the reachability graph of $\mathcal{N}$, denoted as $RG(\mathcal{N})$, is the (reflexive)transitive closure of the above next-state relation. The nodes of the reachability graph represents the reachable state space of the net, whereas the edges (M, M') are labeled with the action a of the transition (p, t, q) which must be executed in M to reach M'.

As [13], we may modify Labeled CPN to define Agent net. We assume that a Labeled CPN, which represents the behavior of an agent, contains one input place (i.e., a place with no incoming transition) and one output place(i.e., a place with no outgoing transition). The input place is used for accepting messages while the output place is used for sending message out.

Definition 1.5 (Agent net). Agent net AN for agent A is a tuple

$$AN = (P, T, \mathcal{C}, cd, A, M_0, p_{in}, p_{out}),$$

where

$$(P, T, \mathcal{C}, cd, A, M_0)$$

is a labeled CPN, $p_{in}(p_{in} \in P)$ is the input place for incoming messages, and $p_{out}(p_{out} \in P)$ is the output place for outgoing messages.

Remark 1.1 In Ref. [13], the definition of agent net is based on a simplified PrT net, while in Ref. [4], the service net is based on a labeled Place/Transition net.

Remark 1.2 Here in our definition, the coloured tokens only represent service data and message data, not a whole service. Ref. [12] mentioned that a service is simply a single and coherent block of activities, thus an agent has to collaborate all actions to perform service processing.

When moving from the definition of Agent to the definition of Agent net, the above 10 state mappings will be transformed to transitions and action labels that are defined on the transitions. Two action labels will be most important to us since they play a key role in the discussion of Section 4. These are the labels corresponding to mapping 1 from Service-Designing to Event-Managing and mapping 2 from Service-Computing to Event-Managing. We use a_{sr} and a_{sp} to denote them, respectively.

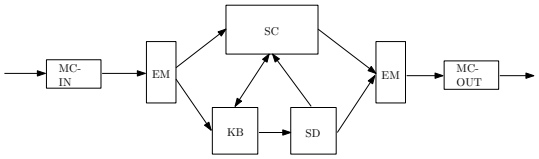


Figure 1. Agent Architecture

Now we give a suggestion to design an agent. An agent may consists of the following modules: Message Communication module(MC), Event Management module(EM), Knowledge Base module(KB), Service Designing module(SD), and Service Computing module(SC)(see Figure 1).

Message Communication: It contains two parts, one is to handle incoming messages, and the other handles outgoing messages. The module is augmented with two message queues: a message-in-queue where the inward messages are temporarily queued up before they can be processed and a message-out-queue where the outward message are queued up before they can be dispatched.

Knowledge Base: It is the information repository of an agent. The knowledge base contains several items, knowledge of own service capabilities, service capabilities of other agents in the society, and the state description in terms of mental attitudes.

Event Management: It has two parts. The first part is to store the offered services from other agents to knowledge base or to send the service requests to Service Computing module; The second part is responsible to send out service offering and service requesting. This module must have the ability to send service requesting out again if it can not receive the required service back in some time period.

Service Design: It can design some new services following the given rules. If the current knowledge contained in the Knowledge Base is not enough to create a new service, it will send a service request to message-out-queue.

Computing Service: It either retrieves services from the Knowledge Base and sends the services out, or creates new services based on current knowledge and rules, thereafter sends the new services out.

3 Mobile Service Of Architecture

In the architecture to be built, agents collect/offer services from/to different locations. If the user task is accomplished by a single agent, then this agent sequentially visits each host where the required resources/services are located. On each host, the agent carries out some computations(or assumes some services) with the local resources, and then moves to the next host to continue its work. If a team of mobile agents cooperate to accomplish a task, then these

agents go to different hosts involved in the computation, compute with local resources and communicate with each other [9].

All the local resource hosts will form an environment which has ability to make agents migrate from one place to another place, meanwhile supports activities such as transmitting agent, issuing a request, suspending agent execution, and resuming execution. Thus, environment needs to be considered in the mobile system design [1].

Following component-based [2] design skill, the architecture model of the system can be described by a Coloured Petri net, which has agents as its tokens. Since our agents are also Coloured Petri net, the whole system can be represented by a Coloured Petri net that has other Coloured Petri nets as its tokens. To distinguish the two-level Petri nets, the base Coloured Petri nets are called System nets(upper level), while the tokens in net form are called Token nets(lower level). These kind of models have be described before such as Coloured Petri net(CPN) [6], objected Petri net(OPN)Valk [11], and multi-agent nets, PN^2 [5].

Here is the idea to construct our model: each token changes its state by occurrences of a transition. A token is available for a transition in the upper-level net if there is at least a service action(requiring/providing) happened in the corresponding lower-level net.

Let AN be an agent net and $\mathcal{A}$ be a finite set of actions, and $AN_{\mathcal{A}}$ denotes the set of all agent nets whose action set is a subset of $\mathcal{A}$, i.e.,

$AN_{\mathcal{A}}$
$= \{AN = (P, T, \mathcal{C}, cd, A, M_0, p_{in}, p_{out}) | A \subseteq \mathcal{A}\}$.

Let $AN^{in}_{\mathcal{A}} (\subset AN_{\mathcal{A}})$ denote the set of agents that has marking only at place p_{in}, and $AN^{out}_{\mathcal{A}} (\subset AN_{\mathcal{A}})$ denote the set of agents that has marking only at place p_{out}.

Definition 2.1 (System net). A System net is a Labeled Coloured Petri net (CPN) defined by a tuple

$$\mathcal{N} =< P, T, \mathcal{C}, cd, A, s, s_0 >,$$

where

- P is a finite set of places,
- T is a set of transition relations, i.e., $\forall t \in T, t \subseteq 2^P \times 2^P$,
- $\mathcal{C}$ is the set of colour classes,
- $cd : T \to \mathcal{C}$ is the colour domain mapping,
- A is a set of action label functions on transition relation T, and $A : (t, cd(t)) \to Bag(AN^{in}_{\mathcal{A}}), \forall t \in T$,
- $s : P \to Bag(AN_{\mathcal{A}})$ is a configuration mapping and s_0 is the initial configuration:

  $$s_0 = \cup_{p \in P} s_0(p),$$

  where $s_0(p) \in Bag(AN_{\mathcal{A}}))$.

Let ${}^\bullet t = \{p|(p, t, q)\}$ and $t^\bullet = \{q|(p, t, q)\}$. Denote

$$s({}^\bullet t) = \{s(p)|(p, t, q)\}.$$

Definition 2.2 A transition t of $\mathcal{N}$ is enabled if for $\forall s(p) \in s({}^\bullet t)$, there exists an agent $AN \in s(p)$ such that it has either service requesting action or service providing action happened. t can be fired if it is enabled and there is an agent $AN \in AN^{out}_{\mathcal{A}} \cap s(p)$.

The firing of a transition leads to a new configuration of the system net. Let s be the current configuration and s' be the new configuration being transferred. Since the configurations are the marked agent nets, if we use $\delta : (AN, M) \xrightarrow{a} (AN, M')$ to represent an agent AN changing configuration from marking M to M' under action a, then the configuration change for a set of agents $\cup_{i=1}^{n} AN_i$ at the same time by actions $a_1, a_2, \ldots, a_n$ would be:

$$\delta : (\cup AN_i, M_i) \xrightarrow{\cup a_i} (\cup AN_i, M'_i).$$

Thus, we have configuration transfer equations as

$$s'(p) = \delta(s(p)) - \cup AN^{out}_{\mathcal{A}},\ \delta^{-1}(AN^{out}_{\mathcal{A}}) \in s(p),$$

$$\text{if } p \in (p, t, q);$$

$$s'(p) = \delta(s(p)) + a(cd(t)),\ \text{if } p \in (q, t, p), a \in A;$$

$$s'(p) = s(p),\ \text{otherwise.}$$

To give an explicit expression, for each agent AN, we use AN^i to represent the current state and use AN^{i+1} to represent the next state. For each place of the system net, since $s : P \to Bag(AN_{\mathcal{A}})$ and $a(cd(t)) \in Bag(AN^{in}_{\mathcal{A}})$ there exists a $bg_p \in Bag(AN_{\mathcal{A}})$ and a $bg_t \in Bag(AN^{in}_{\mathcal{A}})$, such that

$$s(p) = bg_p = \sum_{AN^i \in AN_{\mathcal{A}}} bg_p(AN^i)' AN^i,$$

$$a(cd(t)) = bg_t = \sum_{AN^{in} \in AN^{in}_{\mathcal{A}}} bg_t(AN^{in})' AN^{in},$$

then the above configuration transfer equations can be rewritten as

$$s(p) = \sum_{AN^{i+1} \in AN_{\mathcal{A}}} bg_p(AN^{i+1})' AN^{i+1},$$

$$- \sum_{AN^{out} \in AN^{out}_{\mathcal{A}}} bg_p(AN^{out})' AN^{out},$$

$$+ \sum_{AN^{in} \in AN^{in}_{\mathcal{A}}} bg_t(AN^{in})' AN^{in},$$

$$(p \in (p, \cdot, \cdot), t \in (\cdot, t, p).$$

Next we consider possible migration cases for agents. Here is a mobile service model we can build: at very beginning, an agent sits in a local place with an input from this place, which is the initial marking to

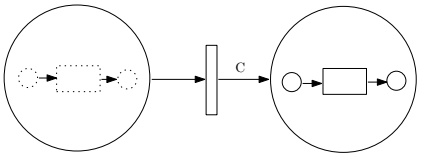


Figure 2. Sequential migration

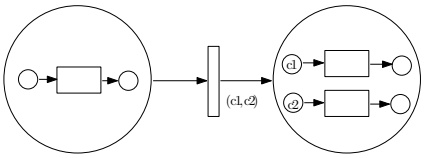


Figure 3. Sequential migration with copying

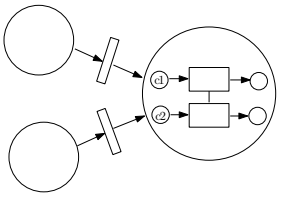


Figure 4. Multi-agent Migration

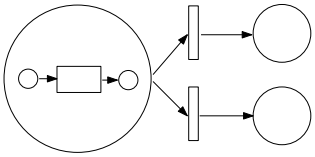


Figure 5. Choice Migration

the agent. Two cases will be involved in the agent execution. In the first case, the agent has enough resource, then it will finish service computing without any new other information, and a service providing action will be performed. In this case, this agent offers a service to the local place. In the second case, the agent does not have enough resource to finish the service, then it needs some additional information from other places, and the agent will issue a service requesting action. In this case, the agent is collecting services. In both cases, the agent will move to other places. We call this moving *Sequential Migration*(see Figure 2).

When arriving to the new places, the agent will get input information from the new place. In Figure 2, the input is c. In the first case, this agent starts a new service, and in the second case, the agent continues the previous service with the new information received from the new place.

After an agent moves from one place to another place, several agent copies may be generated depending on the input message types(colours) attached to the transition. This moving is called *Sequential Migration With Copying*.

For example, in Figure 3, two message types c_1 and c_2 are attached to the transition which generates inputs to two agent copies from the same agent. In this case, both copies are performing service computing for the local host.

One place may receive several agents from different places. The moving is called *Multi-agent Migration*(Figure 4). There are three cases here: 1) all agents are collecting service at the same time, 2) all agents are performing service computing for local host; 3) the mixture of 1) and 2). In all the cases, the agents will experience a resource sharing problem. Hiding actions will be used to assist the service accomplishment.

An agent may also has a chance to make a choice where to go. Generally speaking, it is nondeterministic unless some priority is assigned to the transition. This moving is called *Choice Migration*. Figure 5 is a description.

The last but not the least is *Clone Migration*. An agent may move to different places at the same time, and if this situation occurs, the agent migrates with its clones. For an example, see Figure 6.

4 Mobility Analysis

Our architecture module shows some typical features for mobile service systems.

Agent Location. As [13], we can prove that it is impossible for an agent to be located at more than one place when it is in service. This is a basic requirement for mobile agent system. In service we mean that this agent is doing service computing or service planning based on current resources, in both cases, the agent has to stay in the same place.

Strong Mobility. An agent is active and its state is preserved during migration. We have two cases here. In the first case, an agent needs additional information to complete its service computing. Before migration, the agent state is SD. After migration, since the new information is joining, the state will visit EM, KB and SD sequentially, which means the state can be resumed before the agent continues to execute. In the second case, an agent finishes a service, and sends message out through p_{out} before migration. Because the agent

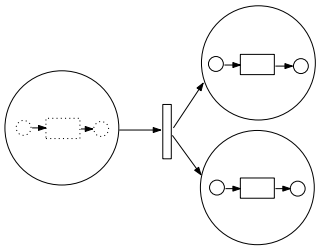


Figure 6. Clone Migration

has only one output and one input place, the only event that will happen is that an input will come to p_{in} from local host if the agent is still in the same place. However, in our system, this agent has to move to a new place after accomplishing service, and gets an input to p_{in} from the new place. In other words, agent's state is preserved after migration.

5 Related Work

The work presented in this paper is heavily dependent on the idea "nets in nets", which is usually used to describe the natural hierarchies in an agent system. This idea has also been employed by other systems such as in [8] and [13].

In [8], a multi agent system architecture MULAN is described. The places of the system contain agent platform as tokens. The transitions describe communication or mobility channels, which build up the infrastructure. Each agent platform offers services to the agent. Agents can communicate by message exchange. Two agents of the same platform can communicate by the transition internal communication, which binds two agents, the sender and the receiver, to pass one message over a synchronous channel. External communication only binds one agent, since the other agent is bound on a second platform somewhere else in the agent system. Agents can leave the platform via the transition *send agent* or enter the platform via the transition *receive agent*.

In [13], The authors model a mobile agent system as a two-layer system based on predicate/transition(PrT) nets. They use a systm net, agent nets, and a connector net to model the environment, agents and the connector, respectively. Agent nets are packed up as parts of tokens in system nets, so that agent transfer and location change are naturally captured by transition firing in Petri nets. Agent nets themselves are active only at specific places and disabled at all the other places in a system net. To do checking, they mapped two-layer model to PrT net.

6 Conclusion

Based on the characteristics of agents, we have studied the service oriented architecture, especially we considered the mobile service of architecture. The new architecture is based on two level model: the agent level and system level. Both agent and system are formalized by Labeled Coloured Petri nets and the system net has agent nets as its tokens. The ways how agents migrate from one place to another place have been described too. Our architecture model has the basic features for agent-based mobile service system: the agent location and strong mobility.

References

[1] F. Fuggetta, G. Picco, and G. Vigna, Understanding code mobility, *IEEE Transactions of Software Engineering*, 24(5):342-361,1998.

[2] D. Garlan, Formal modeling and analysis of software architecture: components, connectors, and events, *LNCS 2804*, 2003, 1-24.

[3] C. Girault and R. Valk, *Petri nets for systems engineering, a guide to modeling, verification and applications*. Springer-Verlag, Berlin, 2003.

[4] R. Hamadi and B. Benattallah, A Petri net-based model for web service composition, *Proceedings of 14th Australasian Database Conference*, Adelaide, Australia, V.17, 2003.

[5] K. Hiraishi, A Petri-net-based model for the mathematical analysis of multi-agent system, pp.3009-3014,2000.

[6] K. Jensen, Coloured Petri nets: basic concepts, analysis methods and practical use, Vol. I,II,III, Springer-Verlag(1992,1995,1997).

[7] G. G. de Jong and Bill. Lin, A communicating Petri net model for the design of concurrent asynchronous modules, In*Proceedings of the 31th ACM/IEEE Design Automation Conference*, June, 1994.

[8] M. Kohler, D. Moldt and H. Rolke, Modeling mobility and mobile agents using nets within nets. *LNCS*2679,pp.121-139, 2003.

[9] X. Ma, J. Lu, X. Tao, Y.Li and H. Hu, A mobile-agent-based approach to software coordination in the HOOPE system, *Science In China (Series F)*, 45(3):203-219, 2002.

[10] A. S. Rao and M. P. Georgeff, Modeling rational agents within a BDI architecture, *Proceedings of The Second Conference On Knowledge Representation and Reasoning*, Morgan Kaufman, pp:473-484, 1991.

[11] R. Valk, Concurrency in communicating object Petri nets. *Concurrent Object-Oriebted Programming and Petri Nets*, LNCS,pp.164-195, 2001.

[12] M. Wooldridge, N. Jennings and D. Kinny, The gaia methodology for agent-oriented analysis and design, *Proceedings of 3rd International Conference on Autonomous Agents*, Seatle, WA, 1999.

[13] D. Xu, J. Yin, Y. Deng and J. Ding, A formal architecture model for logial agent mobility, *IEEE Transaction On Software Engineering*, 29(1):31-45, 2003.

An Environment of Knowledge Discovery in Database

Maria Madalena Dias[1], Roberto Carlos dos Santos Pacheco[2], and Lúcio Gerônimo Valentin[1]

[1] *Departamento de Informática, Universidade Estadual de Maringá,*
Avenida Colombo 5790,
870200-900 Maringá, Paraná, Brasil
{mmdias, lgvalent}@din.uem.br
http://www.din.uem.br

[2] *Programa de Pós-Graduação em Engenharia e Gestão do Conhecimento,*
Universidade Federal de Santa Catarina, Caixa Postal 476
88049-000 Florianópolis, Santa Catarina, Brasil
pacheco@stela.ufsc.br
http://www.stela.ufsc.br/~pacheco

Abstract

After having solved its operational problems, usually emerges in an organization the need for systems able to provide support to the decision-making. Data mining is an area that is growing quickly to assist such new needs of organizations. However, the use of data mining techniques is uncommon due to the difficulty, normally found, in the development of systems related to knowledge discovery. This paper presents an environment of knowledge discovery in database, named ADesC. The main objective is to generate relevant information to decision-making, with the application of data mining techniques. This environment is based on agent technology to aid the accomplishment of its tasks.

Keywords. *Knowledge discovery in database, data mining, agent technology, UML.*

1. Introduction

In spite of the existence of great database with much information about company business, some difficulties are found in the knowledge discovery that is based on such information. These difficulties can be related to the complexity of the knowledge discovery in database (KDD) systems, to the lack of knowledge about data mining techniques, or to the lack of adequate tools.

The available tools in the market frequently assist specific types of applications. Thus, the organization can need to acquire multiple tools to assist its current and future needs.

The design and implementation of systems of knowledge discovery can be constituted of complex tasks, as well as, can be highly dependent on the development team. Its implementation demands the knowledge of advanced technologies of the computer science area, such as: data mining techniques, database management systems, distributed systems, artificial intelligence, etc. The system analyst also needs to have enough knowledge concerning the organization business area. Thus, it is necessary to construct an environment to provide support to the development of those types of systems.

In order to supply such a need, it was specified an environment of knowledge discovery regarding database, named ADesC, that is presented in this paper. This environment is based on the agent technology; it applies data mining techniques and it uses UML (Unified Model Language) diagrams to represent its characteristics.

At the moment, a prototype of the ADesC was implemented and its architecture and the detailed design are undergoing the construction phase.

The technology of agents has been recognized as sufficiently efficient for the development and management of systems of discovery of knowledge in database, mainly when it is treated to extract knowledge from great databases located in diverse nodes of computer networks.

"An agent can be defined as, something or someone that acts by representing another part, with the aim of developing specific functions in benefit of the represented part" [2].

Data mining is a part and parcel of the Knowledge Discovery, in the Databases (KDD) process. Goebel and Gruenwald (Goebel & Gruenwald, 1999) state that the term KDD represents the process by which low-level data are transformed into high-level knowledge. On the other

hand, data mining is defined as the extraction of patterns or models from any reported data.

"Data mining is the exploration and the analysis of a great amount of data, by using automatic or semi-automatic procedures to discover significant patterns and rules" [1].

In the next section of this paper the steps of the KDD process are introduced and some tools of knowledge discovery are described. In section 3 the components of the model of ADesC environment are described, the design of the environment is discussed and the prototype implementation environment is presented. In section 4 the results of two case studies are shown. Finally, in section 5, some conclusions are reported.

2. The KDD Process

The process of knowledge discovery is a complex, semi-automatic and interactive method [3]. The steps of this process, based on [4], and [5], and synthesized in [6], are shown in Figure 1.

2.1. Knowledge Discovery Tools

In agreement with [7], many available tools are generic tools of Artificial Intelligence or of the statistics community. Such tools generally operate separately from the data source, thus, requesting a significant amount of time with data export, import, and transformation. However, according to the authors, the rigid connection between the tool of knowledge discovery and the database analyzed is clearly desirable.

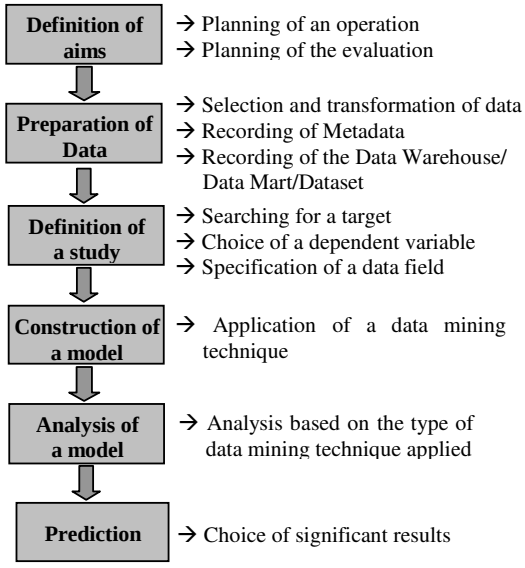


Figure 1. Knowledge Discovery Process

According to Goebel and Gruenwald [7], the characteristics considered in the choice of a tool to provide knowledge discovery should be the following ones:

- Ability to access a variety data sources, on-line and off-line;
- Capacity to include models of data guided to objects or non-standardized data models, such as multimedia, spatial or temporal;
- Processing capacity in relation to the maximum number of tables/rows/attributes;
- Processing capacity in relation to the size of the database;
- Variety of types of attributes to be manipulated with the tool; and
- Query languages used.

Following, some knowledge discovery tools are presented: AIRA [8]; Alice [8]; Clementine [8]; KnowledgeSEEKER [4]; MineSet [9]; DataMind [4]; Intelligent Miner [8]; NeuralWorks Predict [4]; PolyAnalyst [8]; WEKA [10].

Many of these tools solve specific problems of data mining, but they do not consider all the characteristics presented by Goebel and Gruenwald [7].

3. ADesC Environment

The ADesC environment has the following main aims: to help the system analyst to discover knowledge in the database, thus offering facilities in the construction of those systems; to translate UML models into E-LOTOS (Enhancements to Language of Temporal Ordering Specification) and, to offer the final user, an interactive environment that creates important information in relation to the organization decision. Such information is reached by the environment through the application of data mining techniques.

Thus, the ADesC environment will allow the use of diverse data mining techniques. The user will be able to choose the most adequate technique to solve the problem of knowledge discovery. Moreover, besides those characteristics, the ADesC environment makes possible the access to a variety of data sources and supports the construction of integrated data marts. The information contained in the data marts could be directly accessed by the user, or even be used in the search for interesting knowledge through resources offered by the environment.

Resources for graphical representation of the discovered knowledge will also be offered, thus, facilitating the analysis of the results.

3.1. ADesC Model

The main ADesC environment components are: Interfaces Manager, Support, Agent Structure, Translator, Manager User, Analyst User, Final User, Database (DB)

and/or Data Warehouse (DW), Dataset (DS), Metadata and Results. Figure 2 shows the ADesC environment model.

The Manager User, the Analyst User and the Final User present the kinds of users who interact with the environment through the Interface Manager.

The attributions of the Interface Manager are to create and to control, all the objects of the environment interface, and also, the ones used to visualize the results. The Support is responsible for the database identification, regarding what will be researched (operational database or data warehouse); it is also responsible for the recording and access of metadata, for the recording and access of the dataset and, for the recording and access of the application results of data mining techniques, as well.

UML models are translated into the E-LOTOS formal specific language by using the Translator. The objective is to make possible the verification and validation of the specification of a knowledge discovery system.

The DB and DW represent the operational and multidimensional data used as a source of input to KDD systems.

A Dataset is used to represent a specific data set.

The metadata keeps information on the selected data and its transformations.

The discovered knowledge is registered in the database under the title "Results".

The Agent Structure is responsible for the management of the agents that form the KDD. Such structure is compounded by: Agent Server, Service Manager, Transport Coordinator and by the agents.

The Agent Server is responsible for the control of the agent life span.

The responsibility of the Service Manager is to identify the types of service and to create the necessary conditions for the execution of such services.

The Transport Coordinator coordinates the activities and the necessary services for the transportation of agents from one knot to another one on the net.

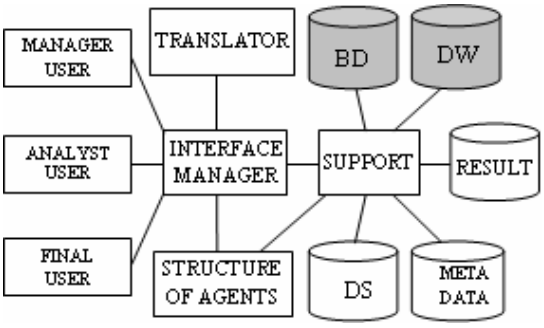


Figure 2. ADesC model

The ADesC agent types are described bellow.

- Search agent: it is the agent, mobile or not, responsible for the search and selection of data in the distributed or local database.
- Transformation agent: it is the agent responsible for the transformation of the data that will create the DW or DS in order to be used in the application of the data mining technique.
- Router agent: it is the agent responsible for the definition of the route, which a search mobile agent should proceed to arrive at the destination knot on the net.
- Technique agent: it is the agent that implements a data mining technique. For each one of the data mining techniques, an agent of that type should be implemented.
- Analysis agent: it is the agent that aids the user when making the analysis of the results, thus, filtering the most interesting knowledge, and, showing the results, by using adequate techniques of visualization in the data mining technique applied.
- Update agent: it is the agent which is responsible for the updating of DW or DS.

The agents are projected as objects. Besides the methods that implement the specific functions established, each agent has the following characteristics: creation, reproduction, expedition, inactivation, activation and removal.

3.2. ADesC Design

The informal design of the ADesC is based on the agent technology used for the definition of the necessary classes of objects, and, it uses UML diagrams for the representation of both, the objects and its behaviors.

Some UML diagrams were constructed to represent the ADesC environment characteristics. Figure 3 shows the Use Case Diagram.

The Use Case Diagram represents the interaction between the system and its users. The ADesC environment foresees three types of users, they are: the manager user, the analyst user and the final user. The cases of use represent the main activities foreseen in the ADesC environment.

The manager user has access to all the functions of the system and it is responsible for the configuration of the system; that is to say, inclusion of users and definition of the types of access provided to each user. The other users have more restricted access.

Object classes were defined to assist the needs for representation, emerging from the components of the model, illustrated in Figure 2. Figure 4 shows the class diagram of the ADesC environment.

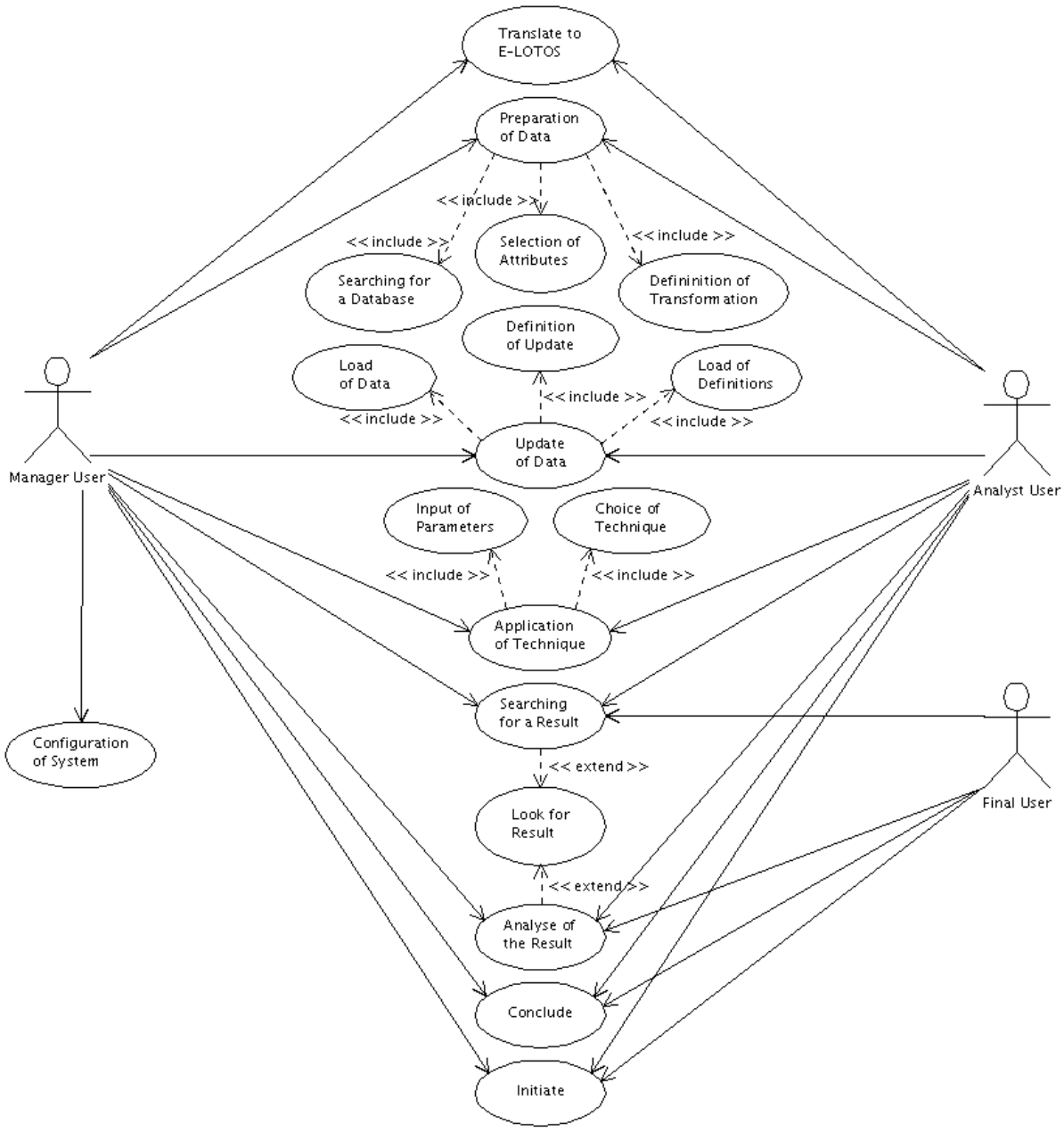


Figura 3. Use Case Diagram

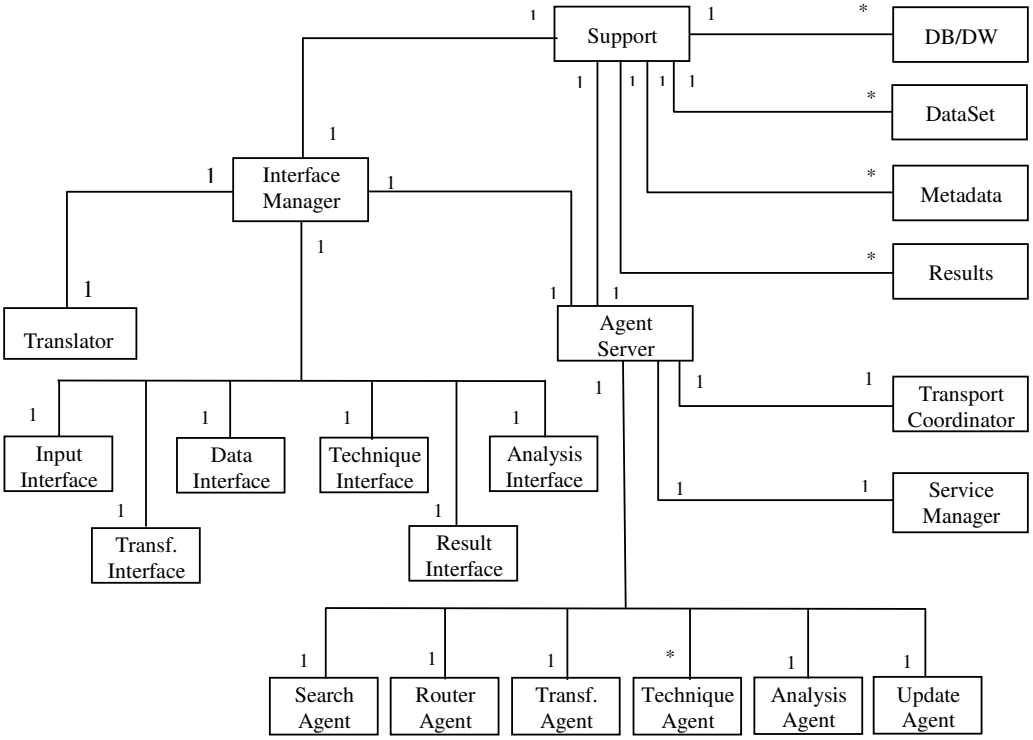


Figure 4. ADesC Class Diagram

3.3. ADesC Prototype

Borland Delphi used the implementation environment due to the easiness that such an environment offers to the development based on objects, besides having the BDE (Borland Database Engine) component.

The BDE component makes the access to several databases possible, such databases are: Oracle client/server, MSSQL Server, DB2, Informix, Interbase, Sybase, Access, Paradox and pattern of files DBF, very common in legacy systems as the ones developed in clipper. Besides its native drivers, BDE allows the installation of new drivers.

4. Case Study

The case study had as main objective the establishment of the knowledge discovery system construction, by using the prototype of the ADesC Environment. In such studies the data mining technique - Discovery of Association Rules - was applied, based on the data obtained from the databases of Brazilian Masters Programs Degree in 1998.

The first study, besides verifying the relation between Fomentation and Productivity, has also analyzed if there is any relationship between the amount of scholarships granted to the Master Program and the Productivity of researches. Table 1 presents the rules generated in the study.

It can be observed that the relative raising in the amount of scholarships of one Program implies in an increase in Production. However, there is a level of saturation in the rule, and that emerges, when the average time of scholarships granted students is between 25 and 30 months. It is interesting to observe that, this fact is connected to the policies of reduction regarding 'the maximum time of scholarships' adopted by CAPES (Coordination of Perfection of Staff of Superior Level).

The second study has analyzed the relationship between the student's formation time either, with the Fomentation availability in the Program (being granted a scholarship), or with the student's participation in projects. The objective was to verify if there is any relationship between the time of the student's formation, and the fact of being (or not) granted a scholarship, and also, to verify the effect of that on his project dissertation. Table 2 shows the rules obtained in that study.

The study has allowed observing the existence of a raising in the medium formation time, when the student possesses scholarship or when his dissertation is related to the research project.

Regarding the concession or granting of scholarship, it was observed a larger impact in the connection or linkage with the project than in the medium formation time.

Table 1. Rules Generated to the First Study

Rules	Support	Confidence
If the financial support of the program is from 19 to 24 months per graduating student, then each researcher will produce from 2 to 3 publications	19,62%	53,76%
If the financial support of the program is approximately from 25 to 30 months per graduating student, then each researcher will produce from 2 to 3 publications	16,88%	48,19%

Table 2. Rules Generated to the Second Study

Rules	Support	Confidence
If the student is granted a scholarship, then, his thesis is linked to the research project	36,88%	53,27%
If the student is granted a scholarship, then his formation time takes more than 36 months	28,34%	40,94%
If the thesis is linked to a Project, then the student's formation time takes more than 36 months	21,30%	42,72%
If the student is granted a scholarship and his thesis is linked to a Project, then his formation time takes more than 36 months	14,11%	38,26%
If the area is from the Earth and Exact Science, then the student is granted a scholarship.	10,16%	82,66%
If the area is from Human Science, then the student is granted a scholarship	13,06%	72,70%
If the area is Engineering, then the student is granted a scholarship	11,44%	65,78%
If the area is Engineering, then the thesis is linked to a project	10,04%	57,70%

In addition, it was observed that, Programs from areas related to Exact Science, Earth studies, Human Sciences and Engineering are the ones that possess the largest number of scholarships granted. Moreover, it was verified that more than half of the Engineering area dissertations possesses a vinculum or connection with the project.

5. Conclusions

The specification of an implementation environment of knowledge discovery systems constitutes a complex task due to the nature and to the heterogeneous aspect of those systems and, also, due to the level of necessary details.

The informal specification of the ADesC environment was accomplished based on the agent technology, for being a technology type that is becoming currently used in the solution of complex computational problems.

The use of UML models in the stage of informal design facilitates the understanding of the system by using graphic representation of its characteristics.

The use of E-LOTOS in the specification of systems facilitates its verification and its validation, turning it into the most reliable and better qualified system.

With the use of the ADesC environment, the models UML, that represent the characteristics of the knowledge discovery system designed, can be translated automatically into E-LOTOS being incorporated to the environment, thus, completing the whole implementation of the system.

After the execution of the data mining algorithm chosen, ADesC will return the results of the accomplished data mining and will help the final user in the analysis of the results, by using the implementation of intelligent agents.

The prototype was successfully used for the search of relationships among data related to science and technology in Brazil. Consequently, it is also possible to develop other types of application.

After the prototype validation, the ADesC architecture and detailed design are being constructed, therefore, new results will be presented in the future.

References

[1] M.J.A. Linoff, and G. Linoff, *Data Mining Techniques*, 1st edn. John Wiley & Sons, Inc., United States of America, 1997.

[2] K. Heilmann, et al., "Intelligent Agents: A Technology and Business Application Analysis", Retrieved from *http://www.haas.berkeley.edu/~heilmann/agents/index.html*, November, 1997.

[3] H. Mannila, "Data Mining: Machine Learning, Statistics, and Databases", In: *Eight International Conference on Scientific and Database Management*, 1996, pp. 1-8.

[4] R. Groth, *Data Mining: A Hands on Approach for Business Professionals*, Prentice Hall, Inc., Upper Saddle River, New Jersey, United States of America, 1998.

[5] R. V. Lans, "O que é Data Mining e uma Análise do Mercado de Produtos", In: *8º Congresso Nacional de Novas Tecnologias e Aplicações em Banco de Dados*, 1997.

[6] M. M. Dias, "Um Modelo de Formalização do Processo de Desenvolvimento de Sistemas de Descoberta de Conhecimento em Banco de Dados", *Tese de Doutorado*, Universidade Federal de Santa Catarina, Florianópolis, Santa Catarina, Brasil, 2001.

[7] M. Goebel, and L. Gruenwald, "A Survey of Data Mining and Knowledge Discovery Software Tools", *SIGKDD Explorations*, Vol. 1, Nº. 1, 1999, pp. 20-33.

[8] M. Mendonça, and N. L. Sunderhaft, "Mining Software Engineering Data: A Survey", Retrieved from *http://www.dacs.dtic.mil/techs/datamining/descriptions.shtml*, October, 2005.

[9] C. Brunk, J. Kelly, and R. Kohavi, "MineSet: An Integrated System for Data Mining", In: *Third International Conference on Knowledge Discovery and Data Mining*, AAAI Press, Newport Beach, United States of America, 1997, pp. 135-138, Retrieved from *http://citeseer.ist.psu.edu/brunk97mineset.html*, October, 2005.

[10] Weka Data Mining System, "Weka Experiment Environment", Retrieved from *http://www.cs.waikato.ac.nz/ml/weka* , October, 2005.

Genre-based approach to Requirements Elicitation

Aneesh Krishna, Rodney J. Clarke*, Aditya K. Ghose
Decision Systems Laboratory, School of I.T and Computer Science
(*School of Economics and Information Systems)
University of Wollongong, NSW 2522, Australia
{aneesh, rodney_clarke, aditya}@uow.edu.au

Abstract

All social systems- material and virtual, persistent and transient- are constituted in and by the effects and processes associated with all forms of meaning. As a consequence, semiotic approaches are appropriated for studying organisations, work, and technologies in general. This paper demonstrates how Systemic Semiotics, an approach that combines a semiotic model of language called Systemic Functional Linguistics with selected concepts from Social Semiotics, can be applied to create Agent-based Systems in which social processes can be elicited from stakeholders, specified by designers, and embedded into actual agent-based systems. The utility of Systemic Semiotics applied to Agent-Oriented Conceptual modelling (exemplified by the i framework) is demonstrated to address the problem of registering and training volunteers in an emergency services organization.*

1. Introduction

It is reasonable to suggest that understanding the semiotic nature of social systems in terms of communication and action might assist in improved computer-based representations in the form of technical artefacts. Just as systems in organisations can be theorised as texts in context, so to can any systems development practices. In fact exactly the same techniques can be applied to the work practices of analysts, a property of semiotic and communicative approaches, referred to as metasymmetry [3]. This opens up the possibility of formally investigating requirements gathering and engineering processes in organisational contexts from a functional linguistic perspective. In an isolated study, Tebble [11] was able to characterise the underlying communication pattern (referred to as a genre structure described in detail in a latter section) associated with certain developer-user discussions. A consequence of that study is that all so-called social process methods [4] such as interviews, structured walkthroughs, and phase sign-offs, could be analysed using systemic semiotics methods. By extension, every systems development methodology consists of a set of formal methods and deliverables, and these also constitute texts. Recall that following Kress [9], a text is a 'completed act of communication in any medium'. A deliverable need not be language based to be a text in the sense defined here. It can be, as it often is, a flowchart, a state diagram or some other pictorial or iconic representation of an entity or process [9]. The very recognisability of methods for developers hints at there generic nature. Therefore, it is also reasonable to suggest that understanding the semiotic nature of social systems may contribute to a better understanding of the process by which technological artefacts are developed and deployed. We will explore the dual utility of a systemic semiotic approach to understanding system artefacts and development processes in this paper.

This paper represents an attempt to support the development of Agent oriented Conceptual Models for producing useful systems incorporating systemic semiotic theory and methods that emphasise the communicative, social and semiotic (meaning making) processes that occur in organisations. We proceed by first applying an Agent-oriented Conceptual Modelling framework called *i** [13], designed for use in early-phase requirements engineering, to the real world problem of registering and training volunteers in emergency services. The case study is used throughout this paper. The *i** framework consists of two main modelling components: the Strategic Dependency (SD) Model and the Strategic Rationale (SR) Model. These models are used to illustrate registration and training example (Figure 1) [12]. Readers are encouraged to read [13] for details on the *i** framework.

During the development of these models a number of practical difficulties arose as a consequence of the informal consultative processes employed during early-phase requirements elicitation sessions in our emergency services example. It became apparent that the underlying modelling notation could be used to drive the design of a set of Requirements Capture Templates (RCT) that could in turn simplify the process of requirements elicitation by structuring the stakeholder interaction [12]. Once the analyst/modeller completed these templates, the data contained within them could be manually transformed into the SD and SR models. These templates serve as a structured repository and a record of stakeholder interaction that can be revisited, re-negotiated or revised. These application-specific domain ontologies can help generate elicitation triggers and test the completeness and consistency of the conceptual models. These aspects are described as 'Ontology-driven template-based *i** model

elicitation'. From the perspective of systemic semiotics, the RCTs function as a set of compositionally related genres referred to as a macrogenre. This observation enables us to move existing systemic semiotic approaches to genre-based elicitation practices into the realm of agent oriented conceptual modelling. The recognition of the generic nature of elicitation necessitates an examination of the kind of language resources that are also in play during early-phase requirements engineering. These resources are introduced and described in the section entitled 'Systemic Semiotic foundation for Elicitation'. The importance of communication resources during requirements elicitation illustrates how systemic semiotic can be used to describe the process of systems development. Finally, section 4 presents some concluding remarks.

2. Ontology-driven Template-based *i** Model Elicitation

Early-phase RE activities have traditionally been done informally [13], beginning with stakeholder interviews and discussions on the existing systems and rationales. Initial requirements are often ambiguous, incomplete, inconsistent, and usually expressed informally. In order to structure these informal consultation processes initial attempts were made to develop a set of templates that the modeller would fill out during a stakeholder consultation session [12]. These so-called *Requirements Capture Templates (RCTs)* would then be eventually be signed off by both the modeller and the stakeholder. The process of filling out these forms provided structure to stakeholder interview sessions. In addition, these forms had been designed to assist in eliciting information specific to the needs of the underlying agent-oriented conceptual model that the modeller was seeking to build. Stakeholders were directed to provide focused input to the conceptual modelling task, while being shielded from the complexity of understanding and using the conceptual modelling language. In this section, we present overview of requirements elicitation process based on these Requirements Capture Templates. The methodological guidelines that we offer are based on our experiences with early-phase requirements modelling of the emergency services organisation using the *i** framework. While these templates do assist in modelling, they provide only a partial AOCM elicitation strategy. In the next section we describe how these templates can be extended to understanding the social process of elicitation, but for now we concentrate on describing the RCTs exclusively.

A key element of the process of agent-oriented conceptual modelling of a large organisation is the development of a hierarchically structured set of models. In our instance, we started with a highest-level SD model which treated the emergency services organisation as a single agent that interacted with a variety of external/entities and agents. A corresponding SR model was built which detailed the goals, tasks, resources and softgoals internal to the emergency services actor and their relationships to external actors and external dependencies. This SR model did not expand on the internal characteristics of the other actors in any great detail. At the next level of abstraction, we constructed an SD model of the emergency services organisation where each individual department was modelled as a distinct actor. The Requirements Capture Templates (RCTs) became useful from this point onwards. Implicit in the RCTs presented here is the notion of an organisational model such as the one shown in Figure 2. The notion of an Organisation Model can be realised by gathering requirements based on RCTs:

(i) The key role of the first RCT is to identify the agents/actors that would form the basis of the SD and SR models to be constructed (involving department-level actors).

(ii) The next step involves elaboration of the high-level functions by identifying the various activities required to support each of the functions. The Function Elaboration Template was designed to elicit information about specific functions within each department and activities supporting such functions.

(iii) The final step involves identification of intentional relationships that are 'internal' for each actor or agent for each activity described in the Function Elaboration Template. The Activity Elaboration Template was designed to elicit information about specific activities within each actor.

We have argued that the designed templates can ease the requirements elicitation process. However, these templates serve other useful functions as well. They can provide a structured repository and record of stakeholder interviews that can be revisited when requirements are re-negotiated or revised (for instance, when changes are made to models, or when inconsistencies are detected). The detailed rationale recorded in these templates can also be of value in business process re-engineering. To anticipate and support future business process re-engineering efforts in the context of the emergency services agency, we are also detailing alternative solution scenarios by completing additional RCTs that answer "how else" questions (while the primary RCTs represent the "as is" scenarios) [12].

An ontology is commonly paraphrased a description of concepts and relationships that can exist for a community of agents. The notion of ontology, as used in computing, refers to a common vocabulary (with a concomitant set of rules) that is used for building and reasoning about systems. Jurisica et al in [8] present a good survey of ontology-based approaches to information systems development. A variety of domain ontologies have been developed and several are publicly available [6][7]. It is therefore not unreasonable to assume that analysts and application developers would have access to reusable enterprise

ontologies as well as reusable function/activity-specific ontologies. A simple approach to conceptualising an ontology is to view it as a concept vocabulary coupled with a set of rules. The rules may be structural rules that may, for instance, organize concepts in a class hierarchy, or they may be semantic constraints or business rules (an example is a rule in a banking application that requires interest rates for loan accounts to be always higher than those for savings accounts).

We propose to exploit the availability of such reusable ontologies in our approach to early-phase requirements engineering via agent-oriented conceptual modelling. Our key premise is that a pre-existing knowledge-base (or even a concept vocabulary) can significantly ease the early-phase requirements modelling task (by providing some modicum of guidance to a modeller who might be venturing into the task with no prior knowledge or understanding of the application domain). A pre-existing domain ontology can therefore help provide focus to a modeller's early interactions with stakeholders. However, our proposal here is to formalize the process by which ontology-driven elicitation might take place. This is then generalized into a full ontology life-cycle in the requirements elicitation context.

Recall that we are interested in ontologies of two distinct kinds: enterprise ontologies and function/task-specific ontologies. Enterprise ontologies can provide guidance in identifying actors while constructing high-level SD and SR models, by making available certain default organisational structures. These can also provide a vocabulary for more refined (lower-level) SD and SR models. Function/task-specific ontologies (often included within enterprise ontologies) provide detailed concept vocabularies for specific tasks, which can serve as elicitation triggers. Ontologies can also provide a benchmark for completeness that serve to drive the elicitation process. Informally, a conceptual model is deemed to be complete with respect to an ontology if it makes reference to every concept in the concept vocabulary of the ontology. This is in many ways analogous to the notion of completeness of formal theories. A theory is considered complete with respect to a language if it commits to the truth or falsity of every proposition in the language. It is not difficult to conceive of an elicitation methodology that uses this notion of completeness of a conceptual model relative to an ontology to generate elicitation triggers. In effect, every instance of incompleteness, that is, every concept in the concept vocabulary that is not referred to by the conceptual model, serves as a trigger for further questions/probes from the modeller. Ontologies can also support consistency testing of conceptual models. A conceptual model would be deemed inconsistent relative to an ontology if it violated any of the rules associated with the ontology. These could be violations of the structural rules (for instance if a subclass-superclass relationship is reversed in a model) or violations of semantic constraints (for instance, an activity that involves an actor making his/her appointments schedule publicly available may violate security constraints). Each instance of inconsistency can serve as an elicitation trigger, obliging the modeller to seek out additional information in the process of resolving the inconsistency (usually by appropriately modifying the conceptual model).

Much of our discussion above assumes that appropriately constructed domain ontology is made available to the modeller at the start of the elicitation phase. This can be an unrealistic assumption since pre-existing ontologies, where available, may turn out to be inadequate. Key concepts from the domain may not be included in the concept vocabulary, while key relationships may not be represented in the rule-set. The challenge, then, is to devise early-phase requirements modelling methodologies that maintain and update ontologies. These same methodologies might also be used to build (if necessary, from scratch) appropriate domain ontologies.

3. Systemic Semiotic Foundation for Requirements Elicitation

So far we have discussed RCTs that have been used to capture within this domain, information concerning activities within functions and within departments that constitute the organisational model of Figure 2. In this section we introduce the concept of a genre to assist in explaining why these templates are in fact useful. A genre is associated with the cultural context of a completed act of communication and can be thought of as a text type or class. A genre consists of a pattern of stages called genre elements that assist us in recognising the kind of culturally defined situation that we are in. If for example we buy bread from a bakery, the social process that we conduct is a particular type of genre called a service encounter genre. Every genre contains predictable stages; in the case of the bread buying service encounter there will likely be an optional greeting element in which the server attempts to facilitate a sale by welcoming the potential customer, there will be an element in which the customer enquires about the availability and price of various types of goods, followed by an element where the price is agreed and paid for a selection of the items, and a leave taking farewell element to end the transaction. The genre as we have described it contains a small series of genre elements that form a pattern that would encompass a range of actual bread buying texts.

Our knowledge of the genre staging- the expected sequence or organisation of the genre elements- is brought to the fore especially in circumstances in which we cannot easily communicate, for example shopping in a foreign supermarket. We know when to nod our head to thank the cashier and when to hand over the money because we are

familiar with the staging of these kinds of social encounters. In many cases we have a working knowledge of genre at the same time as we are acquiring our first language. Most genres are highly elaborate- tailored to specific types of communication- like those found in organisations and associated with various kinds of technical systems. But some genres are very general and form a kind of cultural property that we draw on in many different social occasions. These more general genres are referred to as canonical genres [10] and systemicists have identified a number of families of them including the Factual and Narrative Genre families.

Each box in Figure 2 involves a set of non-activity structured genres in a large multi-level compositional arrangement of reports. To describe a Function you need to describe its constituent Activities. Activities are to Functions as parts are to wholes. The appropriate genre to use in which to communicate this information is in the form of a canonical report genre that describes "… what an entire class of things is like" [10]. In this case the report genre consists of two elements that enable the purpose of the Function (see Figure 2) and the constituent Activities to be described (respectively these elements are called Purpose and Section Preview). In the Function Elaboration Template, the Purpose element is realised by the Department Name, Function Name, and Function Rationale. The Section Preview consists of the constituent function names. The Activity details for the named functions are provided as separate elements in the Factual report. Each constituent Activity for a given function is provided by its own nested report genre (Activity Elaboration Template). Also the "Additional information" elements in the Activity report Genre in Activity Elaboration Template consist of individual description genres. A description genre is used to describe "what some particular thing is like" [10]. Similarly the Function and Activity Rationale in Function Elaboration Template are also expressible using description genres. From the perspective of Systemic Semiotics, the RCTs function is a set of compositionally related non-activity structured genres that collectively form a single macrogenre. If Systemics is useful for analysing RCTs as text types, what are the language resources that are being used to underpin them? Can some of the difficulties associated with the use of RCTs (for example the construction of domain ontologies) be addressed if these language resources are made explicit and factored into our methodology?

Developers in general are presented with a confusing array of language resources during requirements elicitation sessions. Here we identify five sets of language resources that are known to be of particular importance during requirements gathering, elicitation and representation activities. The first set of language resources are referred to as reference which describes how 'participants' or people, places and things get introduced and 'managed' [5] during interviews and other social occasions. Once participants can be disambiguated we can then determine the correct labels for them. A second set of language resources are used for naming participants. They include language resources which can be used to classify things into different types, as well as other kinds of grammatical and semantic resources. The third group of resources is referred to as taxonomy and enables us to move from an everyday understanding of named participants or lexical items to a technical classification relevant to particular social actions and activities called indexical lexical items. Indexical lexical items can be structured into taxonomies, the organisation of which can be modified by adding more options to them (extension), and also by recognising, incorporating, and refining the differences between successive options (elaboration). The reorganisation of these taxonomies can reflect the evolving understanding of a work practice, situation or domain by the development team for example.

Lexical items that are classified into relevant field taxonomies can be used to form messages using the forth set of language resources are referred to as configuration. Each message will have an agent, utilise a medium of a kind, and exhibit a process. The process is represented by a verb group around which the message or clause will be organised. We can distinguish between different types of processes including material processes, behavioural processes, and mental processes. Material processes express some action, event of happening that is taking place in a social situation. Behavioural processes concern aspects of behaviour which are effectively psychological processes while mental processes involve processes of thinking, feeling, or perceiving. Using the final group of language resources, messages can be assembled to form a goal oriented routine work that consists of a sequence of functional stages called an activity sequence or genre. When the language that accompanies goal-oriented work is recorded and transcribed, it will also exhibit a relatively stable and predictable staging [1][2]. These semiotic resources of reference, naming, taxonomy, configuration and activity sequence can be considered as forming a language resource arc that links the situational language also known as register and the broader social discourses at work in organisations and which will be inextricably a part of the development projects and activities. Of particular interest to requirements gathering, elicitation and representation are those classes of activity sequence which are so commonly used that they are considered to be part of our social and workplace literacy- the so-called canonical genres as we have previously seen. Canonical genres have been used to assist analysts during interviews when they are eliciting information about work practice sequencing, determining the identity of work practice elements, when recovering the expected competencies and behaviours of interactants, when evaluating the work from the point of

view of particular classes of participants, and when exploring work experiences. Hence, the importance of Computational Semiotics to the computing disciplines is in its potential to develop effective computer systems using methods which recognise the importance of semiotic processes.

Registration Activity Example: The registration activity in the *i** model may be represented as two agent system consisting of TrainingSystem and Volunteers. A major concern with *i** models in general is that they are sequence-agnostic. Therefore the developer must provide the necessary sequencing information that is critical in performing analysis with an *i** model. In some actual cases this might be trivial; the developer can rely on their experience of similar situations to develop sequencing that is considered appropriate by the clients. In other cases the development of relevant sequences is hampered by a lack of experience in the client domain and under these circumstances it would be useful to have strategies that enable the developer to elicit this information from the clients. Alternatively it may be the case that neither the clients nor the developer have experience in the work sequences that are to be included in the model. Under these circumstances it would be useful to have methods that would enable the developers and clients to jointly construct appropriate sequences by using knowledge of sequences in related domains. Here we are dealing with developing organisational relevant and defensible artefacts by applying in this case the concept of genre, while simultaneously improving the representational capabilities of the hybrid modelling notation described above. Recalling the discussion in the previous section, we stated that it was possible to deduce the sequencing of routine or repetitive activities, work processes or work practices, by examining the staging in the associated communication taking place- its genre. Genre defined as the global rhetorical organisation of a recurrent pattern of communication has been successfully applied to developing contextual descriptions of information systems [1]. The theory of genre relates a completed act of communication to its (organisational)/cultural context.

Genres can provide the sequencing information that is critical for analysing an *i** model. In this case developing a sequence that is useful as a registration process is largely trivial, but the important point to recognise here is that we can develop a relevant sequence by using a shared understanding of communication in related domains and in so doing develop these sequences in a fashion that is explicit and defensible. The utility of the sequence can be workshopped with the clients. Developing a useful registration sequence for our *i** model registration task involves finding a similar sequence in a related genre. We use a Student Loan genre described in [1] and extract from it a sequence which is used to register first time borrowers see Figure 3 (a). The 'loan' part of this genre involved the following stages or genre elements shown as labelled circles- an optional Greeting element (it can be bypassed), followed by a Service Request, Identification Sought, Materials Out, the possibility of a further Service Request and then followed by an optional 'Finis' stage. The optional elements are referred to as phatic elements; language which is used for maintaining social contact or establishing an atmosphere against which subsequent communication can take place. The 'enrolment subsequence' consists of Regulations and Enrolment elements inserted after the Identification Sought element. When developing our *i** sequence we use essentially the same elements and sequence: Identification Sought, Regulations stage and create an Acceptance (of the Regulation) stage as central, see Figure 3. In the *i** sequence we substitute the SR element for an Orientation element, in which instructions are provided to the user. We dispense with the loan specific MO and SR elements and also omit the phatic elements G and F which are of no direct use in a computer based system. Instead to complete our sequence we append a domain specific Course Selection and Registration elements.

4. Conclusions

This paper demonstrates how Systemic Semiotics, an approach that combines a semiotic model of language called Systemic Functional Linguistics with selected concepts from Social Semiotics, can be applied to create and provide value to Agent-based Systems. The RCTs are themselves linked to the Activity Sequence stage- this resource is also known as Genre- and we saw how a macrogenre formed out of compositionally assembled canonical genres accounted for the structure and function of the RCT. In doing so we provided a semiotic account of ontology-driven template-based *i** requirements elicitation processes. The concept of genre finds application at multiple points in this paper and at various levels of abstraction. As this is an ongoing research, there are many directions that need further research. For example configuration- the building of messages is necessarily involved in building constraints. Also the relationship between taxonomy and ontology is also worth exploring more fully.

REFERENCE

[1] Clarke, R. J. An Information System in its Organisational Contexts: A Systemic Semiotic Longitudinal Case Study PhD Dissertation Department of Information Systems, University of Wollongong, 2000.

[2] Clarke, R. J. Systems Resemblance and Workpractice Evolution: Implications for Work Activity (Re)design. In Goldkuhl, G., Clarke, R. J., & Axelsson, K. (Eds.), *Proc. of the International Workshop- Communication and Coordination in Business Processes* Kiruna, Sweden 22nd June 2005 Linkoping University and University College of Borås, Sweden, 2005, pp. 45-61.

[3] Clarke, R. J. The Work that Analysts Do: A Systemic

Functional Approach to Elicitation. Paper presented at Advances in Information Systems Development- Bridging the Gap between Academia and Industry, *Proc. of 14th International Information Systems Development Conference*, Karlstad, Sweden 15th-17th August 2005.

[4] Crinnion, J. *Evolutionary Systems Development: a practical guide to the use of prototyping within a structured systems methodology* London, United Kingdom: Pitman Publishing, 1991.

[5] Eggins, S. *An Introduction to Systemic Functional Linguistics* London, United Kingdom: Pinter Publishers, 1994.

[6] Fikes, R. Sites Relevant to Ontologies and Knowledge Sharing, 1994

[7] Fikes, R. & Farquhar, A. *Distributed Repositories of Highly Expressive Reusable Ontologies, Knowledge Systems Laboratory*, Stanford University, 1997.

[8] Jurisica, I., Mylopoulos, J., & Yu, E. Using Ontologies for Knowledge Management: An Information Systems Perspective *Proc. of the Annual Conference of the American Society for Information Science*, Washington, D.C., USA, 1999.

[9] Kress, G. Language as Social Practice. In Kress, G. (Ed), *Communication and Culture: An Introduction* Kensington: UNSW Press, 1988.

[10] Martin, J. R. *English Text: System and Structure* Philadelphia/Amsterdam: John Benjamins Publishing Company, 1992.

[11] Tebble, H. *The Genre Element in the Systems Analyst's Interview*. Australian Review of Applied Linguistics, 1993, 15 (2), 120-136

[12] Unni, A., Krishna, A., Ghose, A. K., & Hyland, P. Practical Early Phase Requirements Engineering via Agent-oriented Conceptual Modelling. *Proc. of 14th Australasian Conference on Information Systems*, Perth, Australia, November, 2003, 10pp.

[13] Yu, E. Agent Orientation as a Modelling Paradigm. Wirtschaftsinformatik, 2000, 43 (2): 123-132.

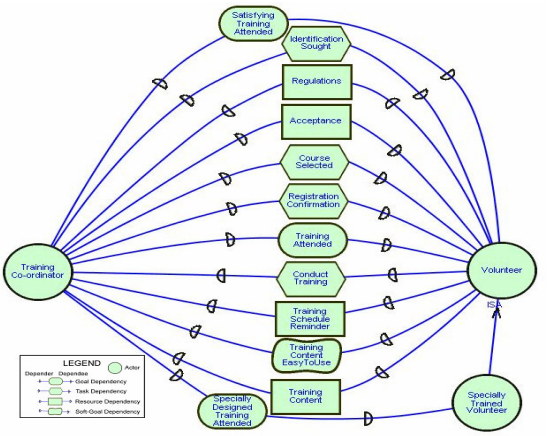


Figure 1: Strategic Dependency model for registration and training, without computer-based system

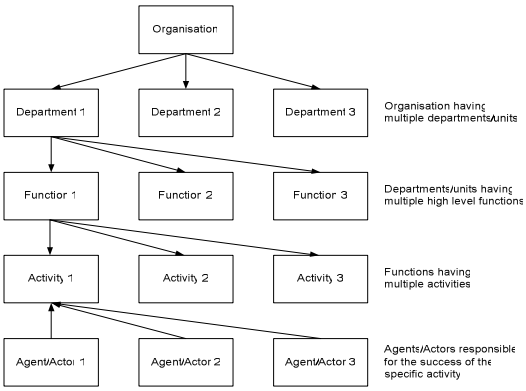


Figure 2: Notion of an Organisational Model

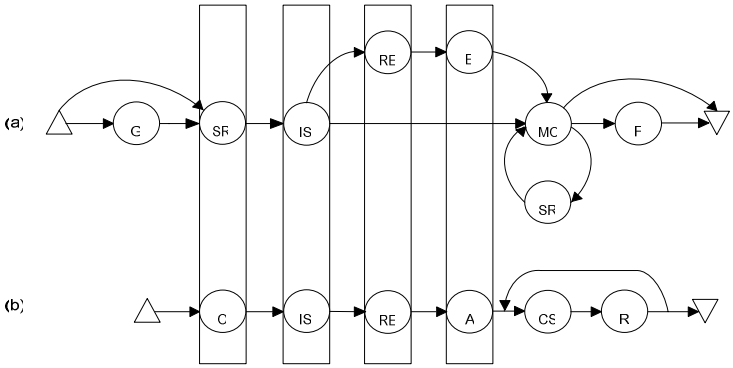


Figure 3: An existing loan genre (a) with an 'enrolment subsequence' is used as the source for sequencing of a canonical registration sequence in (b) that will be used to specify a Registration task in the *i** model. Similarities between the two are described in the text.

Mobility-based Runtime Load Balancing in Multi-Agent Systems

Jan Stender[1], **Silvan Kaiser**[1], **Sahin Albayrak**[1]

[1]DAI-Labor, Technische Universität Berlin, Secretary GOR 1-1,
Franklinstrasse 28/29, D-10587 Berlin, Germany
{jan.stender, silvan.kaiser, sahin.albayrak}@dai-labor.de

Abstract

Supported by the development of Grid Computing technologies, dynamic resource sharing is becoming an icreasingly important issue in the area of distributed computing. In this paper, an infrastructure for dynamic runtime load balancing based on mobile agents is introduced. The approach involves a migration of active agents between hosts in order to exploit distributed system resources to the best possible extent. Service provisioning agents are continuously reachable and overloaded hosts can relocate the agents to other, less loaded hosts at any time. Experimental results show that the approach is suitable for small-scale systems.

Keywords: Load Balancing, Mobile Agents, Grid Computing

1. Introduction

The inherently distributed nature of Multi-Agent Systems brings about a strong relation to the world of Grid Computing. The broad range of common attributes is significant and is stressed in the Grid domain as well as in the domain of software agents [4, 6]. Inspired by the achievements of present day Grid computing technology, Multi-Agent Systems can benefit from the development of uniform, secure and reliable mechanisms for dynamic resource sharing. Grids use load balancing mechanisms in order to allow multiple resource consumers a simultaneous allocation of distributed computing resources. In this paper, an approach for transferring this concept to the agent world is introduced.

Since autonomous agents may consume resources in an ever-changing and barely predictable manner, Grid resource allocation mechanisms do not meet the requirements of agent communities, as they generally schedule the allocation of resources at job startup time rather than runtime. Therefore more dynamic and adaptive load balancing strategies are needed which allow a fast response to changes in the resource consumption behavior of single agents. A possible solution to this problem is a flexible runtime load balancing that utilizes mobile agent technology. The concept relies on a community of mobile agents that dynamically arrange their physical locations, such that available resources are exploited to the best possible extent. Design and implementation of a load balancing infrastructure which involves a reasonable trade-off between migration overhead and quality of load distribution will be described in the following. Experimental results show the viability of the approach.

2. Problem Description

In general, Multi-Agent Systems have a distributed character. Most agent-based applications are designed in a way that agents are spread across multiple hosts. Once started, such agents usually do not change their physical locations during their lifetime.

While agents are busy, they consume system resources, such as CPU capacity, memory or communication bandwidth. If an agent is permanently bound to a certain host, it always depends on the resources made available by that

host. Thus the quality of the services provided by the agent may suffer if the host runs out of resources. Moreover, spare resources in the system might not be used in an optimal manner, as it is possible that a different host is more appropriate to run the agent. A fixed arrangement of agent locations therefore makes it hardly possible to guarantee that resources are optimally used throughout the system. Since agents react to events, e.g. service provisioning requests, their activity potentially occurs asynchronously and is nearly unpredictable. Thus, agent-based applications often underlie an ever-changing demand for resources on all machines.

In order to solve this problem, a self-organized load balancing infrastructure is needed which can quickly respond to changes in the demand for system resources.

3. Agent Mobility and Load Balancing

Mobility is a feature of various Multi-Agent Systems which cuts into weak and strong mobility. While a weakly mobile agent loses its execution state when migrating, a strongly mobile agent is capable of seamlessly continuing its execution at the destination host. Strong agent mobility generally involves three steps: an interruption of the agent's activity on the local host, a transfer of the agent including its program code and state to the remote host and a resumption of the agent's activity at the remote host. The majority of mobility-enabled agent systems are restricted to weak mobility. There are a few ones supporting strong mobility, including JIAC [5], NOMADS [8] and Organic Grid [2]. With the aim of implementing strong agent mobility, NOMADS is built on top of a special Java Virtual Machine which is capable of capturing and restoring the execution state of a Java thread on different machines. Organic Grid relies on a preprocessor that generates Java code from code written in an enhanced Java-based language. The approach adopted by JIAC is to provide a custom language for the implementation of agents which is interpreted at runtime. The execution states of all JIAC agents are part of the JIAC runtime system and can hence be restored on any JIAC-enabled machine.

Strong agent mobility combined with platform independence set the stage for a dynamic resource sharing infrastructure, as provided by various Grid systems. However, most Grid computing infrastructures are designed to only decide at which host to start a job and do not redistribute load in a dynamic manner. There are some systems such as Condor [10] that support load balancing at runtime based on a checkpointing mechanism. When load is getting too high on a single host, jobs running on it can be checkpointed and moved to a different host, i.e. the entire job execution state is captured on the local host and restored on the remote host. This, however, requires a totally homogeneous computing environment, whereas JIAC is Java-based and hence can be run on nearly any machine.

There are some alternative approaches to the problem of Grid load balancing with mobile agents, such as implemented by the Messor system [7]. Messor is capable of arranging a uniform distribution of jobs across peers in large-scale peer-to-peer systems. Agents carrying the jobs migrate between peers, implementing the emergent behavior of an artificial ant colony. However, the job redistribution mechanism is restricted to jobs that are queued, i.e. it does not involve running jobs. This also applies to the Organic Grid [2] approach. Organic Grid manages large tasks with the aid of mobile agents that are cloned on different hosts. Each clone executes a different range of subtasks of the original task, thereby spreading system load across the different nodes. Subtasks that have been started, however, cannot be relocated to a different host. Organic Grid hence also relies on startup time scheduling.

The JIAC infrastructure follows a service based approach. Agents provide services that offer specific functionalities. Service provisioning agents can be either inactive while waiting for new requests or be very busy, using up resources while reacting to a service request at other times. As it is hardly perceivable which host in a Multi-Agent System will be busy at a given time, a runtime balancing mechanism is needed. This allows load distribution in an environment that prohibits load forecasting and therefore renders scheduling at startup time almost useless.

In service-oriented computing infrastructures, downtimes of service providers ought to be reduced to the minimum. Since JIAC offers relocation transparency, service requests never get lost, not even while the service provisioning agent is migrating. As soon as the agent activity is resumed on the target host, service provisioning activity is resumed as well. As opposed to this, systems like *While You're Away* [9] involve a queueing of agents if the network does not provide sufficient resources which causes these agents to be inoperable until execution is continued. Moreover, JIAC does not assume that agents are independent of each other. JIAC agents may communicate in a location transparent fashion and may hence invoke services provided by any other agent at any time.

A placement of agent replicas on different hosts combined with a load-dependent provider selection for each service request might also be a viable approach to a dynamic load balancing system. However, such an approach lacks flexibility in connection with resource-intensive service provisioning activity. It brings about similar problems like the aforementioned Grid systems, in that tasks are scheduled before being executed and cannot be redistributed at runtime.

4. Approach

Dynamic load balancing is a possible application for Agent mobility [1]. In a network of hosts at which agents are running, an agent suffering from lack of free resources can simply be transferred to a remote host where a sufficient amount of resources is available. If such migrations take place in a coordinated fashion, load can be dynamically distributed in way that competition for resources between agents is significantly reduced.

Systems like Comet [3] and WYA [9] have delivered a proof of concept for mobility-based agent load balancing. It has turned out that the ability of agents to change their physical locations in a computer network can be efficiently utilized for load distribution purposes. The kind of agents which systems like Comet and WYA are based on, can be compared to mobile processes, whereas JIAC agents rather have the character of autonomous service providers.

Since services provided by JIAC agents usually remain unused for the most time, it is at best difficult to foresee the behavior of such agents with regard to resource consumption. Due to the asynchronous character of cooperative Multi-Agent Systems, JIAC agents can consume system resources in an ever-changing and barely predictable manner. A load balancing infrastructure for JIAC agents therefore has to quickly respond to changes in the demand for resources of single agents, in order to guarantee full functional capability of the system at any time.

This raises some general questions which have to be taken into consideration with regard to the design of a mobility-based load balancing infrastructure:

Under which circumstances should migrations be initiated? In general, it is important to trigger migrations not too frequently, as a migration implicates some computational and networking overhead, as well as a delay of the agent's activity. On the other hand, the system ought to be capable of quickly responding to significant changes in the demand for resources.

The decision whether and when a migration is initiated should be made locally rather than globally, in order to reduce the risk of creating single points of failure and resource bottlenecks in large systems consisting of many hosts. In general, agents should not migrate unless a migration would bring about a more optimal distribution of load throughout the system. Assuming that no agent in the system frees any allocated resources or allocates any free resources, the system finally ought to reach a state in which no migrations are initiated anymore. With respect to this, it is essential to initiate a migration only as a result of a significant change in the demand for resources. Otherwise, oscillation effects might occur, i.e. a ceaseless movement of agents between hosts might be inevitable.

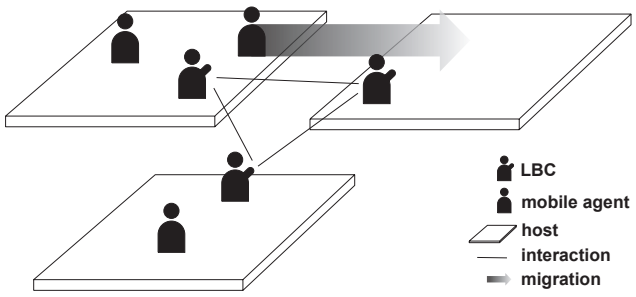


Figure 1. System Architecture

Which agents should be selected for a migration? With regard to this, the main problem is to find out to what extent which local agent is responsible for the load caused on the local host. Unfortunately, this has turned out to be nearly impossible with the JIAC system in connection with certain kinds of system load, such as CPU load. Hence, no satisfactory solution to the problem has been found so far. Agent selection is still a topic for further research.

To which hosts should these agents be moved? It is of little benefit to migrate agents to hosts which do not have a sufficient amount of free resources for additional agents. This would only raise the need for a subsequent migration on the destination host as soon as the agent starts requesting access to the resources provided by that host. Taking into consideration that it is barely possible to assess the exact amount of resources required by a single agent, it is reasonable to assume that hosts with a large surplus of free resources have a higher chance of being able to run an additional agent than others. The lower the utilization of a host is, the more likely it hence should be that the host becomes the destination for the next migration.

5. Implementation

The JIAC load balancing infrastructure requires a dedicated agent on each host to be responsible for the local coordination of load balancing activity. Each such load balancing coordinator (LBC) decides under which circumstances to initiate outgoing as well as to accept incoming migrations. Although LBCs are autonomous, they act as a team in order to establish the load balancing infrastructure. The architecture of the load balancing system is illustrated in figure 1.

Basically, load balancing can be seen as the result of an interaction between LBCs. The fact that the agent discovery service makes it possible for each LBC in the system to find all the other ones, an LBC can find out where to migrate a

local agent by asking other LBCs whether their corresponding runtime environments have sufficient resource capacity to host the agent.

With the aim of finding out when a migration has to be initiated, an LBC c continuously collects information about the current level of load l_c on its local host h_c. All LBCs hold load threshold values t_c representing the largest admissible amount of load, i.e. the smallest possible amount of free resources on their hosts. As soon as the threshold is exceeded, i.e. $l_c > t_c$, a rule-based notification mechanism informs the agent that load balancing activity is necessary. Upon receiving such a notification, a check whether to initiate a migration is performed. Two requirements have to be fulfilled:

1. There's a at least one LBC on a remote host with spare resources which is significantly less loaded than the local one. This requirement helps to mitigate the problems of unnecessary migrations and agent oscillation described in the previous section. All LBCs hold a constant value f_c representing the minimum load difference which is required to carry out a migration. At this time, it is still up to the administrator to find reasonable values for f_c.

2. A fixed minimum amount of time has elapsed since the previous successful migration. This time span t is a global system parameter which also has to be set by the administrator. It provides a means to adjust the reactivity of the system. The smaller t is, the more often migrations will take place. Large values for t will allow migrations only every now and then, and hence will give agents more time to perform their tasks at the cost of the speed at which load is balanced. Reasonable values for t depend on the respective application scenarios.

```
1  proc balance_load(c: LBC, R: Set of LBC)
2  begin
3    if (l_c > t_c) ∧ (t has elapsed since
4         the last migration) then
5      A := {mobile agents on h_c}
6      for an arbitrary a ∈ A do
7        P := {r ∈ R | l_r ≤ min(t_r, l_c − f_r)}
8        D := {p ∈ P | ¬∃p' ∈ P : l_p' < l_p}
9        for an arbitrary d ∈ D do
10         migrate(a, h_d)
11       done
12     done
13   endif
14 end
```

c:	LBC on the local platform	$\mathcal{R}$:	set of all non-local LBCs
l_x:	current load detected by LBC x	t:	global time constant
h_a:	host of agent a	t_x:	load threshold for LBC x
f_x:	minimum migration load difference defined by LBC x		

Figure 2. Load Balancing Algorithm

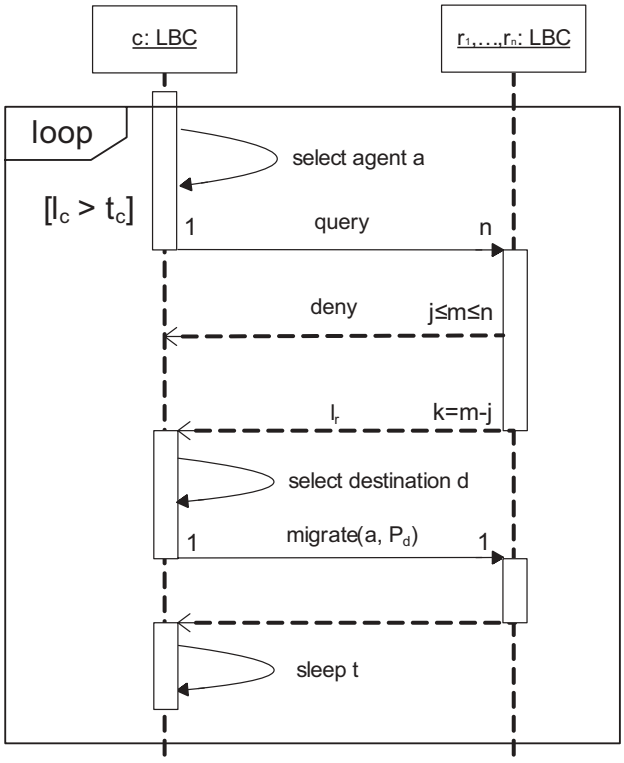


Figure 3. Load Balancing Interaction Protocol

All LBCs use the load balancing algorithm shown in figure 2. In order to provide for a fine-grained redistribution of load in the system, only one agent a is migrated at a time. For the sake of simplicity and due to lack of a good solution to the aforementioned problem concerning agent selection, this agent is chosen by chance. The subset P of potential target LBCs is selected from the set $\mathcal{R} = \{r_1, ..., r_n\}$ of remote LBCs which have runtime environments with enough capacity to host a. The host h_d with the largest amount of free resources eventually serves as the destination host.

Since some of the data used by the algorithm in figure 2 is not locally available for an LBC c, the algorithm is part of an interaction protocol in which all LBCs are involved. Figure 3 illustrates the different steps constituting the interaction. After having selected an arbitrary agent for the migration, a query is sent to all remote LBCs with the intention to find out which remote hosts have spare resources. An LBC r receiving such a query basically does nothing but

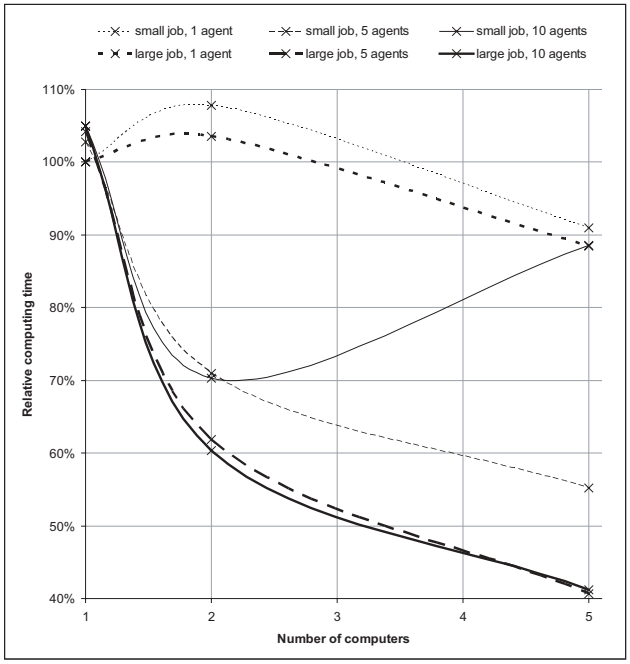


Figure 4. Measurement Results

to check whether its load threshold is exceeded. If so, the reply from r indicates that there is no capacity left for another agent, i.e. accepting the agent would probably cause the load threshold to be exceeded. Otherwise, r replies by sending back its load value l_r which suggests the willingness to accept another agent. A maximum waiting time for replies guarantees that the poll will come to a close. Finally, the least loaded host is chosen and the migration is carried out, followed by a waiting period t.

6. Experimental Results

In order to determine the effectiveness of the load balancing infrastructure, an implementation of a distributed ray tracer has been used. The system consists of a set of ray tracing agents which are responsible for the computation of a picture from an XML-based description of a 3D scene. A dedicated manager agent coordinates the computing activity. When started, it spawns an adjustable number of ray tracing agents. Each of these is assigned the computation of a certain slice of the image. When an agent has completed its task, it initiates a service invocation on the manager agent in order to send back the result. When this has been done by all ray tracing agents, the image is assembled by the manager.

A runtime environment consisting of up to five hosts has been set up, each running on a desktop computer with Windows OS and a CPU clock rate ranging between 600 MHz and 1.2 GHz. Several test series have been carried out, with varying numbers of ray tracing agents and computers. Tests have been repeatedly run in connection with different job sizes. A small job could approximately be completed on a single computer within 10 minutes, a medium job within 30 minutes and a large job within 60 minutes. Load balancing was based on a measurement of CPU load. As for the configuration of the load balancing infrastructure, load thresholds have been adjusted such that all machines can host up to one ray tracing agent without causing the threshold to be exceeded. The ray tracing manager agent and all ray tracing agents were initially started on a certain machine with an average CPU speed.

Figure 4 shows the relative computing time, i.e. the total time needed to compute an image, compared to the time it took on a single desktop computer with an average CPU clock rate. The results reveal that load balancing can significantly reduce the computing time for medium-sized and large jobs. Since the initial distribution of the agents started by the manager takes some time, small jobs could not be distributed in a timely manner. Aside from the forced delay of load balancing activity which is meant to prevent overreactions, this also comes from the fact that a high level of load on a machine brings about a slow-down of the agent management system, which amongst others is needed to carry out migrations. Furthermore, overhead caused by migrations is shown in the "1 agent" graphs. In the worst case, performance can even be lower than without load balancing. Such a loss in performance originates from random migrations which are a result of shortcomings in the load measurement techniques used.

7. Summary and Outlook

In this paper the concept of a load balancing infrastructure for mobility-enabled Multi-Agent Systems has been introduced. The infrastructure implements a dynamic approach to a runtime distribution of load via agent migrations in a network of hosts. When resources on a host are running short, a certain agent which is responsible for the load balancing coordination tries to find a remote host with enough capacity to take over one of the local agents. For this purpose, load balancing coordinators communicate via a certain load balancing interaction protocol. Experimental results confirm that the approach is suitable for small-scale agent-based applications.

The decentralized approach provides for stability and robustness, due to the fact that there is no single point of failure. Unlike systems such as WYA [9], the JIAC load balancing infrastructure does not interrupt and defer the execution of agents after deploying a fixed number of agents. Instead all agents are running and reachable until system load threatens to block an overloaded system. In this case a

fallback mechanism ensures system stability while keeping agent downtime at a minimum.

The load balancing concept offers room for improvements, mainly concerning configuration and scalability. The configuration issue relates to approaches towards an improved automation of the system administration. This involves exploring to what extent it is possible to automatically adjust system parameters, such as load threshold values or migration frequencies. Probabilistic migration decisions might offer an alternative to the deterministic approach.

As far as scalability is concerned, the fact that each LBC communicates with all remote LBCs leads to a high communication overhead in large systems. With respect to this, better alternatives need to be examined, such as hierarchical or decentralized approaches.

As briefly mentioned in section 6, CPU load monitoring is another issue which has to be addressed. Neither a benchmark-based algorithm nor a determination of the CPU load based on data from the operating system kernel deliver reliable load values. In particular, it is hard to find a uniform CPU load monitoring mechanism if different operating systems or multi-processor machines are involved. Mechanisms are needed which enable a transparent monitoring of load caused by single agents, so as to allow for a more efficient selection of agents to migrate.

References

[1] G. Cabri, L. Leonardi, and F. Zambonelli. Weak and strong mobility in mobile agent applications. In *Proceedings of the 2nd International Conference and Exhibition on The Practical Application of Java (PA JAVA 2000)*, Manchester, UK, April 2000.

[2] Arjav Chakravarti, Gerald Baumgartner, and Mario Lauria. The Organic Grid: Self-Organizing Computation on a Peer-to-Peer Network. In *International Conference on Autonomic Computing (ICAC'04)*, pages 96–103, New York, USA, May 2004.

[3] Ka Po Chow, Ricky Y. K. Kwok, Hai Jin, and Kai Hwang. Comet: A communication-efficient load balancing strategy for multi-agent cluster computing. In *Proceedings of Parallel Computing99 (ParCo99)*, Delft, Netherlands, August 1999.

[4] Ian Foster, Nicholas R. Jennings, and Carl Kesselman. Brain meets brawn: Why Grid and agents need each other. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 8–15, New York, New York, 2004. IEEE Computer Society.

[5] Stefan Fricke, Karsten Bsufka, Jan Keiser, Torge Schmidt, Ralf Sesseler, and Sahin Albayrak. Agent-based telematic services and telecom applications. In *Communications of the ACM 44 (4)*, 2001.

[6] M. Luck, P. McBurney, O. Shehory, and S. Willmott. *Agent Technology Roadmap (A Roadmap for Agent Based Computing)*. AgentLink, 2005.

[7] Alberto Montresor, Hein Meling, and Özalp Babaoğlu. Messor: Load-Balancing through a Swarm of Autonomous Agents. In *Proceedings of the International Workshop on Agents and Peer-to-Peer Computing in conjunction with AAMAS 2002*, Bologna, Italy, July 2002.

[8] Niranjan Suri, Jeffrey Bradshaw, Maggie R. Breedy, Paul T. Groth, Gregory A. Hill, and Renia Jeffers. Strong mobility and fine-grained resource control in nomads. In *Proceedings of the Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents, ASA/MA 2000, Zurich, Switzerland*, volume 1882 / 2004 of *Lecture Notes in Computer Science*, pages 2–15. Springer-Verlag, 2000.

[9] Niranjan Suri, Paul T. Groth, and Jeffrey M. Bradshaw. While You're Away: A system for load-balancing and resource sharing based on mobile agents. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, pages 470–473. IEEE Computer Society, 2001.

[10] Todd Tannenbaum, Derek Wright, Karen Miller, and Miron Livny. Condor – a distributed job scheduler. In *Beowulf Cluster Computing with Linux*. MIT Press, October 2001.

Proceedings

International Workshop on Evaluation and Evolution of Component Composition (EECC 2006)

Co-Editors

Wei-Tek Tsai, Arizona State University, USA
Jerry Gao, San Jose State University, USA
Hong Zhu, Oxford Brookes University, UK

Elevating Interaction Requirements for Web Service Composition

M. Hepner M. T. Gamble R. Gamble
Department of Mathematical & Computer Sciences
University of Tulsa
gamble@utulsa.edu

Abstract

Web services are increasingly utilized to create integrated applications from existing components. However, incompatible interaction expectations between Web service interfaces and/or non-Web service interfaces participating in the overall application can inhibit the integration. Frequently, these conflicts are resolved on a case by case basis. Advantages can be gained by understanding integration conflict resolution as formal interaction requirements. This approach enables evaluation of recurring integration conflicts during requirements analysis (rather than as a testing step), thus providing a more complete resolution that is abstracted from specific runtime interaction instances.

1. Introduction

Service-oriented architectures (SOAs) provide a design framework in which to utilize open standards such as Web services (WS) to publish, discover, and communicate between service interfaces. These standards have a positive impact on integration. However, it is likely that substantial conflicts will arise which inhibit interoperability. This is especially true when legacy or third party systems (TPS) are migrated to a service interface [1]. Typically, integration approaches target particular service interoperability needs without regard for more complex application integration. These "hot spots" of integration difficulty focus the attention of service integration designers, but they may also serve to detract from more systemic integration concerns that cross multiple interfaces.

It can be argued that the move to a service-oriented design view is a true paradigm shift requiring service interactions be modeled abstractly as a first-class entity [2]. For example, data that is passed among the same components via non-WS interfaces can compete, conflict, or make inconsistent WS data exchanges. Such exchanges, cast in a WS framework tend to treat the service components as fully encapsulated behind WS interfaces. This is often not the case. Therefore, we advocate elevating the interaction requirements of WS to a level of abstraction in which multiple interface interactions can be expressed to form a *service interaction requirements document*.

Without consideration of the non-WS interfaces present in a component, there can be no guarantee of the correctness of the information processed and exchanged. At a practical level, most interesting systems are large, complex, and contain many proprietary components [3]. Documentation and understanding of the implicit activities of interaction within the system is often lacking. WS integration requirements therefore include:

- A consistent shared data model.
- Verification of all needed data for WS responses.
- Fault management, restart, and shutdown behavior must be made consistent with application expectations.
- Security controls must align with data sharing.

We outline three major requirements that, if not met, can lead to interoperability problems among WS. Once discovered, a design plan for solutions to these problems can be placed in a service interaction requirements document to facilitate resolution.

2. Relevant Work

Requirements for interoperability or integration are examined in terms of migrating existing non-service components to WS [4-6]. The requirements of concern are granularity of service, performance, and semantic meta-data management. Thus, the focus of this work does not include interoperability conflict analysis between WS but rather the use of SOA design approaches as a solution to interoperability in existing architectures, components, and data sources.

Other work examines the process of assessing integration requirements fulfillment. However, in this context, *integration requirements* refers to the

application level requirements that component interaction must satisfy [7]. Thus, this assessment evaluates the integration solution's ability to support the stated application requirements but does not examine the cause of interoperability conflicts, choice of solutions, or flexibility and maintenance issues.

Frequently, integration projects do not apply analysis and design to the selection of an integration solution. This may be caused by a lack of understanding of the basis of interoperability conflicts or application of other decision criteria which are outside the bounds of design (e.g., technology commitments, managerial or corporate standards mandates and limitations of designer skillsets). In particular, the influence of commercial component vendors on integration solutions can be examined from a viewpoint analysis perspective [8].

For SOAs, interoperability refers to the ability of a service to be invoked by any potential client [9]. Interoperability conflicts have been researched in the form of post-integration analysis and in the improved interoperable design of components in the form of WS [1, 10, 11]. Open standards that increase WS interoperability include XML, WSDL, SOAP, HTTP, and UDDI[1]. Using these standards, WS bypass many, but not all, of the issues related to interoperability.

3. Motivating Example

We define a component as an independent and distinct unit that performs some processing task. Components may be legacy systems, third party systems (TPSs), and newly developed systems that may or may not be service based.

Company A wishes to automate order processing with their customers and suppliers. Systems involved in the application integration include the customer's requesting system, Company A's WS and SupplierX's and SupplierY's ordering systems, each shown with two distinct interfaces. Company A's internal systems used for inventory (*IMS*) and customer management (*CMS*) will also be involved in the order process. A representation of the components required in the application integration is shown in Figure 1. The figure displays only part of the data elements and operations which are relevant to our discussion.

[1]XML is the Extensible Markup Language standard. WSDL is the Web Services Description Language standard. SOAP is the Simple Object Access Protocol. HTTP is the Hyper Text Transfer Protocol. UDDI is the Universal Description, Discovery, and Integration standard.

In the proposed order processing system, customers scan information about product items and place orders for the items. When a customer's *OrderRequest* is received by Company A's WS, then send a request (*ItemInvRequest*) to the Inventory Management System (IMS) to verify the item is in inventory. If the inventory quantity is too small to fulfill the customer's order, an *OrderRequest* is automatically placed to the appropriate supplier (SupplierX or SupplierY). After this processing successfully completes, an *OrderResponse* containing a confirmation is sent to the customer by Company A's WS. The last processing step logs the customer's order in the Customer Management System (*CMS*) for record keeping.

Company A uses an in-house WS to accept customer orders. Both suppliers offer WS interfaces for processing orders. The CMS and IMS systems are TPS components that provide WS interfaces. This allows their integration into the application using WS standards. The CMS and IMS systems also provide non service interfaces. For example, the CMS system contains a product catalog that is automatically updated by suppliers. The IMS system periodically updates its own internal inventory database with item information from the product catalog.

4. Service Interaction Requirements

The presence of multiple interfaces indicates less encapsulation of functionality and interaction of a component than may be expected by the service integrator. We advocate the formulation of comprehensive interaction requirements that include non-WS interfaces. Thus, all interfaces of a component must be exposed and uniformly expressed as requirements for interaction in order to compose WS properly and without conflict.

For the specification concerns in this paper, we make the following two assumptions about the application integrations being undertaken: (1) a component can have multiple interfaces, and one of the component interfaces is a WS interface. Each interface is distinguished by the function it performs. Overlapping functionality would be grouped into a single, "larger" interface description.

Let Interfaces(C) be the set of interfaces of component C, where Interfaces(C) = $\{I(C)_1, \ldots, I(C)_k\}$ with finite k. For the purpose of showing the need for requirements elucidation, we designate two specific interface types $I(C)_s$ for the WS interface and $I(C)_r$ for an active, non-WS interface. We can later expand this as needed to multiple interfaces of the same type or multiple different types and use the same principles.

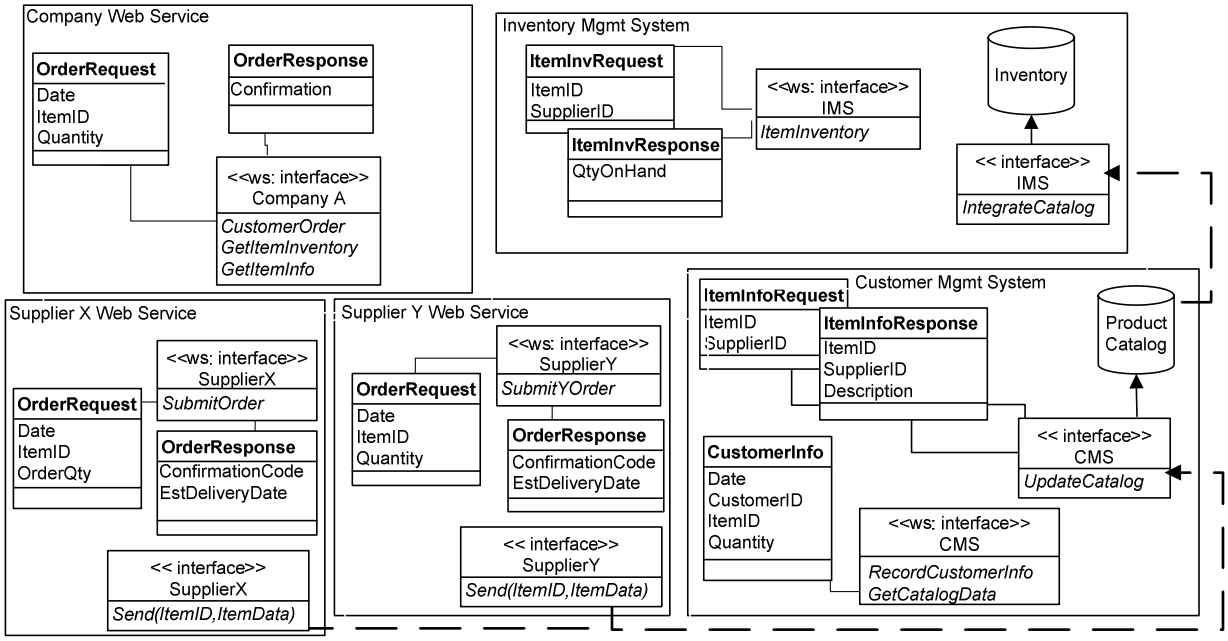


Figure 1. Component Interaction Example

Let D be the set of all possible distinct data elements. We assume that data elements (only) with the same name are semantically equivalent[2]. They are not necessarily syntactically equivalent, however. $I(C)_s$ contains an ordered pair of data element sets $D(C)_s = (\{d_i, \ldots, d_j\}, \{d_n, \ldots, d_m\})$ in which the first set of the ordered pair $D(C)_s$ is request data that can be retrieved by Request($D(C)_s$) and the second set is response data that can be retrieved by Response($D(C)_s$). $I(C)_r$ contains similarly structured data elements in $D(C)_r$, where $\bigcup D(C)_s$ and $\bigcup D(C)_r$ are subsets of D. Request data is sent by a source component and is received as request data by the sink component. Response data is similarly matched between source and sink.

Let A and B be components with Interfaces(A) = $\{I(A)_s, I(A)_r\}$ and Interfaces(B) = $\{I(B)_s, I(B)_r\}$. We illustrate three major requirements that indicate problems with WS integration.

4.1 Requirement 1 – Equivalent Syntax

The first requirement is that *the data exchanged between components is syntactically equivalent.* All possible interface pairs must be examined given more than two components and more than two interfaces per component. Given that we have limited our discussion to two interfaces – one of type s (WS) and one of type r (non-WS) – we examine the intersection of the data involved in the interaction between components A and B. Formally, if Request($D(A)_s$) $\cap$ Request($D(B)_s$) $\neq \varnothing$ or Response($D(A)_s$) $\cap$ Response($D(B)_s$) $\neq \varnothing$, then the WS interfaces share data. If this data does not have equivalent syntax, than an integration solution external to the interfaces is needed to bridge this gap through transformation. The same issue arises for the data sets compared in $D(A)_r$ and $D(B)_r$. The integration solution requirements from the discovery of this type of interoperability conflict should be explicitly stated in the service interaction requirements document.

Table 1 below illustrates the syntax of incompatible items from Figure 1. The data received from the customer in *OrderRequest* of Company A expects the *Date* to be formatted as *mm/dd/yy* (row 1). In contrast, *OrderRequest* of SupplierX uses *mmmddyyyy* date formatting (row 3, Table 1). In addition, the *ItemID* received in the Company A *OrderRequest* has a different format than the *ItemID* contained inside the SupplierX *OrderRequest*. The requirements to resolve these incompatibilities must state what the conflict is and between what entities.

[2] We refer the reader to the vast research on Semantic Web for issues that arise should these not be equivalent.

Table 1. Type 1 Interoperability Conflicts.

$D(A)_s$	$(A)_s$	Request	Type, Format, UOM
Date	Company A	OrderRequest	String, mm/dd/yy
Item ID	Company A	OrderRequest	String, ####-####
Date	Supplier X	OrderRequest	String, mmmddyyyy
Item ID	Supplier X	OrderRequest	String, ####

4.2 Requirement 2 – Complete Request Data

The second requirement is that *messages requesting service provide complete data.* In WSDL terms, the specific message elements required to invoke the service must be fully populated by the requestor. To illustrate, we introduce component F, with Interfaces(F) containing at least $I(F)_s$. Without loss of generality, let component F have a WS interface that receives a request from component A. Then it must be the case that the request data supplied by A encompasses the request data needed by F or $Request(D(A)_s) \supseteq Request(D(F)_s)$. If this is not the case, then component A cannot provide the complete data needed to request service from F. This raises a requirement for interaction and must be stated in the service interaction requirements document. Such a requirement is easily checked by reviewing the WSDL describing the service interface of F.

In Figure 1, when an item's inventory is requested, the message *ItemInvRequest* is sent from Company A to the IMS. This message contains an *ItemID* and *SupplierID.* Examining the *OrderRequest* message received from the customer, it is evident that no *SupplierID* is supplied to Company A. A requirement is needed to detail the processing to obtain the *SupplierID* before the *ItemInvRequest* can be sent. This processing may be added to Company A's WS, coded externally as an adapter to the IMS, or added into a service coordination specification.

4.3 Requirement 3 – Encapsulated Data

The third requirement is that *WS which share data elements must be (logically) encapsulated with respect to each other*. Data shared between two or more WS must be kept consistent such that the WS interfaces appear to totally encapsulate the components. This requirement targets the potential for competing, conflicting, or inconsistent data elements being exchanged outside the service interfaces. Our first concern is where two interfaces of the same component are either both requestors or both responders that share the same date model. That is, where $Request(D(A)_s) \cap Request(D(A)_r) \neq \varnothing$ or $Response(D(A)_s) \cap Response(D(A)_r) \neq \varnothing$. This is a concern because the non-WS interface can pass or accept the same data as the WS. Therefore, we must verify whether the two interfaces are functionally equivalent.

The second concern is the existence of another component sharing the data either as a requestor or a responder outside of the WS interface. In this case, the intersection above is extended as $Request(D(A)_s) \cap Request(D(A)_r) \cap Request(D(B)_r) \neq \varnothing$ (similarly for Response). This can allow a separate communication path for a shared data element and any elements dependent on it. Encapsulation, from a WS perspective, is broken in two components, A and B, if $Request(D(A)_s) \cap Request(D(B)_s) = \varnothing$ and $Request(D(A)_r) \cap Request(D(B)_r) \neq \varnothing$ (similarly for Response). Informally, the components share data in their non-WS interfaces, yet have no such relationship in their WS interfaces and is therefore invisible to analysis of the WS interfaces alone.

Note that $Request(D(A)_r) \cap Request(D(B)_r) \neq \varnothing$ (similarly for Response) does not necessarily imply that the components interact directly on their non-WS interfaces. Though the same data element appears in their interfaces, they may share data via indirect interactions with other components. The service interaction requirements document must flag the occurrence of data sharing across interfaces in order to guarantee the correctness of the WS composition

In Figure 1, components IMS and CMS have WS interactions containing common data elements (i.e., item identifiers). These same components also have non-WS interfaces which act as sinks for data items (via SupplierX and SupplierY) that eventually contribute to the derivation of *ItemID* in their WS interfaces. The non-WS interfaces allow the components to periodically receive updates to a local store containing item data in the form of a product catalog. To analyze only the WS interfaces, we must assume that each component completely encapsulates its own version of the product catalog, allowing it to be logically self-sufficient. For this assumption to be true, we must verify that the product catalog provided to each is the same and possibly immutable [12].

4.4 Interaction Requirements Document

Once the interaction discovery process has been completed, the service interaction requirements document will contain specific statements of potential

interoperability conflicts. This allows proper system design to be performed while taking into consideration how the system may evolve, the ease of system maintenance, and the system execution efficiency. For example, designers can plan to utilize shared integration code when applicable. In addition, application standards can be used to help maintain transaction data consistency and transaction state synchronization rather than creating a new solution for each system-to-system integration. This discovery process will encourage thoughtful analysis of the use and reuse of loosely-coupled integration solutions.

Table 2. Interaction requirements

Interaction Requirement		
	Requirement	**Syntactic Equivalence**
Source	Service/component Request/response Data element	*Company A WS* *OrderRequest* *Date*
Sink	Service/component Request/response Data element	*SupplierX WS* *OrderRequest* *Date*
Conflict	Contributing factor	*Different data formats*
	Resolution	*Convert dd/mm/yy to dddmmyyyy*
	Constraints	*none*

Table 2 above shows an example of an interaction requirement entry in the interaction requirements document. An association between the applicable application constraints and the interaction requirements will be created. This will allow analysis of the interaction resolutions by both the requirement type and the constraints.

Acknowledgement: This work is supported in part by a US AFOSR award FA9550-05-1-0374.

5. Conclusion

Since current integration solutions resolve interoperability conflicts on a case by case basis, there is no method of evaluating the consistent application of these policies. Our approach seeks to make the requirements analysis a repeatable process using a requirement type classification that leads to codifying conflict analysis results within a service interaction requirements document. This approach further allows integration design to be validated for consistency among other application policies and goals such as fault processing, component restarts and shutdowns, and security policy enforcement.

In addition, the identification of system interaction requirements allows future application growth to be considered along with the possibility of component and service upgrades and replacements. Such changes are treated as new integrations and flow back through the same requirements steps. When changes cause new interactions to occur, their interaction requirements may be analyzed to determine their type as well. Placement of integration solutions in the system architecture can have long-term effects for the system evolution and maintenance costs.

6. References

[1] L. Davis, Gamble, R., Hepner, M., Kelkar, M., "Toward formalizing service integration glue code," IEEE Int'l Conf. on Services Computing, 2005.

[2] L. F. Andrade, Fiadeiro, J.L., "Composition contracts for service interaction," *J. Universal Computer Science*, vol. 10, pp. 375-390, 2004.

[3] R. Ellison, J., "Trustworthy integration: Challenges to the practitioner, CMU/SEI-2005-TN-026," Software Engineering Institute, CMU, October 2005.

[4] G. Lewis, Morris,E., O'Brien,L., Smith,D., Wrage, L., "SMART: The service-oriented migration and reuse technique, CMU/SEI-2005-TN-029," Software Engineering Institute, CMU 2005.

[5] I. Wong-Bushby, Egan, R., Isaacson., C., "A case study in SOA and re-architecture at company ABC," Hawaii Int'l Conference on System Sciences, 2006.

[6] V. Niranjan, Anand,S., Kunti, K., "Shared data services: An architectural approach," IEEE Int'l Conference on Web Services, 2005.

[7] T. Wendt, Brigl,B., Winter., A., "Assessing the integration of information system components," presented at Workshop on Interoperability of Heterogeneous Information Systems, 2005.

[8] M. T. Gamble, Gamble, R., and Hepner, M., "Understanding solution architecture concerns," 2nd Int'l Workshop on Models and Processes for the Evaluation of Off-the-Shelf Components, 2005.

[9] M. Stevens, "Service-Oriented Architecture Introduction," http://www.developer.com/services/article.php/1010451, 2003.

[10] M. Hepner, Gamble, R., Kelkar,M., Davis,L, Flagg, D., "Patterns of conflict among software components," *J. Systems and Software*, vol. 79, pp. 537-551, 2006.

[11] M. Hepner, Gamble, R., "Connectors as integration dervices," 5th IEEE/IFIP Working Conference on Software Architecture (WICSA), Components and Services Workshop, 2005.

[12] P. Helland, "Data on the Outside vs. Data on the Inside An Examination of the Impact of Service Oriented Architectures on Data," http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/dataoutsideinside.asp.

Unanticipated connection of components based on their state changes notifications

Luc Fabresse, Christophe Dony and Marianne Huchard
LIRMM – Université Montpellier II
161, rue Ada
F-34392 Montpellier Cedex 5, France
E-mail: {fabresse,dony,huchard}@lirmm.fr

Abstract

Component-based software development is a promising track in software engineering to improve reuse. This paradigm is based on the unanticipated connection of independently developed black-box components. However, any existing proposals enable connections of components based on their state changes notifications without requiring that specific code related to the connection is integrated into components. In this article, we propose a solution to support these kinds of connections. Our solution introduces component properties *and special connectors. We show that properties ease component programming and connectors enforce strict separation between functional code and code dedicated to connection. We develop a prototype in Squeak named* SCL *(Simple Component Language) to give a concrete form to our proposition.*

1 Introduction

Software engineering focuses on component-based models and languages [1, 2, 3, 4, 5, 6, 7] in order to increase reuse as stated by component software development [8]. This new paradigm is based on the following definition: *"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties"* [9]. Starting from this definition, component-based languages have been built with different or adapted abstractions and mechanisms to provide unanticipated connection between two or more components. To support unanticipated connections, a component definition sets up which services are provided and which services are needed and should make no assumptions about its possible connections or other components to be connected with.

In this paper, we focus on connections based on notifications of component state changes. These kinds of connections are really useful to trigger a component service whenever another component notifies that its state changes. Although these *state connections* are very useful, they are not possible in actual component-based languages in an unanticipated way i.e without writing specific connection code in components. We introduce component properties to avoid this constraint and provide a mechanism based on connectors to support unanticipated *state connections* between components. Properties represent external state of components and connectors represent connections between components. We enhance the actual prototype of our language SCL (Simple Component Language), a simple dynamically typed component-oriented language to support properties and connectors. SCL offers a unified mechanism to build standard and based on properties notifications connections.

The paper is organized as follows. Section 2 discusses why connections based on component state changes notifications are interesting. Section 3 presents the basis of component-oriented programming in SCL. Section 4 presents component properties and how to connect components using properties in SCL. Section 5 discusses related works. Finally, section 6 concludes and presents future work.

2 Motivation

Triggering operations as a consequence of state changes in a component is not a new idea. It is a related idea to daemons or procedural attachments [10] in frame languages, where it was possible to attach procedures to an attribute access which is then executed each time this attribute is accessed, or the Observer design pattern [11]. These kinds of interactions are particularly used between "views" (in the MVC sense [12]) and "models".

In component-based languages, this must be done in an unanticipated way and with strict separation between the component code and the connection code to enable components reuse. However, existing proposals fail to solve these two main constraints. Connecting components based on their state changes notifications always requires that component programmers add special code (like event signaling) in components. For example, Figure 1 shows Archjava [4] code of a CHATCLIENT component that has specific code to enable "state connections".

```
component class ChatClient {
  private String chatText;

  public port Accessing {
    provides String getChatText() {
      return chatText;
    }
    provides void setChatText( String s ) {
      String oldV = Accessing.getChatText();
      chatText = s ;
      Accessing.chatTextHasChanged( oldV, s );
    }
    broadcasts void chatTextHasChanged(
      String oldValue,
      String newValue );
  }
  ...
}
```

Figure 1. An incomplete CHATCLIENT component declaration in Archjava.

At connection time, the broadcast service `chatTextHasChanged` of a CHATCLIENT component may be bound to provided services of other components. For example, it can be bound to a service of a CHATCLIENTGUI component that refreshes the GUI. This connection is only possible because the CHATCLIENT component has a broadcast service which is invoked in its code. This constraint of integrated special code in notifier components is inconsistent with unanticipated connection and strict separation between component code and connection code. It has been illustrated in Archjava but similarly appears in other approaches. In fact, a software architect must be able to construct a software by choosing existing components, adapting them and finally build connections between them without requiring that they have been defined in a convenient way. With our approach, we provide component properties coupled with special connectors to support unanticipated "state connections" of components.

3 Component-oriented programming in SCL

In this section, we present SCL, a simple and dynamically typed component-oriented language, upon which we propose a solution to support unanticipated connection of components based on their state changes notifications. SCL integrates the common features of component-based languages: *component*, *service*, *port*, *property* and *connector* and offers standard mechanisms of *service invocation* and *unanticipated connection* with some variations.

A component has internal state and services. Services represent component behavior like object methods in object-oriented paradigm. Ingoing services are services defined in a component and invoked by other components. Outgoing services are invoked by a component (in its code) but not defined on it and this service may be bound at connection time. Required services are those that must be bound at connection time to an ingoing service of a connected component in order to solve the outgoing service invocation.

Ports are interaction points of components and therefore support connections and service invocations. Figure 2 shows a CHATCLIENT component with two ports.

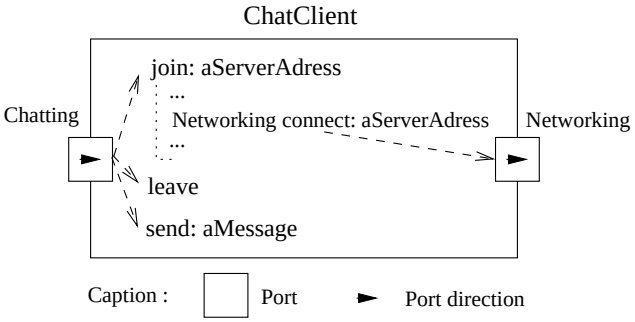


Figure 2. A CHATCLIENT component

`Chatting` is an ingoing port of the CHATCLIENT component providing a set of services that the other components can invoke. In this example, `Chatting` provides the services `join:`, `leave` and `send:`. `Networking` is an outgoing port used in the code of the CHATCLIENT component to invoke services provided by other components. A service invocation is syntactically alike message sending in object-oriented languages. The receiver of a service invocation is a port and the selector is a service name. If the receiver is an ingoing port, the executed service is the matching name service defined on the component whose this port belongs to. If the receiver is an outgoing port, the effectively executed service depends on connections. The outgoing service `connect:` used by CHATCLIENT to make network connection to a chat server has to be bound to a provided service of another component at connection time. Figure 3 shows SCL code of the definition of a CHATCLIENT component.

```
SCLCOMPONENTBUILDER create: #ChatClient
  outPorts: 'Networking'
  inPorts: 'Chatting'.

CHATCLIENT>>init
  (self port: #Chatting)
    addServiceSelector: #join: ;
    addServiceSelector: #leave ;
    addServiceSelector: #send:.

CHATCLIENT>>join: serverAdrs
  (self port: #Networking) connect: serverAdrs.
```

Figure 3. SCL **declaration of** CHATCLIENT

We choose to represent connections between components by *connectors* [13] in SCL in order to provide a good separation of components code and connections code. Connectors connect components through their ports and help solving adaptation problems [14] without using any Adapter pattern [11]. In a connector, outgoing ports are called *source ports* because service invocations come from these ports. *Target ports* are ingoing ports used to invoke services of components. Figure 4 shows the connection of a CHATCLIENT component with a NETWORKMANAGER component that provides services `openConnectionTo:`, `closeConnection` and `sendData:` through its port `Networking`.

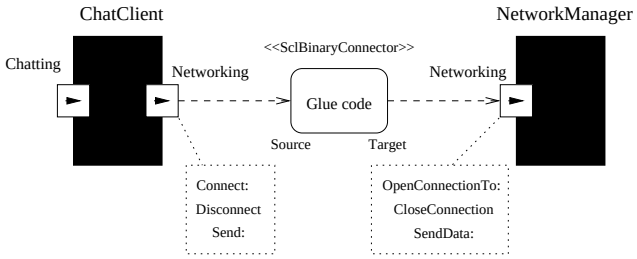


Figure 4. Connection of two components

At connection time, components are like black boxes and ingoing or outgoing services of a component are known by reading the component specification or by introspecting the component. Figure 5 shows the code of the connection of Figure 4 that deals with a frequent adaptation problem which is the non-matching service name problem [15].

In this example, the connection is achieved with the special connector SCLBINARYCONNECTOR which has only one source port and one target port. Glue code is written in the connector to deal with each service invocation and solve adaptation problems: the invocation of the `connect:` service in the CHATCLIENT component code results by executing the `openConnectionTo:` service of the NETWORKMANAGER. This connection mechanism is the basis of component-oriented programming but it can not express

```
chat := CHATCLIENT new.
netManager := NETWORKMANAGER new.

SCLBINARYCONNECTOR new
  source: ( chat port: #Networking )
  target: ( netManager port: #Networking )
  glue: [ :source :target :message |
    ( message selector == #connect: ) ifTrue: [
    ^target openConnectionTo:
      (message arguments first) ]
  ]; connect.
```

Figure 5. SCL **connection code**

unanticipated connections of components based on their external state changes notifications. As in Archjava, a component has to integrate an outgoing port and notifying services to enable these kinds of connections. Our goal is to discard this constraint using component properties.

4 Component properties

To support components connection based on notifications of their state changes, we introduce the concept of *property*. This property concept enhances the idea of property in the Javabeans component model [16] with strict separation between component code and connection code. For example, a CHATCLIENT component has a property named `chatText`. This means that it is possible to get and set a `chatText` value (string messages from chat users) to a CHATCLIENT component. The two first lines of code on Figure 8 show the declaration of the CHATCLIENT component with its `ChatText` and `NickName` properties. The `ChatText` property declaration does not enforce the use of an instance variable named `chatText` to implement this property: other internal implementations can be choosen. When a programmer declares a property, the component is automatically equipped with two ports: an *access port* and a *notifying port*. The property access port is an ingoing port that provides at least getter and setter services. This port avoids the services to respect particular syntactic name convention. The *notifying port* is an outgoing port, which is used to invoke services during property accesses. These services are defined in the SCL component model. For example, the service `nac:value:oldValue:` (nac is an acronym for Notify After Change) is invoked after a property is modified with the new and the old value of the property as parameters. Another service, the `nbc:value:newValue:` (nbc is an acronym for Notify Before Change) service, is invoked before the property is modified with the current value and the next value of the property as parameters. In fact, all defined services have two main characteristics: when they are invoked (before or after the property modification) and what

a connected component is able to do (nothing, prevent the modification or change the property value). An example of connection using properties is depicted on Figure 6 and the corresponding SCL code is shown on Figure 7.

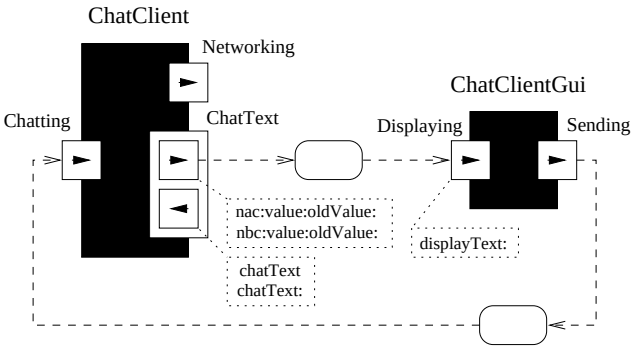


Figure 6. Connect components using properties changes notifications

```
client := CHATCLIENT new.
clientGui := CHATCLIENTGUI new.

SCLBINARYNACCONNECTOR new
  source: ( client notifyPortOf: #ChatText )
  target: ( clientGui port: #Displaying )
  glue: [ :source :gui :message |
    gui displayText: (message arguments second).
  ]; connect.
```

Figure 7. Connecting two components based on a property changes notifications

In this example, each time the `chatText` property of the CHATCLIENT is changed, this results in changing the displayed text on the CHATCLIENTGUI due to the SCLBINARYNACCONNECTOR that only considers `nac:value:oldValue:` invocations on the source port. Special connectors like SCLBINARYNACCONNECTOR ease connection writing. Moreover, a software architect is able to build reusable connectors that can be included in a library of generic connectors. Actually, SCL provides different kinds of connectors like SCLBINARYNACCONNECTOR, SCLBINARYNBCCONNECTOR, PROPERTYBINDERCONNECTOR ensuring that the value of the target property is always synchronized with the value of the source property.

Figure 8 shows the complete code of the CHATCLIENT. Figure 9 shows the complete connection code necessary to build the application. Figure 10 shows a simulation code and Figure 11 shows the screenshot of this simulation[1] execution.

[1]The whole code is available at http://www.lirmm.fr/~fabresse/scl

```
SCLCOMPONENTBUILDER create: #ChatClient
  properties: 'chatText nickName'.
  outPorts: 'Networking' inPorts: 'Chatting'.

CHATCLIENT>>init "presented in Figure 3"

CHATCLIENT>>primitivechatText
  "internal component accessor defined by the
  programmer and used by the generated chatText
  property accessor"
  chatText ifNil: [ chatText := '' ].
  ^chatText

CHATCLIENT>>primitivechatText: newV
  chatText := newV

CHATCLIENT>>primitivenickName
  nickName ifNil: [ nickName := 'anonymous' ].
  ^nickName

CHATCLIENT>>primitivenickName: n
  nickName := n

CHATCLIENT>>leave
  (self port: #Networking) disconnect.

CHATCLIENT>>send: aMessage
  (self port: #Networking)
    send: ('<', self nickName, '> ', aMessage)

CHATCLIENT>>receive: aMessage
  (self accessPortOf: #chatText) chatText:
    (self chatText, String crlf, aMessage).
```

Figure 8. The CHATCLIENT SCL code

Properties are a new feature that helps component programming by providing a higher abstraction to component programmers and software architects. Figure 12 illustrates this fact because a new functionality is added to our chat client with only one "state connection".

This connection allows our chat client application to automatically send the current title played by our music player to other chat users. In other words, each time the `CurrentTitle` property of the component MUSICPLAYER is modified, a message is sent to chat users using the `send:` service of the CHATCLIENT component through its `Chatting` port.

5 Related Work

Our approach is similar to Javabeans component model [16]. A Javabeans programmer declares properties through syntactic name conventions like `get` and `set` and writes the event signaling code to enable connection based on Javabeans properties event signals. The Javabeans model distinguishes different kinds of properties depending on signals like *bound* properties that notify connected Javabeans

```
ChatClientApp>>createChatClientFor: aNickName
  | chatClient chatClientGui |

  chatClient := SclChatClient new.
  chatClientGui := SclChatClientGui new.

  SclBinaryConnector new
    source: (chatClient port: #Networking)
    target: (NetworkManager new port: #Networking)
    glue: [ :chat :netM :message |
      (message selector == #connect:) ifTrue: [
        netM openConnectionTo:
          message arguments first.
      ] ifFalse: [
        (message selector == #disconnect)
        ifTrue: [ netM closeConnection. ]
        ifFalse: [
          netM sendData: message arguments first.
        ]
      ]
    ] ; connect.

  SclBinaryConnector new
    source: (chatClientGui port: #Sending)
    target: (chatClient port: #Chatting)
    glue: [ :gui :chat :message |
        chat send: ( message arguments first )
    ] ; connect.

  SclPropertyBinder new
    bind: (chatClient property: #chatText)
    with: (chatClientGui property: #displayText).

  SclPropertyBinder new
    bind: (chatClient property: #nickName)
    with: (chatClientGui property: #chatUserName).

  (chatClient accessPortOf: #nickName)
    nickName: aNickName.
  (chatClient accessPortOf: #chatText)
    chatText: 'Welcome in ChatClient app v 0.0.1'.
  (chatClientGui port: #Displaying) show.

  ^chatClient
```

Figure 9. Chat client application code

```
| chatClientApp chatServer bCli rCli |

chatServer := ChatServer new startOn: 8080.

chatClientApp := ChatClientApp new.
bCli := chatClientApp createChatClientFor: 'Bob'.
rCli := chatClientApp createChatClientFor: 'Rick'.

(bCli port: #Chatting) join: '127.0.0.1:8080'.
(rCli port: #Chatting) join: '127.0.0.1:8080'.

(rCli port: #Chatting) send: 'Hello guys'.
(bCli port: #Chatting) send: 'Hi Rick'.
(bCli port: #Chatting)
  send: 'Did you have a nice day ?'.
```

Figure 10. Simulation code

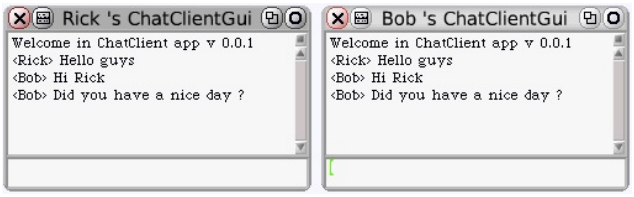


Figure 11. Screenshot of the simulation execution

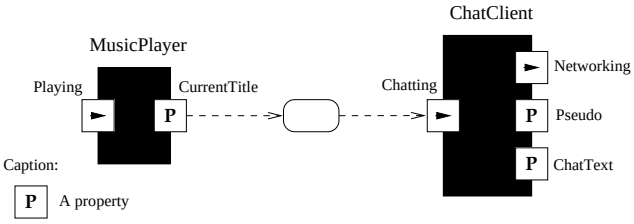


Figure 12. Evolving the chat client application using connection based on properties

after each value changes. A Javabean component programmer has to write a lot of code that is not relevant for the component but for its connection. The automatic and hidden use of Adapter pattern enables to create connections between Javabeans without requiring that notified components integrate specific code like in the Observer pattern (notified component have to define an update method).

In the Corba Component Model [3], the notifier component must integrate an event source that emits an event notifying its value changes and notified components have to offer an event sink that receives compatible events. This is not an unanticipated connection although the Adapter pattern can also be used to avoid specific code in notified components.

In the Fractal model [5], components have interfaces (defining a set of services) that are server (provided) or client (required). Components are connected through primitive bindings or composite bindings. A primitive binding is a fixed interface connection mechanism that binds one client interface with one server interface. Binding components also called connectors represent composite bindings. Like in Archjava with broadcast services, in Fractal, notifying services have to be coded in components and put in client interfaces although they are not required by the component.

6 Conclusion and Future Work

In this article, we show that the unanticipated connection mechanism of components in most component-based languages is not enough to create connections based on component states notifications. We propose a solution to enable this kind of connections in an unanticipated way and with a strict separation between the component code and the connection code. To achieve this goal, we introduce component *properties* based on ones in the Javabeans component model. Properties allow programmers to declare external state of components that will be used by software architects at connection time to create connections among notifications emitted by these properties. The connection code is encapsulated into connectors allowing code separation and extensibility because the software architect is able to build new connectors to extend this connection mechanism. Our proposition is based on our SCL (Simple Component Language) language prototyped in Squeak.

On the one hand, we will extend the property concept to support multi-valued properties i.e. properties whose value is a collection of elements. These properties changes are different such as adding or removing an element and new connectors are needed. On the other hand, connections based on properties notifications have to be used carefully because of the possibility of infinite recursive notification loop. For example, glue code in a connector is executed each time a property notifies a change and this glue code must not change directly or indirectly (through other connections) this property otherwise there is an infinite notification loop during runtime. This problem has to be detected before runtime with program analysis.

References

[1] J. C. Seco and L. Caires, "A basic model of typed components," *Lecture Notes in Computer Science*, vol. 1850, pp. 108–129, 2000. [Online]. Available: citeseer.ist.psu.edu/article/seco00basic.html

[2] R. Monson-Haefel, *Enterprise JavaBeans*. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 1999.

[3] Object Management Group, *Manual of Corba Component Model V3.0*, 2002, *http://www.omg.org/technology/documents/formal/components.htm*.

[4] J. Aldrich, C. Chambers, and D. Notkin, "Archjava: connecting software architecture to implementation." in *ICSE*. ACM, 2002, pp. 187–197.

[5] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani, "An open component model and its support in java." in *CBSE*, ser. Lecture Notes in Computer Science, I. Crnkovic, J. A. Stafford, H. W. Schmidt, and K. C. Wallnau, Eds., vol. 3054. Springer, 2004, pp. 7–22.

[6] P. H. Fröhlich, A. Gal, and M. Franz, "Supporting software composition at the programming-language level," *Science of Computer Programming, Special Issue on New Software Composition Concept*, vol. 56, no. 1-2, pp. 41–57, April 2005. [Online]. Available: http://www.cs.jhu.edu/ phf/publications.shtml

[7] R. Marvie, "Picolo: A simple python framework for introducing component principles," in *Euro Python Conference 2005*, Göteborg, Sweden, june 2005.

[8] G. T. Heineman and W. T. Councill, Eds., *Component-based software engineering: putting the pieces together*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[9] C. Szyperski, *Component Software: Beyond Object-Oriented Programming (2nd Edition)*. Addison-Wesley, 2002.

[10] M. Minsky, "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, P. Winston, Ed. ny: mgh, 1975, pp. 211–281.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison Wesley, March 1995.

[12] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view-controller user interface paradigm in smalltalk-80," in *Journal of Object-Oriented Programming*, vol. 1, Août-Septembre 1988, pp. 26–49.

[13] M. Shaw, "Procedure calls are the assembly language of software interconnection: Connectors deserve first-class status," in *ICSE '93: Selected papers from the Workshop on Studies of Software Design*. London, UK: Springer-Verlag, 1996, pp. 17–32.

[14] J. Sametinger, *Software engineering with reusable components*. New York, NY, USA: Springer-Verlag New York, Inc., 1997.

[15] P. H. Fröhlich, "Component-oriented programming languages: Messages vs. methods, modules vs. types," in *Proceedings of the Workshop on Programming Languages and Computer Architecture*. Bad Honnef, Germany: Technical Report 2007, Institute for Computer Science and Applied Mathematics, Christian-Albrechts-University, Kiel, Germany, May 2000. [Online]. Available: citeseer.ist.psu.edu/317429.html

[16] G. Hamilton, "JavaBeans," Sun Microsystems," API Specification, July 1997, version 1.01.

Service Design with the ServiceBlueprint™

Jochen Meis, Lothar Schöpe
Fraunhofer Institute for Software and Systems Engineering (ISST)
{Jochen.Meis | Lothar.Schoepe}@do.isst.fraunhofer.de

Abstract

Service Engineering is the approved approach for designing customer oriented service processes. Service engineering consists of several phases; from situation analysis to service creation and service design to service management. This article will describe how the method service blueprint can be used to design service processes with special focus on the new application domain smart living. Smart living includes all actions to enlarge a flat to a smart home for living. One special requirement of this application domain is the use of local components (actuators, sensors) within service processes. Now the method service blueprint has to be extended to enable it to model the information produced or consumed by these local components. This article will show how this extended method supports service providers to improve the quality of customer oriented service processes and the derivation of needed interfaces of involved actors.

1. Introduction

A new application of software engineering is the application domain of smart living or sustainable living. Within this domain service platforms are designed and realized with the goal to support value added services. In this context value added services integrates microelectronics, home automation and services to enhance the attractiveness of flats and buildings. Especially real estate companies are interested in an effective design and management of their services. Value added services focused on inhabitants are grouped to consulting and information, care and supervision, leisure time activities, repairs, mobility and delivery, safety and security, supply and disposal [1]. (see Figure 1).

On the one hand basic requirements of installation of local components within the flat, building or campus are needed. On the other hand the integration of external service providers is necessary to provide it-supported value added services (for example emergency sensors for elder people or facility control systems for companies).

Local components are for example outlet systems, light installation, location systems, mobile sensors, cameras as well as white goods (refrigerators, washing machines, etc.) or consumer electronics or devices for energy supply (heating systems, etc.). The networking of the local components is organized by different techniques (EIB, LON, IR, Bluetooth, LAN/WLAN [2][3]). Various mobile devices (cell phone, smart phone, PDA, Tablet-PC) are used to control local components via a service gateway. If needed, because different network technologies are used, the service gateway transforms queries to different protocols.

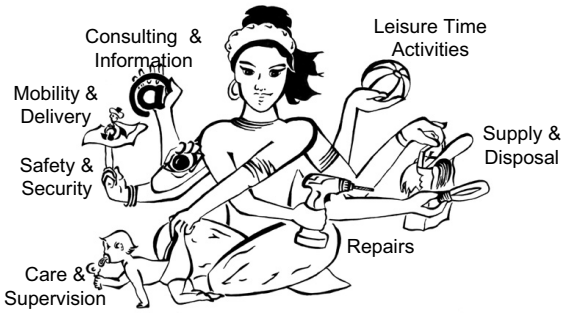


Figure 1: Value added services for smart living

External service providers are companies selling services (e.g. security services, delivery, caregivers). Due to providing it-supported value added services real estate companies expand their offer to a service provider (one face to the customer – from one source) to compete within the future market of smart living [4]. New services will be developed by different process models of service engineering. A typical service engineering model has different numbers of phases. Basically it has four

different phases: situation analysis, service creation, service design and service management. The whole life cycle of a service will be supported by these phases. To provide the service engineering process within smart living the method of service blueprinting can be adopted within the phase of service design. As a part of service design the blueprinting is used to model the overall service process integrating smart local components. Within the blueprint the different action parts of customers and providers will be shown. Therefore, the explicite intelligence of a smart home is unaccounted. For this, the blueprint will be extended with further action of a smart home.

2. State of the art

The using of local components and their networking is proved since several years in national and international projects. The projects are accompanied by universities or various research institutes [3][5][6]:

"AwareHome", Atlanta, USA

"Smart Home", Edingburgh, Great Britain

"InternetHome", London, Great Britain

"HomeLab", Eindhoven, Netherlands

"IT Neighborhood", Stockholm, Sweden

"FutureLife", Hüneberg, Switzerland

"inhaus", Duisburg, Germany

"Das intelligente Haus", Gifhorn, Germany

"Trunifed Haus", Ahaus, Germany

"Vision Wohnen", München, Germany

All projects have a same focus; they are focusing on different technology aspects for home automation. A house or flat was build to focus on specific usage of local components and to prove different networking concepts and technologies (EIB, KNX, Powerline, TCP/IP, etc.). Focusing on diffente aspects like home automation control in combination with mulitmedia and entertainmant devices or technology integration are the main aims of the listed projects. Some projects although take different target groups into account (e.g. families or elderly people).

The "InternetHome" proves different local components of one manufacture (Honeywell) to control the house and proves the benefit to integrate network components from another manufacture (Cisco). Primary aim of "Das intelligente Haus" is the network of local compontents to raise the passive security for intrusion, fire and leakage detection of the home. The secondary aim is focusing on optimal association with resources (gas, water, power) supported by energy control components. "Smart Home" verifies useful local components for assisted care living to get optimal usability to the elderly people. To raise social responsibility and competence of inhabitants within a district is checked in project "IT Neighborhood" by using a local communication network based on TCP/IP. During startup of the "IT Neighborhood" project especial erldery people are trained on using the computer and the internet. Developing an intelligent surrounding to prove assisted living especially for elder people is subject of project "AwareHome". Within the project "AwareHome", local components should identify personal circumstances to alert, to help, to assist elder people in their everyday life. If with local components must be interacted (messages to read out and confirm, make attitudes) this interaction is made by gestures, which are recognized by visual assistance systems installed inside the home.

Focus of these entire projects is raising the comfort and the easy handling by using local components inside a smart home. The realization of services, provided directly inside a customer's home, was not viewed by projects, except "FutureLife", where products could be ordered online and will be placed in the "SkyBox". The developing of the delivery service, as one example of a value added service, was prototyping. But how about developing services for a mass market using a service platform and it-supported value added services? For developing services the usage must be standardised. Service blueprinting visualisation interaction of the involved actors for documentation, service optimization and interface development.

3. Smart Living

The main part of research project "SmarterWohnenNRW" [7] is to develop new concepts of it-supported value added services. The new services will be provided to inhabitants through their real estate company. The services will be provided by professional service providers in cooperation with the real estate company as a new distribution chanal. Another objective within the project is to identifiy local networked components for various it-supported value added services focusing on mass market suitability. Partner of this research project is Fraunhofer-IMS, Fraunhofer-ISST and Hattinger Wohnstätten e.G. (HWG). 200 flats and houses will be renovated and upgraded by the HWG to prepare the service platform for testing and proving all it-supported value added services. On the one hand the project should validate the concepts and on the other hand it should prove the scalability and portability of the developed concepts and techniques for a mass market. Within the scope of the project the houses will be equipped with local components and ip-network. This

will provide an infrastructure to settle different it-supported value added services.

The service gateway realizes the integration and the communication of the local components. The service platform performs the integration of external service providers and their communication (see Figure 2)

One very important thing is to address local components, which can be on a campus or outside the home, to provide local information (for example: weather forecast). The unique address and security connection will be handled by the service platform and service gateway. Another relevant thing is to provide the adequate services to inhabitants.

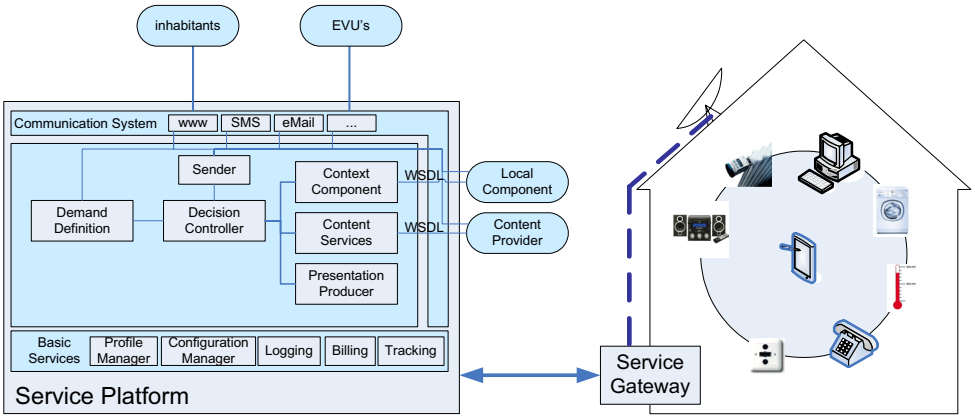


Figure 2: Service platform for it-supported value added services

General service engineering process models does not guarantee the success of an offered service [8][9], but it helps to feed customers expectations. Value added services in general should be customer orientated. The capturing of customers expectations and experiences need to be proved by methods for a successful service developing. The gained expectations and experiences must be transferred into a blueprint of the new service and take part of the service design and service management decision.

4. Service Design

Something special about service processes in the application domain smart living is the use of local components within the service process. In the context of service design and service blueprint normally the customer will initiate activities or will participate in a service process. In smart living szenarios local components will take a part of customers' activities. The local components produce information and additionally information will be sent to local components. How can the method service blueprint be extended to enable it to model these facts? To answer this question is necessary to raise the quality of service processes in the application domain smart living.

Service blueprint is a process analysis method, based on flow chart to systemize the description, documentation and analysis of service processes [8][10][11]. The blueprint structures the process activities on either side – customer and provider. Shostack [10] defined the service blueprint with three activity levels divided by the "line of visibility" and the "line of interaction". The first increment, amplified by Kingman-Brundage, contains more activity levels. It differs between customer activities, onstage activities, backstage activities, support activities and management activities [12]. These different kinds of activities show the organised process of a service blueprint from customer to service provider including support and management. The second increment, processed by Kleinaltenkamp and Fließ, includes the "line of order penetration". It divides the management activity into preparation and facility activities [13][14]. The main part, the "line of visibility", didn't change through the increments of the service blueprinting. Figure 3 shows the different structures of increments in service blueprint.

Till now the service blueprint specific options allows integrating and visualisation different kind of information such like time, costs and resources into the model. An analysis reflects the important communication points to customers and shows the potential fail points. But there is no possibility to integrate automatic raised information from home automation components inside the building. This information are unaccounted during the service process and have to be bring into the it-supported value added service process afterwards.

Analyzing the collected information and placing them within the service processes to raise the quality of service to customers is the main focus of blueprinting

[10]. Different service process modelling methods can be used for analyzing customer expectations and designing customer services [8]. The "flow chart" for service design and management by Ramaswamy [11] adopts different modelling constructions such as decision branch and process owner. It models the steps of functions to deliver services to customers. The service process can although be modelled with the "Event-driven Process Chains" (EPC), which is an important aspect of the Architecture of Integrated Information Systems (Aris) [15].

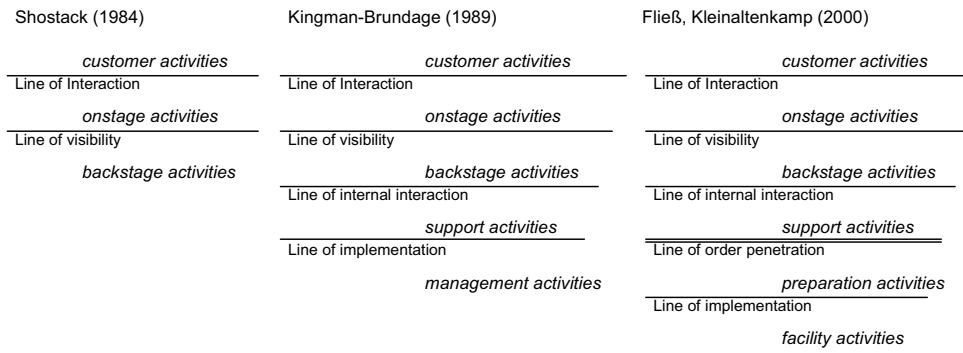


Figure 3: Increment overview of Service Blueprinting

The amount of information, available at customer service interactions, is one major part of the service delivery quality. Further information, collected from customer profiles and additional local components (e.g. thermostat, heating systems, and entry access), should be integrated in the service design and service management process. Situations and activities triggered from a customer and identified by home automation (e.g. leaving home, leakage, consumption) assists services like relocation service, security service, delivery service etc. Figure 5 shows the usage of service blueprint as an abstract example. With little modifications and extensions it allows dealing with automatically collected information from local sensors and actuators. Customers' activities can be swayed by local components and their individual profile.

Dealing with local components, directly or indirectly influenced by customers, a new line will be added to the structure of service blueprint (see Figure 4). The "line of crossbench interaction" separates the activities done directly by inhabitants from activities enforced indirectly by inhabitants (e.g. specific conduct at smart homes).

In smart homes inhabitants will interact more and more with local components. In preparation for value added services acting independent the need to distinguish between direct and indirect customer activities increases. Two objectives can be developed from the new structure of this blueprint. First it will identify local components needed for the value added service, because they are modelled as self-acting customer activities. Second it will obviously point out the potential of a service within a smart home. The potential of new innovative services in the environment of the product "living" will include a large quantity of providing a better service, a higher quality, lower prices or a combination of them assisted by local components.

An outpatient care caregiver is involved in this enfolding it-supported value added service. The service platform instantiate the value added service and coordinates the need of an inhabitant. The infrastructure for local components is established by the real estate company. Personalized information about the vital values and personal data of an inhabitant will be provided by the service platform. Inside the home different local components or smart wear will be used to collect various information like vitality (e.g. heartbeat or blood preasure) or behaviour (e.g. moving or lavatory use) of the inhabitant.

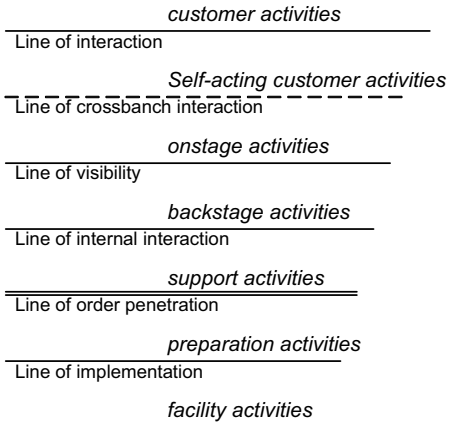


Figure 4: Extended service blueprinting

Smart wear products are vests or shirts with sensors or sensor controlled mattress. Especially sensor

controlled mattresses are qualified to prevent cod death. Finally the responsible caregiver determined different critical values for the inhabitant - his patient - and insert them into service platform. The collected values determine specific situations.

The service platform will act as an information logistic centre, where information is collected by local sensors and actuators. This information, dependent on context and situation, are handled by it-supported value added services. Afterwards information will be send through different communication channels (Internet, cellular phone network) to various end devices (fixed or mobile), in which the service platform takes different situations into account (present or absence of a person, reachability of a mobile end device, personal defined constraints).

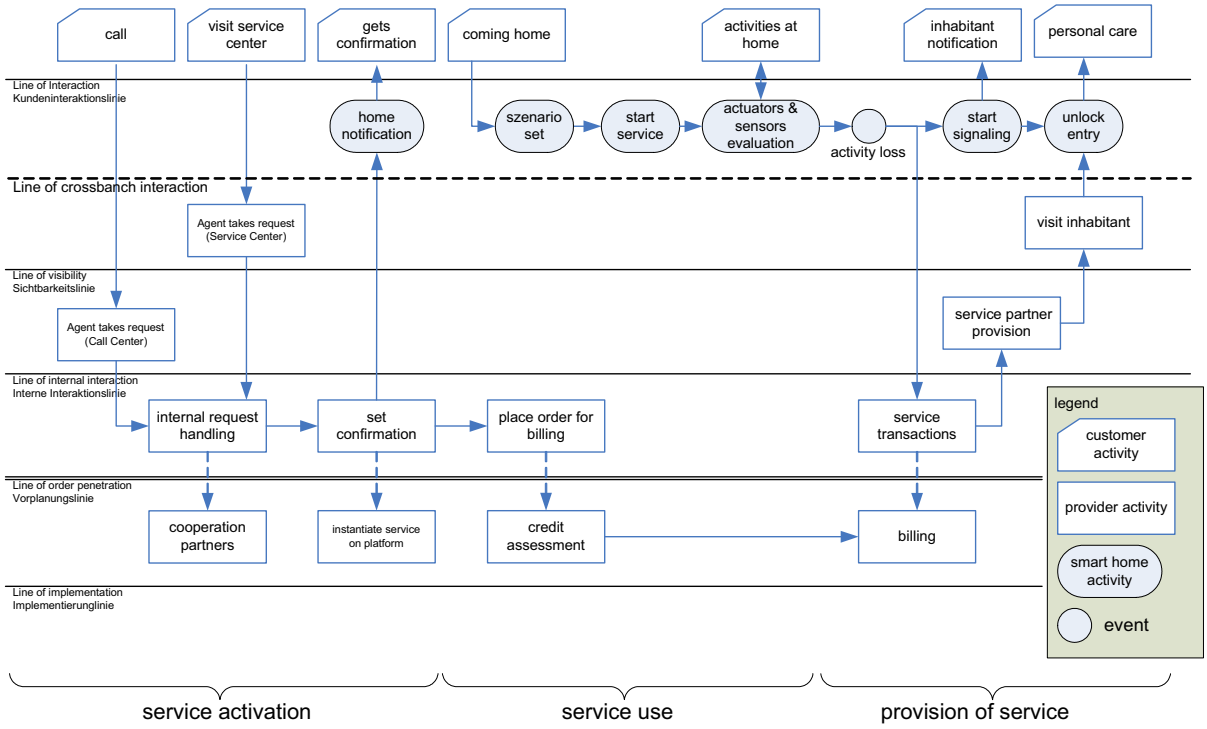


Figure 5: Service blueprinting with home automation based on care service example “VitalCheck”

If a critical value will reach, the service platform informs, depending of escalation breakpoints, the relatives, neighbors, friend, personal responsible or caregiver. The caregiver will get the notification from the service platform and now he can check, whether he should contact, visit the person or alert the emergency. His decision will depend on the information he has. The service platform provides him the actual and the developing of vital values, to confirm his decision. (see Figure 5)

In addition to the vital values it is possible to provide more information about the person. Maybe he was on a trip and activates the alarm. The emergency will correctly know where the person is, because the service platform locates the end device of the person.

5. Conclusion

Simple houses are extended by a service platform for the realization of it-supported value added services to smart homes. The local components are used to supply additional information for external service providers. Especially the multiple usages of local components to deliver information to different service providers are very important. Driven by changing demands of inhabitants, real estate companies extend their offers with additional services. To manage a significant number of services a strongly flexible architecture and an easy entry for service providers is needed. The flexible architecture

increases the offer of it-supported value added services for the service platform.

Service blueprinting as a part of the service engineering process clarified the usage to design service processes. The extension of service blueprinting showed the impact of developing new services especially in the application domain smart living. Further research will be focus on developing an adequate notation including different local components according to new value added services from different domains.

Furthermore the interfaces between local componentes, service gateway, service platform and service providers has to be defined for a whole service chain. The home automation provides a standardisation settled by the Digital Living Network Alliance [16]. For provider interaction the service process can be descripted and automaticaly supported for example by eFlow [17] or openXchange [18]. The bridge from home automation to service provider will a main focus for further research.

6. References

[1] Sustainable Homeservices „Benchmarking Sustainable Services for the Housing Sector in the City of Tomorrow“, http://www.sustainable-homeservices.com/pdfs-downloads/FlyerENGLISH.pdf, 04.01.2006.

[2] Weinzierl, T.; Schneider, F.: Gebäudesystemtechnik. In: Das intelligente Haus Wohnen und Arbeiten mit zukunftsweisender Technik. Hrsg: Tränkler, H.-R.; Schneider, F., Pflaum Verlag, 2001, pp. 349-361.

[3] Broy, M.; et. al.: Integrierte Gebäudesysteme – Technologien, Sicherheit und Märkte. SecuMedia Verlag, Ingelheim, 2002.

[4] Tränkler, H.-R.: Zukunftsmarkt Intelligentes Haus. In: Das intelligente Haus Wohnen und Arbeiten mit zukunftsweisender Technik. Hrsg: Tränkler, H.-R.; Schneider, F., Pflaum Verlag, 2001, pp. 17-35.

[5] Vossen, G.; et. al.: Vernetzung und Netzwerk in privaten Haushalten. Informationssysteme Report, Leonardo Computing GmbH, 2004.

[6] Schneider, F.: Heimautomatisierung: Anforderungen, technischer Stand und Trends. In: Das intelligente Haus Wohnen und Arbeiten mit zukunftsweisender Technik. Hrsg: Tränkler, H.-R.; Schneider, F., Pflaum Verlag, 2001, pp. 59-79.

[7] SmarterWohnenNRW http://www.smarterwohnen.net.

[8] Kim, H.-W.; Kim, Y.-G.: Rationalizing the customer service process. In: Business Process Management Journal, Volume 7 Number 2 2001, pp. 139-156.

[9] Pires, G.; Stanton, P., Stanton, J.: The Role of Customer Experiences in the Development of Service Blueprints.

[10] Shostack, G. L.: Designing services that deliver. In: Harvard Business Review, Volume 62 Number 1 1984, pp. 133-139.

[11] Ramaswamy, R.: Design and Management of Service Process, Addison-Wesley Publishing Company, Reading, Ma., 1996.

[12] Kingman-Brundage, J.: The ABC’s of Service System Blueprinting, In: Designing a Winning Service Strategy, Hrsg: Bitner, M. J.; Crosby, L. A., Chicago 1989, pp 30-33.

[13] Fließ, S.; Kleinaltenkamp, M.: Blueprinting the Service Company: Managing Service Process Effenciently and Effectively, In: Paper presented at the 10th Workshop on Quality Management in Services, Aston University, Birmingham (UK), May 17-19, 2000.

[14] http://www.serviceblueprint.de.

[15] Scheer, A. W.: ARIS-Toolset: Von Forschungs-Prototypen zum Produkt, In: Informatik-Spektrum 19 (1996), Springer-Verlag, pp 71-78.

[16] Digital Living Network Alliance, http://www.dlna.org

[17] Casati, F.; Ilnicki, S.; Jin,L.; Krishnamoorthy, V.; Shan, M.: "eFlow: a Platform for Developing and Managing Composite e-Services"; Software Technology Laboratory HP Laboratories Palo Alto; HPL-2000-36; März, 2000.

[18] openXchange, http://www.openxchange.org, IST-2000-28548.

Towards Context-based Mediation for Semantic Web Services Composition

Michael Mrissa, Chirine Ghedira and Djamal Benslimane
Université Claude Bernard Lyon 1
Lyon, France
{firstname.surname}@liris.cnrs.fr

Zakaria Maamar
Zayed University
Dubai, United Arab Emirates
zakaria.maamar@zu.ac.ae

Abstract

Web services composition is a keystone towards the development of interoperable systems. However, despite several efforts for explicit semantic description, Web services still face data-heterogeneity challenges. Correct interpretation of data exchanged between composed Web services is hampered by different implicit modeling assumptions and representations. In this paper, we demonstrate the importance of context to facilitate data exchange, and describe a context representation model for Web services. Then, we present a context-based mediation architecture that helps performing meaningful composition.

1. Introduction

In the domain of service-oriented computing, Web services[1] are now accepted as a standard means for enabling composition scenarios, and their capacity for loosely-coupled interactions allows answering to complex user needs. Despite these advantages, the standard Web services protocol-stack[2] was not initially planned for satisfying the requirements of semantic interoperability. Recent languages and frameworks such as OWL-S [12], the Web Services Modeling Ontology WSMO [1] or WSDL-S [13], explicitly describe the semantics of Web services, then referred to as semantic Web services [7]. These initiatives use ontologies[3] as shared vocabularies to facilitate semantic reconciliation.

However, such approaches do not take the context of information into consideration. The term context is defined in the rest of this paper as the collection of implicit assumptions that are required in order to perform accurate data interpretation. A semantic concept should be interpreted differently, depending on the context it relates to. Consequently, semantic description and interpretation mechanisms need consider context in order to interpret information efficiently. In the domain of Web services composition, context interpretation is generally buried in code, and its explicit descriptions are inexistent. Changes in context are dealt with manually, which requires lots of time and handmade work, and reduces availability and reliability of information systems.

In this paper, we provide a framework for explicitly describing and managing context in order to enable meaningful data exchange between Web services that participate in a composition. Our contribution consists of 1) a proposition of context representation model for Web services, followed by 2) a method for annotating WSDL input and output messages with context, and 3) a service-based mechanism for context-aware mediation in a composition. This paper is organized as follows. In Sect. 2, we show, with a working example, the value added by managing the context of data in Web service composition. In Sect. 3, we present the notion of *semantic object*, introduced in [2], that supports our work, prior to showing how we adapt the associated model to the domain of Web services. In Sect. 4, we detail our annotation for integrating context representation into the Web service protocol stack, and present a context and service-based mediation architecture for Web services composition. In Sect. 5 we overview related work on mediation and semantics for Web services, and context representation. Finally, Sect. 6 concludes the paper and sets guidelines for future work.

2. Motivating Example

We demonstrate, with a simple booking example, how context changes the interpretation of information that flows between Web services. The example concerns a trip to Japan. A hotel, which has attractive rates, provides a Web service endpoint described with the usual WSDL file, for booking rate estimate. In order to estimate the affordability

[1] A Web Service is a software component that is described and being accessed via standard XML-based protocols.

[2] Simple Object Access Protocol SOAP [3], Web Service Description Language WSDL [5] and Universal Description, Discovery and Integration protocol UDDI [16].

[3] An ontology is a shared description of a domain knowledge [11].

of this hotel for a European passenger, the following composition of Web services occurs: *hotel booking* WS_1 to calculate charges based on the number of booked nights, and *banking* WS_2 to manage payment.

From a technical perspective, WS_1 sends a parameter[4] named "price_yen", and WS_2 receives a parameter named "price_euros". Although it is possible to use different type systems, we consider, for illustration purposes, both parameters are described in XML Schema type system [17], and are of type "double". This information, obtained from WSDL description, shows low level data compatibility between Web services.

Also, these parameters bind to particular semantics. WS_1 delivers a value that should be interpreted as a figure in Yens, whereas WS_2 expects a value that it will interpret as a figure in Euros. We assume both parameters bind to the same "Price" semantic concept, described in a common ontology. Existing approaches to semantic description and mediation of Web services, overviewed in Sect. 5, explicitly describe this information, to allow inferring a relation between these parameters, and performing appropriate conversion. Such approaches use ontologies as references to solve structural and semantic heterogeneities.

Now, let us inject context into these parameters. WS_1 binds to a "Japanese Hotel Booking" context, in which charges have a scale factor of 1000, prices do not include Value-Added Tax (VAT), dates required for conversion rates are in Japanese format (yyyy.mm.dd); and WS_2 binds to a "French Banking" context, where charges have a scale factor of 1, prices include Value-Added Tax, dates required for conversion rates are in French format (dd.mm.yyyy). This example shows that heterogeneous contexts exist too, so an agreement on the interpretation of values must be reached, by reconciling context descriptions. Context information is related to the local and implicit assumptions on the interpretation of data, and is not described in the domain ontology. Usually, in a semantic composition, context heterogeneities

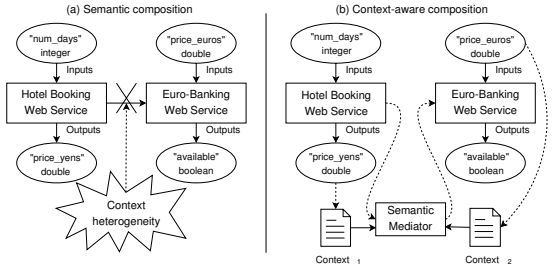


Figure 1. Semantic vs. Context-aware composition

[4]We refer to parameter as the parts of messages described in WSDL

are resolved in an ad-hoc way at the level of the receiver Web-service, if at all, as presented in Fig. 1-(a). This reduces adaptability of Web services as parts of composition, and furthermore, it gives them the responsibility of solving context heterogeneities. To conduct context-aware composition, the context of data must be explicitly described, and a mediation mechanism must handle data flow, as shown in Fig. 1-(b). As standard semantic Web services do not handle such representation, we hence propose an annotation of WSDL to explicitly describe context of data, on the basis of the following model.

3. Context Representation

In this section, we study a model for describing the semantics of semi-structured data [2]. Then, we adapt this model to facilitate context description for data exchange in Web service composition.

3.1. Definition of a Semantic Object

Based on Sciore et al.'s notion of *semantic value* [14], Bornhövd defines a model for describing the underlying semantics of semi-structured data, with the concept of *semantic object* [2].

An ontology Φ is defined as a vocabulary for the notation and representation of a given application domain, and formalized as a finite set of concepts:

$$\Phi = \{C_1, \ldots, C_n\}, n \in \mathbb{N} \ .$$

Each concept C_i has a physical representation type $RepType(C_i)$ associated with it, to which the domain $Dom(RepType(C_i))$ specifies the possible values for the representation of data corresponding to this concept. Also, a semantic object $SemObj$ is defined as a triple:

$$SemObj =< C, v, \$ >,$$

Where $C \in \Phi$ denotes the ontology concept to which the semantic object adheres, and the value $v \in Dom(RepType(C))$ is the physical representation of v according to the physical representation of type C. $\$$ specifies the context of $SemObj$, and is defined as:

$$\$ = \{< C_1, v_1, \$_1 >, \ldots, < C_k, v_k, \$_k >\}, k \in \mathbb{N} \ .$$

where $< C_k, v_k, \$_k >, 1 \leq i \leq k$, are semantic objects, referred to as "modifiers" in this paper, that describe different semantic aspects of $SemObj$. In turn, modifiers may provide additional context information in $\$_k$. Are also introduced the notion of complex semantic object, the means for semantic conversion, comparability and equivalence of semantic objects, that build the basis of this framework.

3.2. A Context Model for Web services

The model described previously provides a sound basis for describing messages parts of Web services as semantic objects, its main advantage being the combination of intensional and extensional descriptions. The concept of a semantic object is intensionally described in a domain ontology, while context is extensionally described under the form of additional attributes, preventing an excessive complexity of the domain ontology.

Bornhövd's model was designed for handling semi-structured data in a large sense. However, in the domain of semantic Web services, a major design choice to semantic description is the separation of the grounding and abstract views on data. Generally, the grounding representation of data follows the XML Schema type system, while its abstract part is described in some ontology language. Such separation of concerns requires an explicit description of the physical representation of semantic objects. Therefore, we deem appropriate to include the physical representation in the definition of semantic object, as follows:

$$SemObj =< C, v, t, \$ >,$$

where $C \in \Phi$ denotes the ontology concept to which the semantic object $SemObj$ adheres, the value $v \in Dom(t)$ is the physical representation of v according to t, that specifies the type of v defined in a specific type system, and $ specifies the context of $SemObj$. Such definition does not modify previous work on the model, however, it clarifies the definition of data type, which is now clearly distinguished from the conceptual reference C of the semantic object. Then, existing mediation approaches presented in Sect. 5, that also deal with low level descriptions of data types, can be combined to the context-based mediation architecture presented in the following. In the example of Sect. 2, a possible semantic object would be:

<ns:Price, 55.00, xsd:double, Context>

where ns:Price is a qualified name referring to the concept of price described in a domain ontology, 55.00 is the value sent by the Web service, xsd:double is the type in XML Schema, and Context is a list of semantic objects that describe implicit semantic aspects such as VAT rate and scale factor.

Also, Bornhövd adds the vocabulary for describing modifiers to domain ontologies, while stating that a context description is always a subset of all the meaningful aspects of an ontology concept, which are potentially infinite. However, Web service providers should be free to decide which subset of modifiers is relevant to their application. In which case, the size of domain ontologies would grow depending on providers' needs for context description. To overcome this problem, we deem appropriate to separate context ontologies from domain ontologies, so that they do not surcharge the latter. Context ontologies should describe all the modifiers Web service providers associate to a concept.

As context ontologies specify structural representation and meaning of terms, we need to define how to extensionally describe context values for Web services. To do so, we define two categories of modifiers. Static modifiers have to be (statically) specified to clarify the meaning of data, and dynamic modifiers can be (dynamically) determined from other parts of the semantic object. We add static modifiers to the WSDL, so that our approach is compliant with the standard Web services protocol stack. We do not need to specify dynamic modifiers, as they can be inferred from static ones. In the next section, we present a solution for annotating descriptions of composed Web services, in order to make context information available at the execution stage of composition, before describing a rule-based solution for context mediation in a composition.

4. Context Management

4.1. Annotating WSDL with Context

Based on the model developed previously, we enrich the description of Web services with context information. Using the WSDL extensible elements, we annotate WSDL message parts, so that they can be described as *semantic objects*. WSDL depicts how to access a Web service and what operations it performs. Access is subject to specific communication protocols, and operations use input and output arguments that have a number, an order, and a type. To keep the paper self-contained, we overview a simplified structure of the WSDL metamodel in Fig. 2.

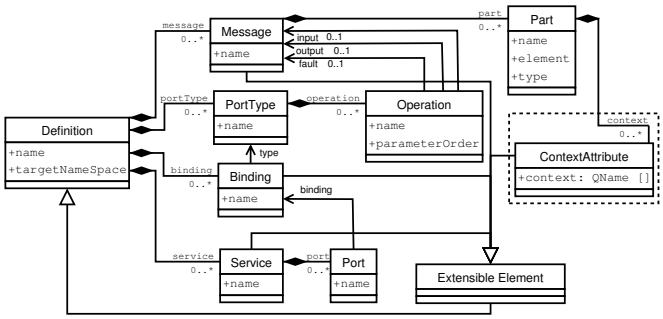


Figure 2. Context in WSDL metamodel

Our annotation takes advantage of the extensibility elements proposed in the WSDL [5], so annotated descriptions can operate seamlessly with classical or annotation-aware clients. As `<message>` elements are composed of one or several parts, we annotate `<part>` elements with a `context` attribute that contains the values of static modifiers. Each `context` attribute contains a list of qualified names. The first qualified name of the list specifies the ontology concept of the value (C). Following elements refer

to instances (called individuals in the OWL vocabulary) of static modifiers described in a context ontology. Thus, with our annotation, a value v, and its data type t described in WSDL are enhanced with the concept C and the modifiers necessary to define the context \$, thus forming a semantic object $< C, v, t, \$ >$ as defined previously. A sample annotated WSDL file is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://localhost:8080/axis/EuroBanking.jws"
 ...
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:ctxt="http://www710.univ-lyon1.fr/ mmrissa/context/context.xsd"
 xmlns:ctxt1="http://domain.ontology.org/Price.owl"
 xmlns:ctxt2="http://context.ontology.org/context/PriceContext.owl#">
 ...
  <wsdl:message name="checkPriceRequest">
   <wsdl:part name="price" type="xsd:double" ctxt:context="ctxt1:Price
ctxt2:France ctxt2:VATIncluded ctxt2:ScaleFactorOne"/>
</wsdl:message>
 ...
  <wsdl:portType name="EuroBanking">
   <wsdl:operation name="checkPrice" parameterOrder="price">
   <wsdl:input name="checkPriceRequest" message="impl:checkPriceRequest"/>
   <wsdl:output name="checkPriceResponse"
message="impl:checkPriceResponse"/>
  </wsdl:operation>
</wsdl:portType>
 ...
</wsdl:definitions>
```

To obtain a complete context \$, values of dynamic modifiers are inferred at runtime by rule-based mechanisms. This comes with several advantages: rules are easily modifiable, so our architecture is more adaptable to changes in the underlying semantics. Also, often-changing modifiers could not be statically stored, so using rules simplifies the annotation to WSDL. Furthermore, rules separate application logic from the rest of the system, so updating rules does not require rewriting application code. In the following, we detail our context mediation architecture, that integrates into composition as a Web service, and show its interactions with a rule-based component (a rule engine).

4.2. Context Mediation for Web Services

As a first step, we examine how context management capabilities should be granted to composition scenarios. Based on previous works (Sect. 5), we follow a decoupled approach, and we encapsulate context mediation functionalities into a Web service front-end. This solution presents two main advantages. Firstly, the mediation Web service can be triggered, via its WSDL interface, so it remains independent from composition languages and engines. Secondly, it is straightforward to handle context by invoking the mediation Web service between every two composed Web services. The main problematic aspect of this solution is its limited scope, as data flow is bound to data types specified in WSDL descriptions. However, it is possible to generate adapted WSDL interfaces at design time.

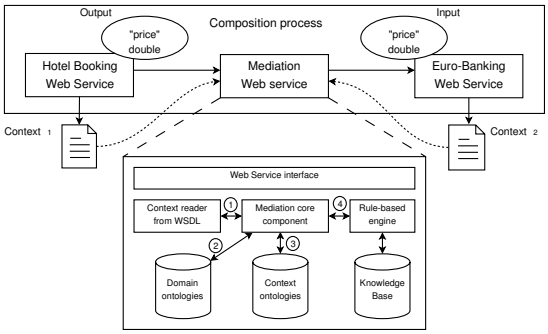


Figure 3. Functioning of the mediation Web service in the sample composition

Now, let us detail the operation of the mediation Web service. Its role is to convert data from the context of the Web service it originates (source context), to the context of the Web service it is being sent to (target context). For each message part, the role of the mediation Web service consists in 1) building source and target contexts using WSDL annotation, ontologies and rules; 2) examining differences between contexts and performing data conversion to target context, or generating an error message, and 3) sending results to the appropriate target. As Fig. 3 shows, the mediation service communicates with four components: a context reader to extract context extensibility attributes from WSDL descriptions, context and domain ontologies to identify concepts, and a rule engine to determine values of dynamic modifiers and to perform the conversion.

For each message part concerned with the mediation process, the mediation Web service performs the following steps. First step, it generates an in-memory model of WSDL descriptions, extracts context annotations and builds contexts of the message part. Second step, it identifies the first qualified name of the annotation as the ontology concept of the parameter. Then, it verifies that the concepts of both annotations match. Third step, the mediation Web service extracts contexts from the context ontologies referred to in the WSDL annotation. Contexts of semantic objects are the sets of subsuming semantic objects described in the context ontology. Additional annotation attributes help specify the values of static modifiers. Fourth step consists of two stages. Stage one, the rule engine infers the values of dynamic modifiers from the values of static modifiers of the annotation. Stage two, for each modifier, the rule engine applies appropriate conversion to the parameter, in order to convert the value of the modifier to the target context. If an error happens at some point of the process, an exception is thrown, and the mediation Web service sends a failure message. If the mediation process was performed correctly, data is converted to the target context and transmitted to the

next Web service.

A prototype has been fully developed as a proof-of-concept to demonstrate the feasibility of this architecture under the Java^{TM} NetBeans environment. We developed a graphical user interface to read and write context annotations from and to WSDL files. This tool enables providers or advanced users to annotate WSDL files with context, so that it becomes possible to compose them with context-aware mediation Web services. We also developed a mediation Web service, that reads context annotation from WSDL files and converts data received from its source context to a target context. Our implementation performs dynamic and accurate context interpretation, and enables meaningful execution of composition. To illustrate our proposal, we implemented the example of Sect. 2. With annotated WSDL and the mediation Web service, not only the "Price" concepts match, but data is transformed at-runtime, to comply with the different scale factors, heterogeneous date formats that allow getting up-to-date conversion rates between currencies, and different VAT rates that also are not always included in the price.

Our current composition example runs into a BPWS4J (http://www.alphaworks.ibm.com/tech/bpws4j) composition engine hosted in a Apache Axis container (http://ws.apache.org/axis). We use Jena 2 (http://jena.sourceforge.net/) API and a Drools (http://www.drools.org/) rule engine, to access and manipulate OWL ontologies, infer values that modifiers should take and perform data conversion at runtime. Our prototype includes domain and context ontologies designed with Protégé (http://protege.stanford.edu/) for describing the "Price" concept and context[5].

5. Related Work

This section presents different initiatives that relate to the semantic and mediation aspects of Web services, and to previous work on context description. These related works helped us build ideas, and backed our approach as they are important references of the domain.

5.1. Semantics for Web services

Semantic Web services constitute an active domain of research. Most approaches rely on ontologies to express the semantics of a domain, however, there are several ways of inserting semantics into Web Services. One way involves using description languages like OWL-S [12], and another way consists in extending syntactic standards like WSDL with semantic features (WSDL-S) [13].

[5] Available at http://www710.univ-lyon1.fr/∼mmrissa/price.owl and http://www710.univ-lyon1.fr/∼mmrissa/PriceContext.owl

OWL-S [12] is a subset of the OWL (ex-DAML) ontology language. It is a general ontology that provides support for building semantic Web services. OWL-S was designed to be coupled with standard description formats like WSDL. It consists of three ontologies, namely *process model*, *service profile* and *grounding*, that respectively describe *"how the service works"*, *"what the service does"* and *"how to access the service"*. Inspired from OWL-S, several research projects have been developed, such as ODESWS [6] that models Web services using problem-solving methods.

From the DERI laboratory, WSMO [1] is a formal language and ontology that describes varied aspects of semantic Web services. It supports the development and description of semantic Web services with the WSMF [7] conceptual model, that enables mediation as a service, so that it allows maximal decoupling between component Web services.

With WSDL-S, Miller et al. annotate WSDL with several extensions related to operations and messages [13]. These extensions refer to concepts of domain models to specify semantics of messages, but also preconditions and effects of operations.

5.2. Mediation between Web services

Mediation between Web services is a hot topic and receives a lot of attention from the research community. Many mediation approaches rely on the concept of mediator for solving data heterogeneities between participants of an interaction.

Cabral and Domingue [4] provide a broker-based mediation framework for composing semantic Web services. Their approach follows WSMO conceptual framework [1] that recommends using strongly decoupled and service-based mediation capacities. The mediator component is a key part of their architecture and it mediates concepts between different ontologies.Williams et al. [18] use agents to perform semantic mediation between input and output parameters of Web services by encapsulating the composition into an agent, that controls the developpement of the operation. Spencer et al. [15] present a rule-based approach to semantically match outputs and inputs of Web services. An inference engine analyzes OWL-S descriptions and generates multiple data transformation rules using a description-logic reasoning system.

5.3. Approaches to Context Description

In many works, context refers to the interactions with the surrounding environment. In this paper we follow another definition and only detail similar work for space limitation purpose. In the domain of database interoperability, the Context Interchange approach [9] provides formalisms

for context representation, based on the notion of semantic value. It has proved to be a highly scalable, extensible and adaptable approach to semantic reconciliation of data, and was first introduced by Sciore et al. [14]. Goh [10] and Firat [8] presented implementations and extensions to this approach. Then, Bornhövd [2] adapted this model to the description of semi-structured data. We use this latter work as a basis to our approach.

While mediation and semantic description of Web services in a composition are very active research fields, to the best of our knowledge, none of these works actually consider the use of explicit context description to solve heterogeneities of type value in Web services composition.

6. Conclusion

In this paper we highlighted the importance of context in Web services composition, and proposed an architecture for handling context, in order to obtain meaningful composition scenarios. We first demonstrate, with an illustrative example, that context is required for a correct interpretation of data exchanged between Web services. Then, we provide a solution for describing and handling such information, using context ontologies and rule-based mechanisms. This proposal includes a model for context description, coupled with a method for annotating WSDL parameters, propelling them to the level of *semantic objects*, while respecting WSDL extensibility elements; and a service-based context mediation architecture.

Future improvements of context-based mediation in Web service composition targets three aspects: first, the study of methods for enabling the integration of the mediation Web service into dynamic composition; second, the addition of capabilities to the mediation mechanism, like the ability to compose Web services specialized in conversion and to integrate them in the mediation process; and third the extension of this architecture to Web service discovery and selection stages, that could consider the possibilities of context mediation as a selection criteria in the discovery step.

References

[1] S. Arroyo and M. Stollberg. WSMO Primer. WSMO Deliverable D3.1, DERI Working Draft. Technical report, WSMO, 2004. http://www.wsmo.org/2004/d3/d3.1/.

[2] C. Bornhovd. Semantic metadata for the integration of web-based data for electronic commerce. In *Int'l Workshop on E-Commerce and Web-based Information Systems (WECWIS), Santa Clara, CA*, pages 137–145, 1999.

[3] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple object access protocol (SOAP) 1.1. Technical report, The World Wide Web Consortium (W3C), 2000.

[4] L. Cabral and J. Domingue. Mediation of semantic web services in irs-iii. In *First Int'l Workshop on Mediation in Semantic Web Services (MEDIATE 2005), Amsterdam, The Netherlands*, December 12th 2005.

[5] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1, W3C Note. Technical report, The World Wide Web Consortium (W3C), March 2001.

[6] O. Corcho, A. Gomez-Perez, M. Fernandez-Lopez, and M. Lama. Ode-sws: A semantic web service development environment. In I. F. Cruz, V. Kashyap, S. Decker, and R. Eckstein, editors, *SWDB*, pages 203–216, 2003.

[7] D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. Technical report, Vrije Universiteit Amsterdam, 2002.

[8] A. Firat. *Information Integration Using Contextual Knowledge and Ontology Merging*. PhD thesis, Massachusetts Institute of Technology, Sloan School of Management, 2003.

[9] C. H. Goh, S. Bressan, S. Madnick, and M. Siegel. Context interchange: new features and formalisms for the intelligent integration of information. *ACM Trans. Inf. Syst.*, 17(3):270–293, 1999.

[10] C. H. Goh, S. Bressan, S. E. Madnick, and M. Siegel. Context interchange: New features and formalisms for the intelligent integration of information. *ACM Trans. Inf. Syst.*, 17(3):270–293, 1999.

[11] T. Gruber. What is an ontology? http://www-ksl.stanford.edu/kst/what-is-an-ontology.html, 2000.

[12] D. L. Martin, M. Paolucci, S. A. McIlraith, M. H. Burstein, D. V. McDermott, D. L. McGuinness, B. Parsia, T. R. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. P. Sycara. Bringing Semantics to Web Services: The OWL-S Approach. In J. Cardoso and A. P. Sheth, editors, *SWSWPC*, volume 3387 of *Lecture Notes in Computer Science*, pages 26–42. Springer, 2004.

[13] J. Miller, K. Verma, P. Rajasekaran, A. Sheth, R. Aggarwal, and K. Sivashanmugam. WSDL-S: Adding Semantics to WSDL - White Paper. Technical report, Large Scale Distributed Information Systems, 2004. http://lsdis.cs.uga.edu/library/download/wsdl-s.pdf.

[14] E. Sciore, M. Siegel, and A. Rosenthal. Using semantic values to facilitate interoperability among heterogeneous information systems. *ACM Trans. Database Syst.*, 19(2):254–290, 1994.

[15] B. Spencer and S. Liu. Inferring data transformation rules to integrate semantic web services. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 456–470. Springer, 2004.

[16] UDDI. *Universal Description, Discovery, and Integration of Business for the Web*, Oct. 2001. URL: http://www.uddi.org.

[17] W3C. XML Schema Part 2: Datatypes Second Edition. Technical report, W3C, October 2004. http://www.w3.org/TR/xmlschema-2/.

[18] A. B. Williams, A. Padmanabhan, and M. B. Blake. Experimentation with local consensus ontologies with implications for automated service composition. *IEEE Trans. Knowl. Data Eng.*, 17(7):969–981, 2005.

The Research and Design of Layered-metadata used for Component-based Software Testing

Ma Liangli
(Computer Science & Technique College, Hua Zhong University of Science and Technology,Wuhan,Hubei,China,430074)
(Computer Engineering Department , Naval Engineering University, Wuhan,Hubei, China,430033)
maliangl@163.com
Lu Yansheng
(Computer Science & Technique College, Hua Zhong University of Science and Technology,Wuhan,Hubei, China,430074)
Liu Mengren
(Computer Engineering Department , Naval Engineering University, Wuhan,Hubei, China,430033)

Abstract

In the context of component-based development and testing, metadata provides a mechanism in which assumptions about components may be communicated to both developers and testers. This paper describes a refined view of metadata in which metadata is partitioned into three layers: a metadata model layer, an application layer, and a data layer. Both UML and XML representations of the layered metadata model are described. The paper concludes with a brief discussion of results obtained when applying this method to test a GIS application developed for a naval shipyard.

Keywords: Component, Component-based integration testing, metadata, UML,XML

1. Introduction

Interest in component-based systems continues to grow as a result of the potential they offer in managing software complexity. Although a component-based development method has many advantages, serious drawbacks also exist impacting a wide range of software engineering activities.

A core set of drawbacks center on problems relating to quality assurance. Although components are typically verified before being integrated into an application, quality related problems still exist. As a result, techniques that can improve the quality of component-based systems, such as testing, become a hot topic of research.

Empirical evidence suggests that in many situations, it is the lack of information about externally-provided components that is the primary source of quality related concerns. In response to these concerns, Orso et al.[1] developed a method, in which *metadata* is used to communicate information between component developers and component users. Metadata, embedded in the component, is used to describe various static and dynamic aspects of the component. This embedded metadata facilitates a number of component-centric activities including maintenance and comprehension. Perhaps more importantly however, metadata can be used as the basis for automated component-based testing.

Stafford and Wolf[2] find the input path of a component can influence the output directly, and develop a methodology supporting these concepts. Whaley et al. [3] propose a framework in which models are used to express acceptable sequences of method calls. In this manner, the customers can evaluate their use and check whether a particular sequence of calls is indeed permitted. Wu et al. [4] propose an approach in which metadata is expressed in terms of UML models targeting the addition of specific test elements of software components by the providers. They define a set of adequacy testing criteria in which the UML interaction diagrams are used for extracting the test elements. The proposed UML test model provides a comprehensive technique for functional testing of third party components designed for situations in which inspection of the component source code is not possible.

An approach to metadata-based testing inspired by the design by contract principle[6] has been proposed by Brucker and Wolff[5]. In this approach, an attempt is made to generate components that have the ability of testing themselves. This is achieved by building some

testing code into the component whose execution tests the component.

In this paper, the idea of layered-metadata is introduced. We describe its model and architecture and present an approach for creating encapsulated metadata schemes that adopt object-oriented modeling techniques and embed metadata as encapsulated items within the data itself. There are three layers, separately naming metadata-model layer, application-model layer and data layer. The layered-metadata is divided into descriptive layered-metadata and operational layered-metadata. Then we describe respectively the content of descriptive layered-metadata and operational layered-metadata and give the UML representation of a given layered-metadata based on an example of a component named *RUNENVIRONMENT* describing the hardware environment. Furthermore, we give the XML representation of layered-metadata, and combined with above example describes the relation between layered-metadata information and XML document. At the end of this paper, we present the results of the method applied to a GIS application developed for a navy shipyard.

2. Layered-metadata Description

In general**,** components are developed for use in a wide variety of contexts. In the context of this discussion, we assume that the following five types of information can be associated with a component:

(1) Information to evaluate the component; for example, information describing static and dynamic metrics that have been computed for the component.

(2) Information to deploy the component; for example, additional information about the interface of the component.

(3) Information to test and debug the component; for example, finite-state machine representations of the component, regression test suites together with coverage data, and information about dependences between inputs and outputs.

(4) Information to analyze the component; for example, summary data-flow information, control-flow graph representation of part of the component, and control-dependence information.

(5) Information on how to customize or extend the component; for example, a list of the properties of the component, a set of constraints on their values and the methods to be used to modify them.

2.1 Layered-Metadata

In this paper, a solution is presented to create encapsulated metadata schemes that adopt object-oriented modeling techniques and embed metadata as encapsulated items within the data itself.

The design of our layered-metadata focuses on cross-platform settings, remote procedure calls, application operation requirements and the registry of component functionality etc.

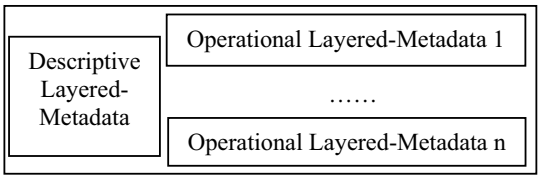


Figure 1: The Metadata Object Model

Figure 1 illustrates a possible framework for object-oriented metadata objects, which include two types of metadata components: descriptive metadata and operational metadata. Descriptive metadata includes a general description of component attributes, which is interpreted by component users, such as data descriptions, distributed information, and metadata reference, etc.

Operational metadata contains machine-executable information which can be applied automatically when a component is integrated into an application environment. This may facilitate dynamic interactions, integration, and implementation-specific interactions among platforms.

2.2 The architecture of Layered-metadata

According to the requirements of component users and component developers, we have developed a layered-metadata model consisting of three layers: a metadata-model layer, an application layer, and a data layer. The relationships between these layers are shown in Figure 2.

The highest layer of our model is the metadata schema language, which is used to describe a conceptual metadata schema as well as an application schema at the application model layer. The metadata schema also provides the metadata element definitions for metadata dataset. A metadata dataset describes the administration, organization and content of a dataset at the data level. The architecture provides a language-based implementation framework for metadata structure and encoding.

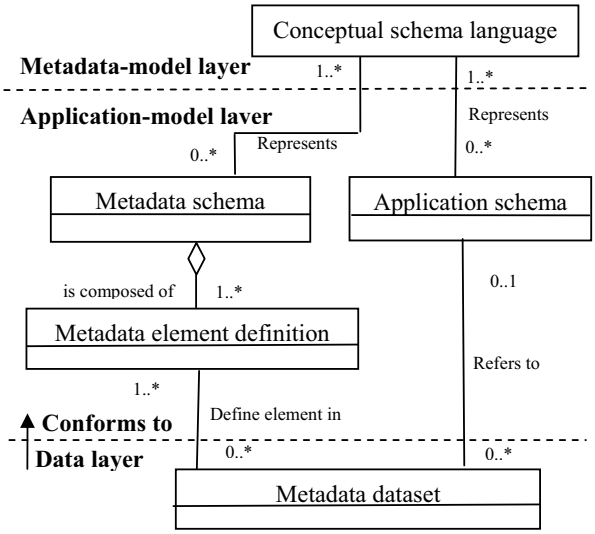


Figure 2: The architecture of a layered-metadata

3. Layered-metadata Design

3.1 Operational Layered-metadata design

The design of operational layered-metadata for general software components needs to consider what kinds of operations or tasks are associated with the related component. In general, four representatives tasks are illustrated here for the specification of software component layered-metadata contents, which include *data input requirements*, *data output specifications*, *run-time system requirement* and *component registration* (see Figure 3).

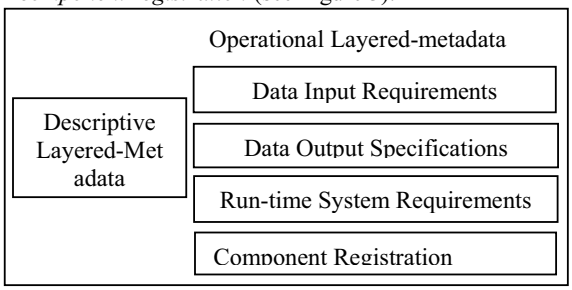


Figure 3: The design of Operational Layered-metadata

In Figure 3, data input requirements layered-metadata will specify the requirements of input data for operations. The contents of data input requirement layered-metadata will include the following items: Data input format; Data uncertainty threshold (the feasible range of data accuracy for this operation); Prerequisite (coordinate systems, projections, topology, etc.); Component category(network analysis or application modeling).

Data output specification layered-metadata focuses on the specification of output data objects generated by the components. These information items can be used by subsequent operations. The content of data output specification layered-metadata will include the following items: Data output format; Data uncertainty threshold; Operation effects (the change of data characteristic after this operation);

The design of run-time system requirement is to facilitate cross-platform components and applications. Different components may require a unique run-time environment, such as the local disk size, CPU speeds, and the size of RAM. The contents of run-time requirement will include the following items: Hardware requirements(CPU speed, temporary disk size, RAM, etc.); Virtual machine requirements(Java or Microsoft Virtual machine); Component profiles(component size, type and run-time efficiency).

The component registration layered-metadata will be used to register components via a universal registry or web service registration framework on the network. The contents of component registration layered-metadata will include the following items: Unique Component ID(for registration); Functionality classification(specific applied function).

3.2 Descriptive Layered-Metadata Design

In this paper, all descriptive layered-metadata content is described in terms of objects and parameters of those objects. For instance, object references may be used to describe a parent-child hierarchy among the objects in the layered-metadata. Also an object may be entirely contained in another object allowing hierarchical design decisions to be reflected in the layered-metadata content if desired. For each descriptive layered-metadata context, the allowed layered-metadata content is documented using whatever format is appropriate for the data, usually a combination of diagrams, tables and text. Here based on an example of a component named RUNENVIRONMENT to describe the hardware environment we have developed, which gives the relating content using UML, as shown in Figure 4.

In Figure 4, the layered-metadata objects are shown as classes with attributes. Object parameters are shown by using class attributes. Parameters that are references to other objects are represented with associations between classes. Parameter names for object references are given using UML role name for the association. Lists of object references are used when the association's multiplicity designates the possibility of more than one reference, and all such lists are ordered. An object contained in another object is shown with a composition relation.

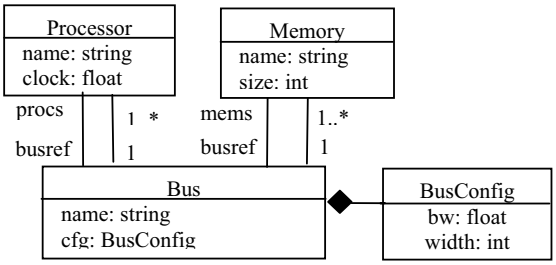


Figure 4: The UML representation of a given layered-metadata

3.3 Layered-metadata XML Representation

Because of the different application environment which component is integrated, it must allow the layered-metadata to be created and modified with standard text manipulation tools. So the layered-metadata presented in this paper is represented in XML format, which allows a component user to easily predict the XML form from the documentation, or to derive the layered-metadata context from the XML.

There is a top-level element, defined here as <layered_metadata>. An additional sub-element will be used to identify the layered-metadata context of component. Top-level layered-metadata objects appear as elements within this context element where the object element start tag corresponds to the object type name in the context documentation. Each object element has an attribute called *ident* with a string value that uniquely identifies the object and that may be used to reference this object. This attribute is associated with the *ID/IDREF* features of the XML Documentations Type Description(DTD) format and the XML Schema specification. Within an object element, each object parameters is described by using an element with a start tag that is the same as the parameter name. Scalar parameters values are given as text in the content of the element, and may be strings, integers, and floating point numbers. Object references are empty elements that have an attribute *objref* with a string value the same as that of the *ident* attribute of the object being referenced.

4.Conclusions

With the depth and development of component technology on the application, it is important how to implement effective testing when component is integrated into the application environment. Based on previous research work, in this paper, we present an idea of layered-metadata, design and implement its model and architecture. Furthermore, we give a layered-metadata example describing hardware environments. In order to verify the validity of above idea more objectively, we use above layered-metadata method to generate 15 test cases of above component RUNENVIRONMENT in the case of source code unavailability. Then we further analyze related source code in detail, and compare the results of test cases coverage between above method(simply called L-METADATA) and traditional test cases generation method (simply called NOMETA). The results are shown in Table 1:

Table 1 Test coverage percentage for test cases selected by the LAYERED-METADATA and the NOMETA

	NOMETA	L-METADATA
Statement coverage percentage	94.5%	95.6%
Branch coverage percentage	93.8%	94.7%
c-use	91.9%	93.1%
p-use	90.8%	91.9%

In the future, main research will focus on the improvement the interface format specification of metadata and the establishment of notification facilities etc.

5. References

[1] A. Orso, M. J. Harrold, Component Metadata for Software Engineering Tasks, Proceeding of EDO 2000, LNCS Vol. 1999,Springer.

[2]J.A.Stafford,A.L.Wolf, Annotating Components to Support Component-based Static Analyses of Software Systems, Proceeding of the Grace Hopper Celebration of Women in Computing,2001

[3]J.Whaley,M.C.Martin,M.S.Lam, Automatic Extraction of Object-oriented Component Interfaces, Proceeding of the International Symposium on Software Testing and analysis,Roma,Italy,2002.

[4] Y. Wu,,M. Chen, UML-based Integration Testing for component-based Software, In Proceeding of the 2nd International Conference on COTS-Based Software Systems(ICCBSS), February, Ottawa,Canada,2003

[5]A.D.Brucker,B.Wolff, Checking OCL Constraints in Distributed Systems Using J2EE/EJB, Technical Report 157,July, University of Freburg,2001.

[6] B. Meyer, Applying Design by Contract, IEEE Computer,25(10),October 2002.

QoSPL: A QoS-Driven Software Product Line Engineering Framework for Distributed Real-time and Embedded Systems

Shih-Hsi Liu[1], Barrett R. Bryant[1], Jeff Gray[1], Rajeev Raje[2], Mihran Tuceryan[2], Andrew Olson[2] and Mikhail Auguston[3]

Abstract

The current synergy of Component-Based Software Engineering (CBSE) and Software Product Line Engineering (SPLE) requires evolution to facilitate Distributed Real-time and Embedded (DRE) system construction. Such evolution is driven by inherent Quality of Service (QoS) characteristics in DRE systems. This paper introduces a QoS-driven SPLE framework (QoSPL) as an analysis and design paradigm for constructing a set of DRE systems as a product line. Leveraging separation of concerns, DRE systems are analyzed and designed by a collection of QoS systemic paths, each of which individually determines how well the service performs along the path and as a whole represents a behavioral view of software architecture. The paradigm reduces construction workload from the problems of tangled functional and QoS requirements and abundant infeasible design alternatives, and offers a less subjective QoS evaluation. The adopted formalisms also facilitate high-confidence DRE product line construction.

1. INTRODUCTION

Component-Based Software Engineering (CBSE) offers a possible solution to foster high-confidence system construction stipulating design by contract [3]. Software Product Line Engineering (SPLE) [10] further enriches CBSE by analyzing and reusing common features. Distributed Real-time and Embedded (DRE) systems are mission-critical and highly complex [14], requiring satisfaction not only of time-critical issues, but also numerous functional and Quality of Service (QoS) requirements specific to the mission. To handle such complexity, the synergetic technologies of the current CBSE and SPLE concepts are adapted to DRE system construction. Such technologies, however, are not a panacea because of the inherent QoS characteristics of DRE systems [13]. The three main obstacles to building such systems by the current CBSE and SPLE techniques are:

Challenge 1 - The QoS Sensitive Problem: The performance fulfillment of mission-critical component-based DRE systems pertains to the availability of system resources, which directly affect the stringent QoS demands of the system [14]. Insufficient or unmanageable QoS provisioning [16] over system resources results in inferior performance or failures on missions.

Challenge 2 - The Tangled Requirements Problem: In the validation of requirements after component composition by CBSE, QoS tuning is problematic in that obtaining an optimal solution from hundreds of requirements is arduous and tedious. Composition may require effort on many infeasible designs to satisfy all requirements. For dynamic evaluation, such wasted effort is eliminated by evaluating functional and QoS requirements interchangeably. Such tangled requirements, however, complicate the evaluation and suppresses the development pace, because CBSE treats components as composition units and primarily concentrates on functional requirements.

Challenge 3 - The Abundant Alternatives Problem: One of the common objectives of CBSE and SPLE is to increase the diversity among software products. Despite abundant variable alternatives generated, many of them are inadequate regarding satisfying their requirements. Infeasible design alternatives should be avoided.

The three challenges are derived from the fact that functional and QoS requirements are equivalently important for DRE system construction. To address these obstacles, this paper introduces a separation of concerns analysis and design paradigm in the vision of the UniFrame project [11], a standardized framework for knowledge-based component composition, as part of a QoS-driven product line framework (QoSPL). QoSPL expresses a DRE system as a collection of QoS systemic paths [16]. To perform a service obeying its functional requirements (e.g., draw a virtual object on a monitor) at the service level, a sequence of components (i.e., a functional path [16]) collaborates with each other in a specific order. Each component carries out a specific task (e.g., rendering) and the combination of these tasks fulfills the overall requirements. Regarding QoS requirements, a QoS systemic path quantitatively describes *how well* the corresponding functional path can be satisfied. Such a path applies QoS formulae (from the specification document) for analyzing the resulting QoS behavior. Given specific component dependencies and composition rules, QoS systemic paths for each QoS property are constructed and analyzed using a specification language approach in a way that the synergy of commonality and QoS

1 Department of Computer and Information Sciences, University of Alabama at Birmingham, Birmingham, AL 35294, USA, {liush, bryant, gray}@cis.uab.edu

2 Department of Computer and Information Science, Indiana University Purdue University Indianapolis, Indianapolis, IN 46202, USA, {rraje, tuceryan, aolson}@cs.iupui.edu

3 Department of Computer Science, Naval Postgraduate School, Monterey, CA 93943, USA, maugusto@nps.navy.mil

satisfaction analyses for constructing a product line is fulfilled. Suitable QoS systemic paths are accumulated, rendered and assured by a modeling approach as a behavioral view of the DRE software architecture. The entire product line can be constructed in the same way by selecting different combinations of QoS systemic paths. Because this paradigm only pertains to quantitative QoS (e.g., deadlines) described by the QoS formulae and in the representations of QoS systemic paths, non-quantitative QoS (e.g., security) is not considered in this paper.

The contributions of this paper are twofold: (a) The paper presents an approach that solves the above stated obstacles by utilizing QoS formulae indicated in the specification documentation for more precise and less subjective QoS measurement (Challenge 1), by reducing the amount of workload using a separation of concerns evaluation (Challenge 2) and by eliminating infeasible design alternatives in compliance with QoS constraints specified in QoS requirements (Challenge 3), and (b) It describes a specification language and modeling approaches to analyze commonality, variability, and reusability at the component and service levels, which facilitate finer-grained analytical results.

The paper is organized as follows: the next section introduces the formalisms applied in the paper; Section 3 presents QoSPL and a case study of Mobile Augmented Reality Systems (MARS) [12]; Section 4 summarizes the related work; and Section 5 concludes the paper.

2. BACKGROUND

Two-Level Grammar++ (TLG++) [2] is an object-oriented formal specification language that consists of two Context-Free Grammars (CFGs) defining syntax and semantics, respectively. TLG++ has been used for design space exploration and QoS requirements analyses in UniFrame [8]. In this paper, TLG++ is used for syntactically and semantically defining QoS properties with respect to corresponding QoS systemic paths and for analyzing commonality and variability of a product line.

Timed Colored Petri Nets (TCPNs) [4] are formalisms beneficial in modeling concurrent and asynchronous distributed systems with ancillary notations for time and user-defined data types and values. A TCPN is graphically represented by a Petri Net graph, which statically expresses the interrelationships between states of a modeling system by the abstractions of tokens, places, arcs, types, and transitions. The dynamic and executable properties of the TCPN are described by variables and binding, enabling, occurrence, and occurrence sequences and steps on the Petri Net graph. The reachability tree [4] is the execution result derived from the static and dynamic properties of the Petri Net graph. The execution orders of the reachability tree can be expressed by sequences of markings, which are distributions of tokens over the places of a TCPN.

The QoS systemic paths expressed in the first CFG of a TLG++ class are manually depicted in a Petri Net graph. The QoS semantics described in the second CFG of the TLG++ class are manually mapped to the dynamic properties of the Petri Net graph. Besides the synergetic advantage of the commonality and QoS satisfaction analyses, TLG++, applied in the UniFrame project reduces the possible accidental complexity. TCPN enriches TLG++ by introducing asynchronous, concurrent, and time properties to design and analyze a DRE product line. A MARS example is presented in the next section to describe how to apply these two formalisms and why they are suitable and beneficial to construct a DRE product line.

3. QoSPL

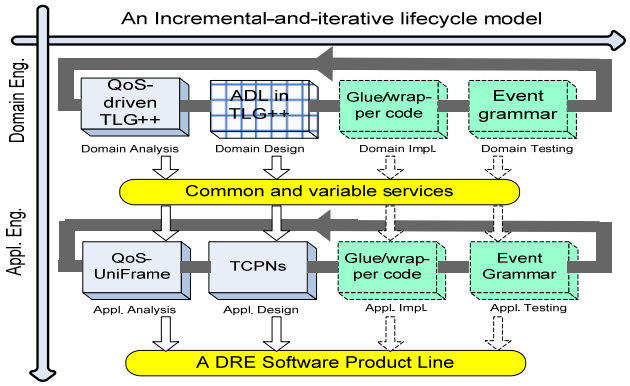


Figure 1. The overview of QoSPL.

QoSPL for DRE systems complies with the purposes and concepts of SPLE and software architectures [15] and introduces a QoS perspective to analyze and design a DRE product line. Figure 1 shows the overview of QoSPL based on the incremental-and-iterative lifecycle model. The objectives in the domain engineering process are to analyze common and variable requirements, to design prescribed reference architecture [10] for its product line, to implement a set of reusable core assets and to verify and validate the core assets. TLG++ is employed in the analysis workflow and is used as an Architecture Description Language (ADL) [15] in the design workflow (the grid box). The application engineering process aims at reusing the core assets of each workflow in the domain engineering process to exploit the artifacts of variable features of each workflow on individual applications. QoS-UniFrame [7], a TCPN-based modeling tool, is applied in the analysis and design workflows in this process. The implementation and testing workflows of both processes (dashed boxes) are out of the scope of this paper.

This section presents QoSPL using a case study of a Battlefield Training System (BTS) from the domain of mobile augmented reality. A Mobile Augmented Reality System (MARS) is a DRE system concentrated on enriching the user environment by merging real and virtual objects. Generally, a MARS consists of six subsystems: computation, presentation, tracking and registration, environmental model, interaction, and wireless communication [12] (as shown in Figure 2). The BTS is an outdoor MARS for training soldiers to conduct military operations. The

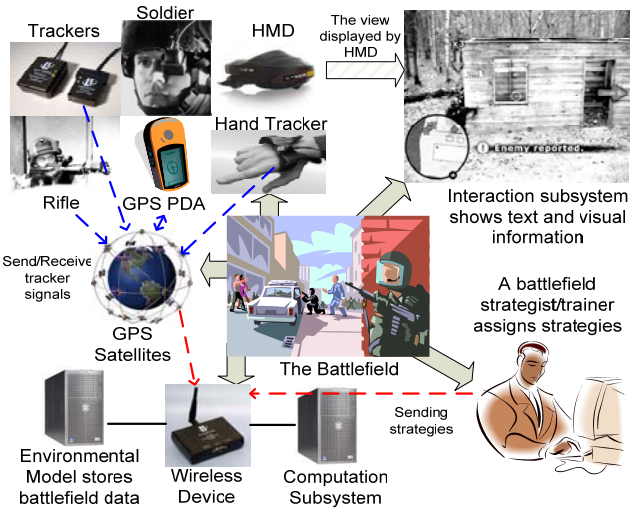


Figure 2. The overview of the BTS example.

hardware requirements of the BTS are a video see-through Head-Mounted Display (HMD), a headphone, position sensors (PS), orientation sensors (OS), mobile devices, and a specialized rifle comprising both types of sensors. UniFrame, in its knowledge base, stores functional and QoS requirements of components for rendering processing, speech processing, speech recognition, speech synthesis, text processing (TxP), tracking processing (TkP), presentation (Present), wireless communication (WC), and environmental model (EM) that stores the geometrical and hierarchical 3D information. The following paragraph describes a scenario for a BTS example.

A soldier is to rescue a virtual hostage in a battlefield. The position and orientation sensors on his body send back the 3 Degrees Of Freedom (3DOF) data to the tracking subsystem every half second via wireless communication. As the soldier is standing on specific positions with specific orientations in some buildings derived from a predefined tactical scenario stored in the computation subsystem, his HMD displays the enemies registered by the tracking system, computed by the presentation subsystem and rendered by the interaction subsystem. The soldier communicates with the command center via his headphone. The information of the soldier's current position is displayed on the HMD by text.

The scenario is applied to the domain engineering and application engineering processes respectively using TLG++ and QoS-UniFrame in the next two subsections.

3.1 Domain Engineering

This section describes the commonality and QoS satisfaction analyses for the quantitative services of the BTS example using TLG++ in the analysis workflow.

Figure 3 defines the `TextDeadlineFromClient` class (i.e., service) in TLG++ for commonality and QoS satisfaction analyses. Lines 2 to 9 constitute the first CFG that describes all possible QoS systemic paths of `TextDeadlineFromClient`. Line 2 shows the types of

```
1  class TextDeadlineFromClient.
2    Syntax :: Sensor WC TkP.
3    Sensor :: OS PS ; PS ; OS.
4    PS :: ps1 ; ps2 ; ps3.
5    OS :: os1 ; os2 ; os3 ; os4.
6    WC :: wc1 ; wc2.
7    TkP :: trackProcessing.
8    PreCondition, PostCondition :: Boolean.
9    Sum :: Double.
10   semantics of sendPositionFromClient :
11     PreCondition := semantics of queryComponent with OS PS
           WC and TkP, //please refer to [8] for the semantics
12     if PreCondition then Sum := semantics of sumOfMTAT
           with OS PS WC and TkP,
13     else ErrorMessage, end if,
14     Double semantics of sumOfMTAT with OS PS WC and
           TkP :
15       return OS semantics of getMTAT + …….
16     PostCondition := semantics of queryPattern with Sum.  …
17 end class. //please refer to [8] for the semantics
18 class OS extends Id.
19   token : "{letter}({letter}|{digit})*".
20   semantics of getMTAT : ……//please refer to [8] for how to
       //access its value specified in TLG++
21 end class.
22 class TextDeadlineFromServer.
23  Syntax :: GetEnvInfo TxP TkP Present Interact WC Display.
24  GetEnvInfo :: EM TkP.
25  FontResult :: Font.
26  Character :: Integer.
27  PreCondition, PostCondition :: Boolean.
28  Sum :: Double.
29  semantics of sendTextToClient :
30      Sum := EM semantics of getMTAT with getEnvInfo +
31      TkP semantics of getMTAT with getTrackingResult +
32      TxP semantics of getMTAT with decideTextContents +
33      TkP semantics of getMTAT with registerTextResult +
34      Present semantics of getMTAT with glutBitmapCharacter
          with FontResult and Charater +
35      Interact semantics of getMTAT with
          manageDisplayPosition +
36      WC semantics of getMTAT with sendText +
37      Display semantics of getMTAT with display.   ……
38 end class.
```

Figure 3. The commonality and QoS analyses in TLG++.

components involved in the text rendering task. The path starts from obtaining tracking results from the position and orientation sensors (Sensor) of a soldier. The wireless communication (WC) transmits the results to the `trackProcessing` (TkP) component. Three combinations in line 3, divided by semicolons as the counterpart of the meta-symbol "|" in Extended Backus-Naur Form, show that the `MTAT` (Mean Turn Around Time) values for tracking position and orientation are affected by the combinations. Because there may be more than one appropriate component for functionality-determined tasks, lines 4 to 6 show the suitable different components for position sensors (PS), orientation sensors (OS), and wireless communication (WC). After parsing the first CFG, 38 (4*3*2*1 + 3*2*1 + 4*2*1) parse trees, as shown in Figure 4, will be generated following the gray arrows (i.e., six horizontal and a vertical

arrows). Infeasible paths of the composed service can be eliminated in compliance with the pre- and post-conditions for composition (e.g., QoS constraints), as shown in lines 11 and 16. The semantics of pre- and post-conditions can be obtained in [8]. The `OS` class describes legitimate instance identities in line 19 and the `getMTAT` method, is invoked in `TextDeadlineFromClient` and returns a value defined by MARS experts and stored in the knowledge base. The `TextDeadlineFromServer` class accumulates `MTAT` values by assigning specific functionalities (e.g., `getEnvInfo`) in the `getMTAT` method of each component in line 23. In the case study, all components are deployed on a Local Area Network (LAN), and the communication time between components is negligible. The QoS evaluation for MTAT at the service level is therefore the sum of all individual components along the `TextDeadlineFromServer` QoS systemic path. Lines 30 to 31 access the environmental and tracking information. The text contents are computed by the text processing component (`textProcessing`) in line 32. The registration for the new text result is updated in line 33. Consequently, the `renderingText` component invokes its function `glutBitmapCharacter` (an OpenGL utility function), to depict the text. The text outcome is managed by the interaction subsystem, transmitted by the wireless communication subsystem and displayed on the HMD.

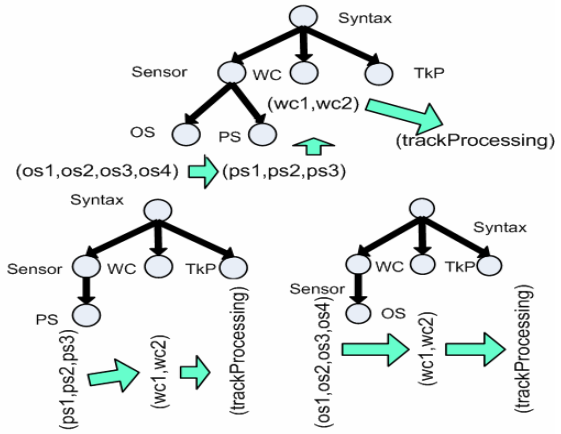


Figure 4. The parse tree of `TextDeadlineFromClient`.

The commonality, variability and reusability analyses in this workflow can be observed in Figure 4. `WC` and `TkP` are mandatory/reusable component types (shown as intermediate nodes in a parse tree); because these types exist in all three parse trees and have no further children. `Sensor` is mandatory and an "OR" (i.e., more-of) component type, because it is present in all parse trees and each tree has three different intermediate child types, which relates to line 3 in Figure 3. `trackProcessing` is a mandatory component (shown as a leaf in a parse tree). The components of the `OS`, `PS`, and `WS` component types are alternative atomic features. Because each component appears once in each parse tree in Figure 4, there is no optional (i.e., one or none) component in the BTS example. The commonality and reusability rate of each component and each QoS system path within the family will be decided based on the following intuition: because more satisfactory QoS systemic paths have higher probabilities to be selected for the product line construction, common and reusable components are most likely found in the paths with higher analysis results computed from their evaluation formulae.

Table 1. The symbol table of `TextDeadlineFromClient`.

Identity	os1	os2	os3	os4	ps1
Type	OS	OS	OS	OS	PS
MTAT (ms)	0.2	0.3	0.45	0.2	0.25
Identity	ps2	ps3	wc1	wc2	trackProcessing
Type	PS	PS	WC	WC	TkP
MTAT (ms)	0.15	0.2	0.5	0.4	0.9

Table 1 is a partial symbol table generated after parsing `TextDeadlineFromClient` where the values are defined by BTS experts and accessed from the knowledge base. Derived from Table 1, `os1`, `os4`, `ps2`, `wc2`, and `trackProcessing` are the most commonly used components at the component level. At the service level, [`ps2`, `wc2`, `trackProcessing`] is the most appropriate QoS systemic path within the family, because its cumulative `MTAT` value is the shortest among all paths (1.45 *ms*). Without the orientation information of a soldier, however, it is impossible to depict a virtual object on the HMD correctly. Consequently, [`os1`, `ps2`, `wc2`, `trackProcessing`] and [`os4`, `ps2`, `wc2`, `trackProcessing`] are the most promising QoS systemic paths (1.65 *ms*) for `TextDeadlineFromClient`. The TLG++ classes of `ThreeDimDeadlineFromClient`, `ThreeDimDeadlineFromServer`, and `SpeechDeadline` are omitted due to space considerations.

3.2 Application Engineering

After the commonality and QoS satisfaction analyses, QoSPL obtains quantitative (i.e., *how well*) results of each service as shown in the middle round box in Figure 1. In application engineering, QoSPL utilizes QoS-UniFrame [7] to construct a set of DRE systems. QoS-UniFrame is a TCPN-based modeling toolkit implemented in the GME[4] (Generic Modeling Environment), a meta-configurable modeling environment. Initially, QoS systemic paths (i.e., the outcomes of the domain analysis workflow), the reference architecture, and sorted QoS requirements are acquired from the knowledge base. The behavioral view of a DRE system that comprises a collection of satisfactory and necessary QoS systemic paths is depicted by the Petri Net graph of a TCPN in QoS-UniFrame. QoS-UniFrame also generates the reachability tree of the Petri Net Graph. Such a tree introduces a sequence of markings, which individually represents a state and as a whole describes a valid

4. http://www.isis.vanderbilt.edu/Projects/gme/

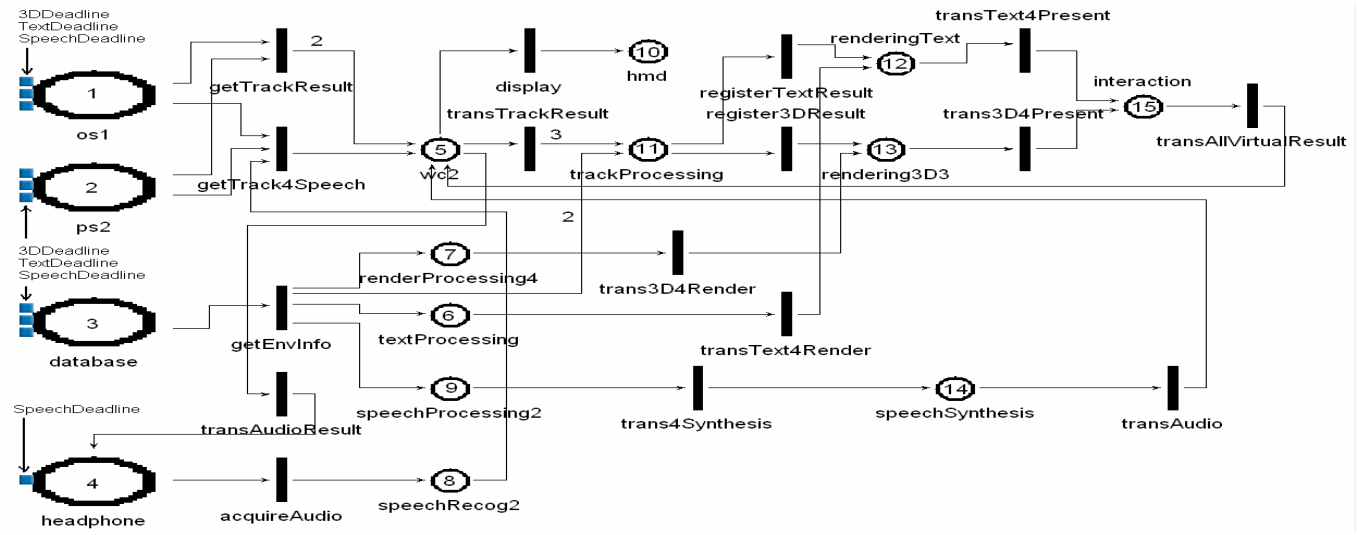


Figure 5. The behavioral view of the BTS example.

execution order under given static and dynamic properties (i.e., enabling and firing rules in transitions and/or arcs [4]) and constraints. As system-level QoS predicates are inserted in specific transitions, and the reachability tree is generated, the modeling results validate the QoS requirements at component, service, and system levels in such a behavioral view of software architecture.

Figure 5 shows the behavioral view of the BTS example using a Petri Net graph. `TextDeadline` (i.e., `TextDeadlineFromClient` and `TextDeadlineFromServer`), `ThreeDimDeadline` (i.e., `ThreeDimDeadlineFromClient` and `ThreeDimDeadlineFromServer`), and `SpeechDeadline` are three required QoS systemic paths described in the BTS scenario. In Figure 5, circles and ellipses, representing places in TCPNs, are the necessary hardware and software components; black bars, as transitions in TCPNs, are functionalities provided in the components; the small boxes within `os1`, `ps2`, `database`, and `headphone` are the tokens for representing the initial marking of the Petri Net graph; and the number 2 below `trackProcessing` is the weight assigned to the arc from `getEnvInfo` to `trackProcessing` for the purpose of virtual object registration. Because TCPNs are used for modeling DRE systems including synchronous and asynchronous characteristics, a timer and predicates are embedded in the transitions of Figure 5 to control the token flows.

Figure 6 shows a sequence of markings resulting from QoS-UniFrame that expresses a valid execution order considering three deadlines. Row Marking contains the place identities numbered in Figure 5. For simplicity, rows M0 to M8 only express the number of tokens (instead of the identities) in each place after specific transition(s). M0 is the initial marking as depicted in Figure 5. Timers and predicates within transitions decide the following markings to be generated along with arcs and weights. For example, the predicate in `transTrackResult` specifies that if (a)

Marking	(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)
M0	(3,3,3,1,0,0,0,0,0,0,0,0,0,0,0)
M1	(1,1,3,0,2,0,0,1,0,0,0,0,0,0,0)
M2	(0,0,3,0,3,0,0,0,0,0,0,0,0,0,0)
M3	(0,0,3,0,0,0,0,0,0,0,3,0,0,0,0)
M4	(0,0,0,0,0,1,1,0,1,0,2,0,0,0,0)
M5	(0,0,0,0,0,0,0,0,0,0,2,1,1,1,0)
M6	(0,0,0,0,1,0,0,0,0,0,0,0,0,0,2)
M7	(0,0,0,0,3,0,0,0,0,0,0,0,0,0,0)
M8	(0,0,0,1,0,0,0,0,0,2,0,0,0,0,0)

Figure 6. The markings of the BTS example.

the tokens of `TextDeadline`, `ThreeDimDeadline`, and `SpeechDeadline` are residing in `wc2`, and (b) each currently evaluated deadline is fewer than its QoS constraint, `transTrackResult` fires a token to `trackProcessing` at the beginning of the next periodic firing. To evaluate system-level QoS requirements (e.g., all deadlines should not exceed 10 seconds), the corresponding QoS evaluation formulae can be embedded into either each transition (i.e., dynamic evaluation) or into the transitions right before the interaction subsystem (i.e., evaluation after composition). If all QoS requirements are satisfied, the Petri Net graph (Figure 5) is the behavioral view and a member of the product line of the example.

To construct other members of the product line of the BTS example, different combinations of QoS systemic paths can be instantiated in the TCPN model. For example, [`os4`, `ps2`, `wc2`, `trackProcessing`] can be selected as an alternative of `TextDeadlineFromClient` and composed with the other QoS systemic paths depicted in Figure 5. As long as all QoS requirements are fulfilled at the component, service, and system levels, the second member of the product line is designed. Please note that the QoS evaluation formulae and predicates need no revision as long as all requirements remain the same. Other product line members can be constructed using the same approach as shown in the bottom round box in Figure 1.

A BTS product line is constructed by QoSPL that shares common components and services with their unique features and satisfies both functional and QoS requirements at component, service, and system levels. Following the incremental-and-iterative lifecycle, the common components and/or services of the BTS product line may reduce the development cost and time to market.

4. RELATED WORK

Feature-Oriented Reuse Method (FORM) is derived from Feature-Oriented Domain Analysis (FODA) [6]. Non-functional features are used for decomposing functional features. Such a framework is not sufficient to analyze and manage numerous DRE QoS requirements (Challenge 1). *KorbA* [1] separates abstraction, specificity (SPLE) and composition (CBSE) concerns strictly. The specification and realization of each sub-component are activated interchangeably (Challenge 2). It provides quality assurance and quality assessment techniques by integrating inspections (Challenge 1) and quantitative analysis of UML models. *Quality-Driven Architecture Design and Analysis (QADA)* [9] processes quality analysis and architecture design phases interchangeably (Challenge 2). Scenario-based (i.e., consensus or questionnaire (Challenge 1)) analysis is applied to evaluate the conceptual architecture alternatives. The architecture design phase iteratively consults the feedback from the quality analysis and then designs the architecture accordingly. *Mini-Middleware* [5] treats the DRE middleware as the assemblage of common white box components that provides commonly used features. To construct a middleware product line, the DRE middleware is specialized and optimized by shrinking its specific functionalities based on stringent QoS properties.

QoSPL not only comprises the FODA, SPLE, CBSE, and quality-driven characteristics, but also solves the three obstacles existing in FORM, KorbA, and QADA. The core difference between QoSPL and Mini-Middleware is QoSPL utilizes two formalisms to facilitate high-confidence DRE system construction.

5. CONCLUSION

This paper presents a novel design and analysis paradigm for developing a QoS-oriented product line in the domain of DRE systems. For domain engineering, TLG++ analyzes the common and variable features as well as QoS requirements at a finer-grained abstraction level. For application engineering, QoS-UniFrame concurrently analyzes and designs DRE systems by collecting satisfactory QoS systemic paths out of each family. After assurance by the TCPN reachability tree, the output of QoS-UniFrame is the behavioral view of the DRE software architecture. QoSPL solves the QoS-sensitive, component composition, and abundant alternatives problems that plague many CBSE and SPLE. QoS analyses for shared resources have been tackled by a system level statistical and stochastic approach [7]. QoS-UniFrame promises to analyze finer-grained QoS at the component and service levels.

REFERENCES

[1] C. Atkinson, et al. *Component-based Product Line Engineering with the UML*. Addison-Wesley, 2001.

[2] B. R. Bryant, B.-S. Lee. Two-Level Grammar as an Object-Oriented Requirements Specification. *Proc. 35th Hawaii Intl. Conf. System Sciences*, pp 19-26, 2002.

[3] G. Heineman, W. T. Councill. *Component-Based Software Engineering*. Addison-Wesley, 2001.

[4] K. Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use, Volume 1, Basic Concepts*. Springer-Verlag, 1997.

[5] A. Krishna, et al. Model-driven Middleware Specialization Techniques for Software Product Line Architecture in Distributed Real-time and Embedded Systems. *Proc. MoDELS 2005 Workshop MDD for Software Product-Lines: Fact or Fiction?*, 2005.

[6] K. Kang, et al. FORM: A Feature-oriented Reuse Method with Domain-specific Reference Architectures. *Annuals of Software Engineering*, 5: 143-168, 1998.

[7] S.-H. Liu, et al. QoS-UniFrame: A Petri Net-based Modeling Approach to Assure QoS Requirements of Distributed Real-time and Embedded Systems. *Proc. 12th Intl. Conf. Eng. of Computer Based Systems*, pp 202-209, 2005.

[8] S.-H. Liu, et al. Quality of Service-Driven Requirements Analyses for Component Composition: A Two-Level Grammar++ Approach. *Proc. 17th Intl. Conf. Software Eng. and Knowledge Eng.*, pp 731-734, 2005.

[9] M. Matinlassi, et al. Quality-Driven Architecture Design and Quality Analysis Method: a Revolutionary Initiation Approach to Product Line Architecture. Tech. Report, *VTT Pub.: 456*, VTT Electronics, 2002.

[10] K. Pohl, G. Böckle, F. van der Linden. *Software Product Line Engineering: Foundations, Pinciples, and Techniques*. Springer-Verlag, 2005.

[11] R. R. Raje, et al. A Quality of Service-Based Framework for Creating Distributed Heterogeneous Software Components. *Concurrency and Computation: Practice and Experience*, 14(12): 1009-1034, 2002.

[12] T. Reicher. *A Framework for Dynamically Adaptable Augmented Reality Systems*. Doctoral Dissertation, Technical University of Munich, 2004.

[13] D. C. Schmidt. Model Driven Development for Distributed Real-time and Embedded Systems (Keynote presentation), *8th Intl. Conf. Model Driven Eng. Languages and System*, 2005.

[14] D. C. Schmidt. R&D Advances in Middleware for Distributed Real-time and Embedded Systems. *Communications of the ACM*, 45(12):43-48, 2002.

[15] M. Shaw, D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[16] N. Wang, et al. QoS-enabled Middleware, *Middleware for Communications*. ed. Q. H. Mahmoud. John Wiley and Sons, 2003.

Performance Evaluation of Component System based on Container style Middleware

Yong Zhang, Ningjiang Chen, Jun Wei, and Tao Huang
Institute of Software, Chinese Academy of Sciences
E-mail :{ yzhang, river, wj, tao}@otcaix.iscas.ac.cn

Abstract

For design-level performance prediction of component-based system (CBS) hosted by container style middleware, this paper proposes an approach integrating middleware components and their performance attributes into application Unified Model Language (UML) model, based on architectural patterns. The performance model derived from the integrated UML model can efficiently represent the impact of middleware. So, analysts do not have to know about the internal details of middleware when evaluating the performance of CBS. The process is illustrated by a case study, and the comparison of prediction with measurement shows the efficiency of proposed approach.

1 Introduction

Software architecture plays an important role in determining software quality characteristics, e.g., performance. Performance effect of architectural decision can be evaluated at an early phase by constructing and analyzing quantitative performance model. Several approaches to deriving performance model from architecture description have been proposed [1, 2].

Middleware for distributed system is a kind of system software that resides between the application and the underlying operating system, network protocol stack, and hardware. Software middleware can help to distributed system faster development cycles, decreased effort, and greater software reuse [3]. At the same time, the middleware will obviously impact the architecture and the performance of component application. It is necessary to consider the impact of middleware at the performance modeling [4].

At times, middleware are not included as a part of CBS architecture description, but as part of its environment. So, performance model derived from architecture descriptions, by directly using current architectural level modeling methods, cannot reflect the impact of middleware. Several work, as in [5, 6, 7, 8, 9, 10], analysts have to build performance model for application and middleware separately and then composite them, even model from scratch by hand. These approaches require detailed knowledge of middleware internals and modeling language itself, which decreases the ease of use and automatization of modeling, then hinders successful application of early performance analysis.

The goal of this paper focuses on modeling the performance impact of Container style middleware, broadly utilized infrastructure for server-side component technology [11]. Based on architectural pattern adopted in Container style middleware, this paper proposes an approach to integrating middleware component interactions and performance information into UML models of application. Performance model including the impact of middleware can be derived from resulting UML models. The parameters of performance model can be obtained from UML activity diagram [12], in which performance information is annotated with a language extension for UML, called UML Profile for Schedulability, Performance and Time (SPT Profile) [13].

The method proposed in this paper does not require performance analyst to know about middleware internal details, when evaluating the performance of CBS hosted by container style middleware. The architectural pattern-based approach can be extended to deal with different style middleware. The process is illustrated by a case study, and the comparison of prediction with experiment measurement shows the approach is efficient.

The rest of this paper is organized as follows: Container style middleware and its impact on CBS are analyzed in Section 2 and section 3; The approach to getting integrated UML description and modeling process are presented in Section 4; The approach is demonstrated by a case study in Section 5; We introduce some related work in Section 6 and conclude in Section 7.

2 Container style middleware

Distributed object middleware helps to alleviate complexities associated with developing distributed software, enabling separation of concerns between application logic and system service, such as

communication, transaction, security, component life cycle management, etc. The execution environment that is responsible for adding above technical concerns to component is generally called Container [11].

Container middleware must be flexible enough to integrate, manage, reuse, and extend these services. Two architectural patterns are adopted for it: Proxy pattern, and Chains of interceptor [11]. These two patterns enhance system flexibility and extensibility, enabling easily add functionality to the system for dynamically changing its behavior, allowing for executing services if a kind of trigger (e.g., a communication call event) fires.

The main component types in container architecture include Container, Dispatcher, Interceptor, and Context. Interceptor is registered with dispatcher (it is a part of Container). Once interceptor is registered, the container notifies the dispatcher of any events that have occurred. Upon receiving an event, the dispatcher examines the event to determine which interceptors need to be notified, and pass them the context containing the event (as shown in Fig.1). When triggered, the interceptor examines the context and executes its related functionality.

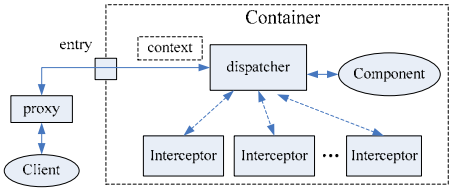


Fig. 1 Interaction relation of interceptors in Container

An example of container style middleware, J2EE application server, is shown in Fig.2. Middleware runtime services (such as transaction, security) are implemented as concrete interceptors. Invocation handlers at client side and server side, together with stub, are responsible for the processing of distribution. Application server also enables client-side interceptors for flexibility.

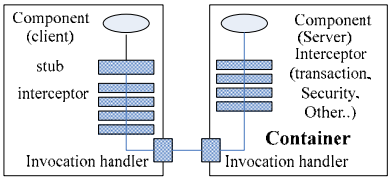


Fig. 2 Structure of Application Server

3 Structure and performance impact

Client attempting to make a request to the server will not send the request directly to the server, but rather to a stub acting as a client side proxy, which forwards the request to container through invocation handler. After processing inside container, server side proxy sends the response back to the client, along the same path in the opposite direction.

The stub and invocation handlers perform additional operations on the request and the response (marshaling and unmarshaling), to transform the data (e.g., parameter values) from the native format to a language independent wire format and back. This allows cooperation between client and server, implemented in different languages and running on variously platforms. Meanwhile, all these operations will incur performance overhead.

When the request arrives at server side container, container creates invocation context for request, which provides operation information for accessing resource, security, the current transaction, or component instance-specific information. Invocation context will be passed through chains of interceptor, triggering related interceptors (middleware service) explicitly declared in component deployment descriptor. In addition, some implicit services such as component instance management and life cycle management will also be used. At last, request will be sent to component business method. After processing, response will be sent back along the chains of interceptor to server proxy, which continues sending back to client, as explained above.

The creation of invocation context and various middleware services inside container all will affect the performance. Different middleware services serve the request concurrently under the control of different processes/threads. In the chain of interceptors, the predecessor interceptor finishes processing and forwards the context to the successor for further processing, while the former continues serve next request.

The overhead of determining interceptor or chain of interceptor can be included in the context creation phase. Initialization and destroy of container usually happen at the time of component deployment and undeployment. In view of performance is a runtime attribute of system [2], so the impact of these two phases needs not to consider.

The performance impacts of middleware relate with specific application. It is necessary to provide relative middleware usage information, such as, which invocations are remote, what middleware services will be used, and their executing demands, etc. In this paper, the middleware usage description will be provided in XML-based file, the Schema of which is shown in Fig.3.

The elements in Fig.3 are declared according to each pair of interacting components that use middleware. For each invocation between client and server component, there is a <invocation> declaration: <transitionID> represents the transition referring to this call in UML activity diagram; <invocationType> represents remote or local call; The details of <isRemote> specifies the service

demand of processing phases of remote invocation; <services> represents middleware services to use during invocation, in which the details are specified; In this paper, each kind of middleware service as a whole is specified, instead of modeling its internal details; <instantanceHandling_time> and <stateHandling_time> declare performance information about component instance management and state management.

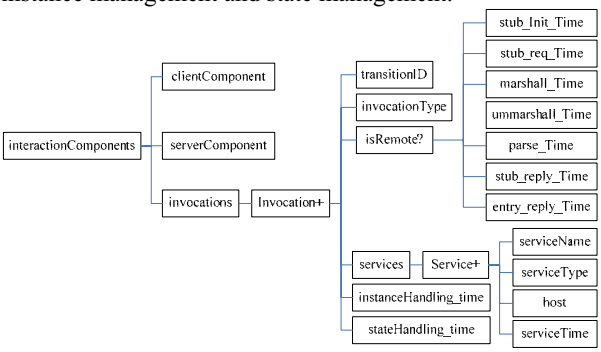


Fig. 3 XML Schema for middleware usage information

4 Obtaining integrated UML models

In order to enable the performance model derived from UML model to reflect the impact of middleware, it is necessary to add the middleware components and their interaction relation to application UML model.

In [14, 15], authors propose a method transforming UML model into LQN (Layered Queueing Network) performance model. The input is UML collaboration, UML deployment diagrams and activity diagrams, the output is the corresponding LQN model, which can be read directly by existing LQN solvers [16]. The integrating method proposed in this paper will build on above transforming method. In particular, we mainly add middleware information to above UML models. To each architectural pattern analyzed in former section, we will look for concrete middleware components and information in middleware usage file, and then add these to CBS UML models.

The integrating begins with the UML collaboration, which represents high level architectural view. For Proxy pattern, components (*stub, invocationHandler_client, and invocationHandler_server*) and a series of "client-server" collaboration relations among them need to be added to original UML collaboration. For Chains of interceptor pattern, components (*Container, all middleware service components, instanceHandler, stateHandler, etc*) and a series of "chains of interceptor" relations among them also need to be added. In this paper, each kind of middleware service will be model as a single service component.

The integrating continues with UML activity diagram. Middleware component interaction details need to be added to original activity diagram, guided by changed UML collaboration. UML activity diagram uses swimlane to represent behavior of concurrent components [12]. For each component added to UML collaboration, a swimlane and component actions will be added to activity diagram. For each middleware service, only one action will be added for its service behavior, and named like *"middleware service name"+"handling"*. The added actions of other components will be explained in following algorithm.

After adding middleware components, the original direct call between the client and server component needs to be redirected. If the call is synchronous, both request and reply need to be redirected.

Finally, the integrating algorithm deals with the UML deployment diagram to reflect the allocation of middleware components to hardware devices. The details of above integrating are detailed as follows.

Parsing XML-based middleware usage describing file;
Obtaining integrated UML Collaboration {
For each *interactionComponents* element {
find collaboration relation between *clientComponent* and *serverComponent* in UML Collaboration;
If *invocationType* between *clientComponent* and *serverComponent* is remote {create components *stub, invocationHandler_client, invocationHandler_server* and *Container;*
create *client-server* collaboration in turn among *clientComponent, stub,invocationHandler_client , invocationHandler_server,* and *Container;*
For each *service* element {create component according to *serviceName* ;}
create components *instanceHandler* and *stateHandler*;
create *chain of interceptor* collaboration in turn among *Container, $serviceName_{(1)}$, $serviceName_{(2)}$,…,$serviceName_{(n-1)}$, instanceHandler, stateHandler ,* and *serverComponent*;
} else {/*if invocation is local, only service components are added*/
For each *service* element {create component according to serviceName;}
create components *container*, *instanceHandler* and *stateHandler*;
create *client-server* relation between *clientComponent* and *Container*, *chain of interceptor* relations among *Container, $serviceName_{(1)}$, …, $serviceName_{(n)}$, instanceHandler,stateHandler,*and *serverComponent*;};
delete original collaboration between *clientComponent* and *serverComponent*;
}
}
Obtaining integrated UML Activity diagram {
For each invocation between *clientComponent* and *serverComponent* {
find transition referenced by this invocation (say *original transition*) and delete it;
find source and target partition, source and target action of this transition;
If *invocationType* is remote {
If *stub* not added {/*before the first invocation, adding stub initializing */
create partition *stub* and action *stub_init* in *stub*;

find starting state *s* of partition *clientComponent*， *and* target of transition starting in state *s* ; (say *s'*)

create transition *s->stub_init->s';* }

create action *stub_req* in *stub*, partition *invocationHandler_client* and its action *marshaling*, partition *invocationHandler_server* and its action *unmarshaling*, partition *Container* and its action *parse* ; /*redirect the call */

create *transition source->stub_req->marshal->unmarshaling->parse*;

}

For each *service* sub-element of *services* element {

create partition *service*;(named by $service_{(1)}$, ...,$service(n)$)

create actions *"waiting" and* "$service_{(i)}$"+"handling" in each *service*;

}; /*adding interaction relations among these service partitions*/

create one join in $service_{(1)}$, one input coming from *waiting* of $service_{(1)}$, another from *parse* of *entry*, and output pointing to $service_{(1)}$*handling;*

For i=2 to n {

create one fork in $service_{(i-1)}$, input coming from action $service_{(i-1)}$*handling* of $service_{(i-1)}$, one output pointing to *waiting* of $service_{(i-1)}$, another pointing to *waiting* of $service_{(i-1)}$;

create one join in partition $service_{(i)}$, one input coming from *waiting* of $service_{(i)}$, another coming from fork in $service_{(i-1)}$, and output pointing to action $service_{(i)}$*handling;*

};

create partition *instanceHandler* and actions *waiting and instanceHandling,* partition *stateHandler* and its actions *waiting and stateShift*;

/*adding interaction between the two partitions resembles that of above service partitions, here omitted */

/*finally, adding invocation to business method of serverComponent*/

create transition from action *stateShift* of *instanceHandler* to target action of *original transition*;

if invocation is synchronous {/* the response also need to redirect */

find transition representing response, *source action stat*e and *target action state* of the transition;

create actions *marshaling* in *invocationHandler_server* , *unmarshaling* in *invocationHandler_client, entry_reply in Container,* and *stub_reply in stub* ;

create transition *source action state-> entry_reply-> marshaling-> stub_reply->unmarshaling->target action state;*

}

}

}

Obtaining integrated UML Deployment diagram {

find node containing *clientComponent* and *serverComponent* respectively;

if *invocationType* is remote {

add *stub* and *invocationHandler_client* to node containing *clientComponent*;

add *invocationHandler_server* and *entry* to node containing *serverComponent;*}

add *instanceHandler* and *stateHandler* to node containing *serverComponent*;

For each *service* sub-element of *services* element

add component $service_{(i)}$ to node designated by *host* ;

}

Fig.4 Integrating middleware information into UMLs

5 Case study: modeling an Online Store

As an illustration of the proposed method, a case study was conducted, modeling the performance of an online store based on EJB container middleware. Fig.5-Fig.7 show the UML models of this case. The scenario can be described as follows: *Client* makes a remote synchronous invocation to *CustomerControlBean* to find the required customer information, in which need use middleware security service; and then updates email address of customer to *database*, in which need middleware transaction service supporting.

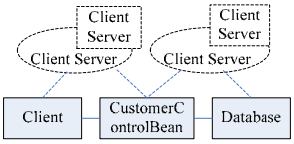


Fig. 5 UML Collaboration

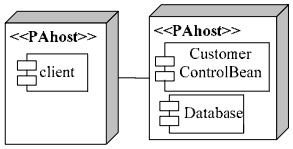


Fig. 7 Deployment diagram

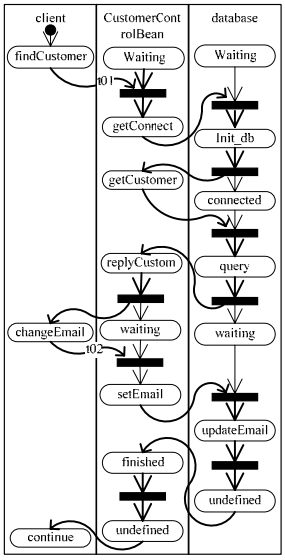


Fig.6 Activity diagram

According to the structure given in Fig.3, Fig.8 gives middleware usage file of the case. The performance estimates for the EJB container middleware components were obtained from measurement on a prototype implementation of the online store application based on a J2EE Application Server called OnceAS [17].

<componentInteractions clientComponent =Client serverComponent =CustomerControlBean>

<invocations>

<invocation transitionID=t01 invocationType=remote instanceHandling_time=0.8ms stateHandling_time=0.2ms>

<isRemote stub_Init_Time=3.2ms stub_req_Time=0.1ms marshall_Time=2.1ms unmarshall_Time=2.3ms parse_time=0.5ms stub_reply_Time=0.1ms entry_reply_Time=0.15ms/>

<services>

<service serviceName=SecService serviceType=Security host=serverNode serviceTie=3.3ms/>

</services>

</invocation>

<invocation transitionID=t2 invocationType=remote instanceHandling_time=0.8ms stateHandling_time=0.2ms>

```
    <isRemote          stub_req_Time=0.1ms          marshalltime=2.1ms          </services>
unmarshallTime=2.3ms     parseTime=0.5ms     stub_reply_Time=0.1ms          </invocation>
entry_reply_Time=0.15ms/>                                                  </invocations>
    <services>                                                          </componentInteractions>
        <service     serviceName=TxService     serviceType=Transaction
  host=serverNode   serviceTime=6.5ms/>
```

Fig.8 EJB middleware usage description file

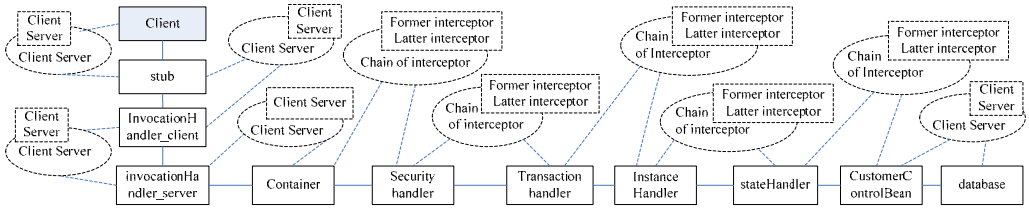


Fig9.Integrated structural and behavioral view

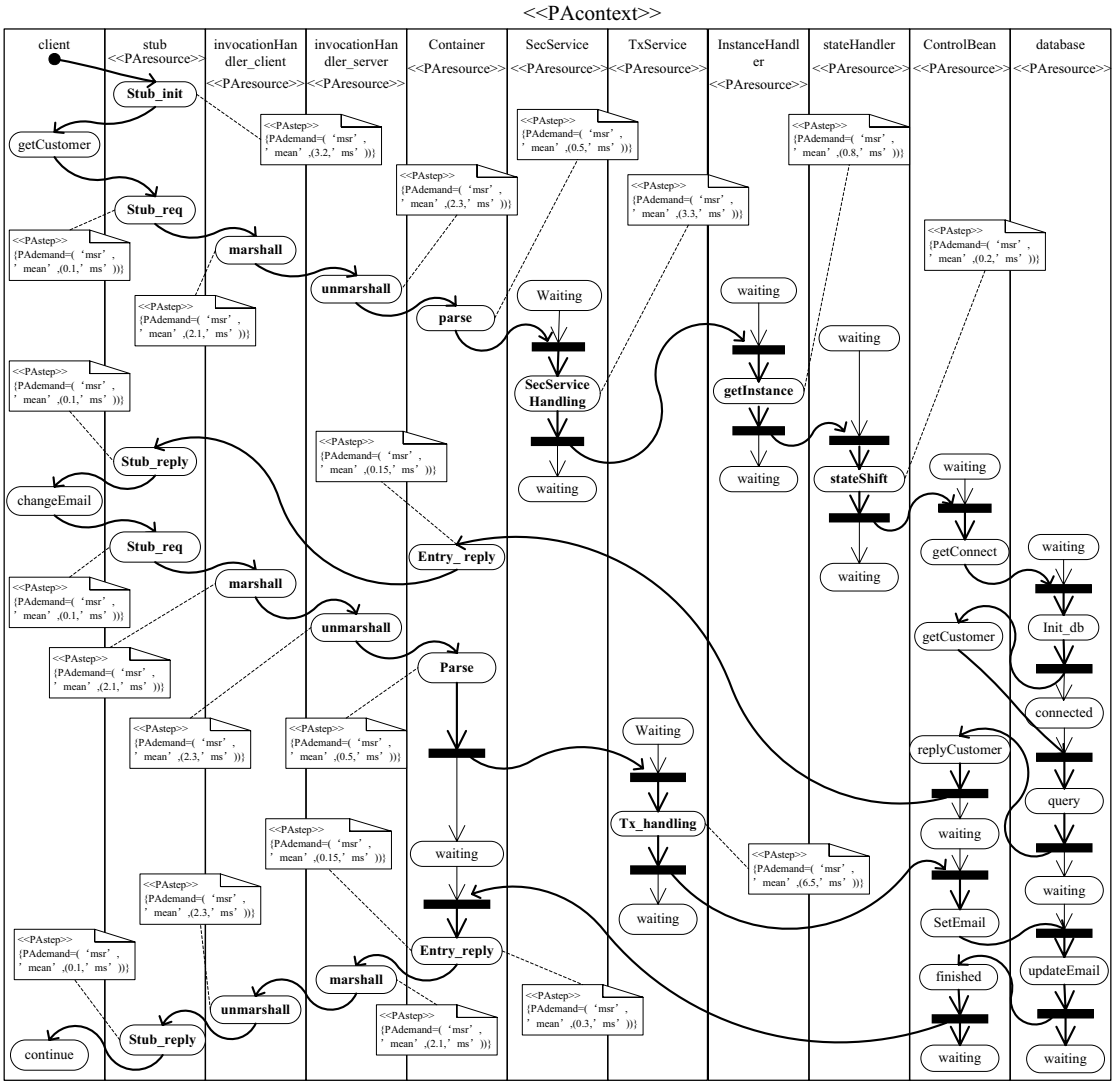


Fig.10 Integrated activity diagram annotated with performance information

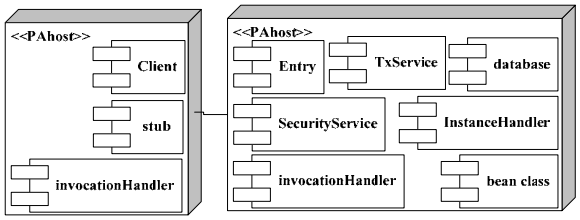


Fig11.Integrated deployment diagram

Applying the algorithm given in Fig.4, the resulting UML models are shown in Fig.9-Fig.11. Fig.9 shows integrated UML collaboration; Fig.10 shows integrated activity diagram annotated performance information with SPT profile. In this case, we take the response time index as illustration. The information is annotated by giving the actions the <PAstep> stereotype and specifying a tagged value PAdemand to represent execution time, which is provided in Fig.8. For clarity, the performance information of application components is omitted in diagram. Bold arcs represent the calling path after redirection. Fig.11 gives integrated deployment diagram.

Using the method proposed in [14, 15], the LQN performance model of online store can be derived from Fig.9-Fig.11. Then, performance estimates are extracted for varying system parameters by using the LQN model solver provided in [16]. To validate the presented performance model, we conducted measurements with our benchmark implementation. Fig.12 shows the response time of updating customer email information as a function of the total number of clients, including model prediction and experiment measurement result. We let the number of concurrent clients vary between 10 and 200 with the increment of 10 clients. Figure 12 indicates that the model is able to predict the response time of system reasonably well-the greatest difference between the measurement and the prediction is only 12%.

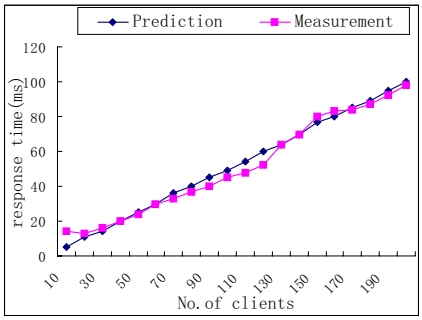


Fig.12 Comparison of prediction with measurement

6 Related work

To reflect the effect of middleware to system performance, one straightforward method is direct inclusion middleware components overhead into application components, for example, adding the overhead of stub creation and marshaling to business method execution demand. This coarse-grained dealing approach is simple and does not increase the complexity of performance model; however, the precision of model is not enough. Moreover, the resulting model cannot efficiently identify performance bottlenecks occurred in middleware layer.

Another method is directly modeling entire system including middleware and application. As in [10], author describes a framework for constructing LQN performance model based on the modular structure of Application Server and application components. In [5, 7, 8], authors model the performance for CORBA-based system using QN (Queueing Network model)/LQN formalism. Compared with the first method explained above, the method helps to improve the accuracy of performance model. However, it requires performance analyst to be familiar with internal details of middleware, which decreases the ease of use.

To predict the performance for CBS hosted by middleware infrastructure, in [18] authors propose a solution based on empirical testing and mathematical modeling. The models describe generic behaviors of application server components running on COTS middleware technologies, the parameters value in model are discovered through empirical testing. In this solution, incorporating application-specific behavior into the equation is difficult, and the results from the empirical testing cannot be generalized across different hardware and software platforms, so different platforms need different test cases, which is economically impractical.

In order to derive performance model from UML, a first approach based on architectural patterns for client/server systems is presented in [19]. The authors, rather than proposing a transformational methodology, describe the pattern through Class and Collaboration diagrams and directly show their corresponding EQN (Extended Queueing Network) models. The aim of our work mainly obtains integrated UML descriptions based on architectural patterns, then derives performance model from existing method automatically.

In [20], the authors propose automatic inclusion of middleware performance attributes into architectural UML software models, and a method based on Model Driven Architecture (MDA) that transform a middleware-independent UML model into a

middleware-aware UML model (effectively an MDA PIM-to-PSM transformation). Their idea is like ours. However, the transformation method of which is different. Our proposed method based on architectural pattern-based can be extended to deal with different style middleware. In addition, they mainly address the impact of remote invocation communication, and consider middleware services very simply; whereas ours emphasize the effect of middleware services and give the solution, besides remote invocation communication.

7 Conclusion

To reflect the performance effect of Container style middleware to CBS, this paper proposes an approach integrating middleware component interactions and performance attributes into application UML model, based on the architecture pattern adopted in Container middleware. Thus, derived performance model from the resulting UML models can efficiently represent the impact of Container middleware. A case study based on EJB container middleware has been described, and the comparison of model prediction with experiment measurement shows the efficiency of proposed approach. The integrating process can be executed automatically. And the approach can be extended to deal with different style middleware.

In the future work, we will deal with how to identify performance bottlenecks and choose proper configure parameters for Container middleware among alternatives to accomplish anticipated performance goal.

References

[1] Lloyd G. Williams and Connie U. Smith, "Performance Evaluation of Software Architecture", the Proc. Workshop Software and Performance, 1998.

[2] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, Marta Simeoni，"Model-Based Performance Prediction in Software Development: A Survey", IEEE Transactions on Software Engineering, 2004, Volume 30, Issue 5, pp. 295 – 310.

[3] Wolfgang Emmerich, "Software engineering and middleware: a roadmap", Proceedings of the International Conference on Software Engineering, on The Future of Software Engineering, 2000.

[4] Murray Woodside, Dorin Petriu, Khalid Siddiqui,"Performance-related Completions for Software Specifications", the Proc. of International Conference on Software Engineering, 2001.

[5] P. Kahkipuro, "Performance Modeling Framework for CORBA Based Distributed Systems," PhD thesis, 2000.

[6] D. Petriu, H. Amer, S. Majumdar, and I. Abdul-Fatah, "Using Analytic Models for Predicting Middleware Performance," Proc. Second Int'l Workshop Software and Performance, pp. 189-194, Sept.2000.

[7] T. Verdickt, B. Dhoedt, F. Gielen, and P. Demeester, "Modeling the Performance of CORBA Using Layered Queueing Networks," Proc. 29th Euromicro Conf., pp. 117-123, 2003.

[8] C.U. Smith and L.G. Williams, "Performance Engineering Models of CORBA-Based Distributed-Object Systems," Proc. Computer Measurement Group Conf., pp. 886-898, Dec. 1998.

[9] S. Chen, Y. Liu, I. Gorton, and A. Liu, "Performance Prediction of Component-Based Applications," J. Systems and Software, vol. 74, no. 1, pp35-43, 2005.

[10] Jing Xu, A. Oufimtsev, M. Woodside, L. Murphy, "Performance Modeling and Prediction of Enterprise JavaBeans with Layered Queuing Network Templates", Proc. Workshop on Specification and Verification of Component-Based Systems, 2005.

[11] F.Buchmann, R.Meunier, H.Rohnert, P. Sommerland, and M.Stal, Pattern-Oriented Software Architecture: A Survey of Patterns. Wiley Computer Publishing, 1996.

[12] Object Management Group, "UML Specification 1.4".

[13] Object Management Group, "UML Profile for Schedulability, Performance, and Time," Apr. 2003.

[14] D.C.Petriu and X.Wang," From UML Description of High-level Software Architectures to LQN Performance Models," Proc. Applications of Graph Transformations with Industrial Relevance Workshop, pp.47-62, 1999.

[15] D.C. Petriu and H. Shen, "Applying the UML Performance Profile: Graph Grammar-Based Derivation of LQN Models from UML Specifications," Proc. 12th Int'l Conf. Computer Performance Evaluation, Modeling Techniques and Tools (TOOLS 2002), pp.159-177.

[16] Franks,G.,Hubbard,A.,Majumdar,S.,Petriu,D.C.,Rolia,J.,W oodside,C.M, A toolset for Performance Engineering and Software Design of Client-Server Systems. Performance Evaluation, Vol.24, Nb.1-2(1995), pp.117-135

[17] http://www.once.com.cn

[18] Chen, S., Gorton, I., Liu, A., Liu, Y., 2002. Performance prediction of COTS component-based enterprise applications. In: Proceedings of 5th International Conference on Software Engineering Workshop on Component-Based Software Engineering: Benchmarks for Predictable Assembly.

[19] H.Gomaa and D. Menasce,"Performance Engineering of Component-Based Distributed Software Systems", Performance Engineering, R.Dumke et al., pp.40-55,2001

[20] Tom Verdickt, Bart Dhoedt, Frank Gielen,and Piet Demeester, "Automatic Inclusion of Middleware Performance Attributes into Architectural UML Software Models", IEEE Transactions on Software Engineering, Vol. 31,No.8,2005.

Two Perspectives on Open-Source Software Evolution: Maintenance and Reuse

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

Kai Chen
Institute for Software Integrated Systems,
Vanderbilt University
Nashville, TN, USA
kai.chen@vanderbilt.edu

Abstract

After a software system has been delivered, it inevitably has to change to remain useful. This is called software evolution. In most cases, software evolution is referred as software maintenance, which means a software evolution process is considered as a software maintenance process. On the other hand, software reuse also frequently occurs in software evolution. More or less of a software product needs to be reused in software evolution. Therefore, a software evolution process is also a reuse process. However, the reuse process is largely ignored, especially in the open-source software study. In this paper, we study open-source software evolution from both the maintenance and the reuse perspective, and we propose a maintenance-based evolution model and a reuse-based evolution model for open-source software. These models describe the evolution of an individual component and can help us to identify reusable open-source components and improve their reusability. We then study the evolution of twelve kernel components of Linux by using these two evolution models.

1. Introduction

Software evolution [1] [2] is a process that a software system changes from a simpler or worse state to a higher or better state [3]. Software evolution is inevitable, because changes need to be made on it to fix defects or to satisfy the new requirements. Traditionally, and it is still commonly agreed on, that the evolution process of a software system is the maintenance process of the system. On the other hand, software reuse always occurs in software evolution, which could be an ad hoc small scale reuse or a systematic large scale reuse. Therefore, software evolution can also be considered as a reuse process.

Traditionally, the reuse process and the maintenance process are considered as two different processes. The reuse process is usually considered to occur in software development. Reusable components are identified and adapted to the new system. The maintenance process is considered to occur in software evolution. Any changes to the software product after it is delivered are considered as maintenance. Therefore, for the development-then-evolution model of closed-source software, reuse process and maintenance process can be largely viewed as two independent processes.

Open-source software is characterized by the philosophy, "release early and release often" [4]. Usually, the documentation for open-source software on requirement, analysis, and design is limit and incomplete [5]. The evolution process for open-source software is different from closed-source software. A lot of research has performed to study the evolution patterns of open-source software [6] [7]. Because new versions of open-source software are released frequently, an evolution process is also a development process. In the development-and-evolution model of open-source software, the software reuse is mixed with the software maintenance. Therefore, to view software evolution as a reuse process is especially important for the open-source software research.

In this paper, we study open-source software evolution from both the maintenance and the reuse perspective. The remainder of the paper is organized as follows: Section 2 describes the related work of other researchers on incorporating maintenance and reuse in software evolution. We describe our maintenance-based and reuse-based evolution model in Section 3. Section 4 presents our case study on Linux. Our conclusions appear in Section 5.

2. Related work

Software evolution is a process of making changes to software artifacts, while preserving the relationships between those artifacts. Researchers have noticed the existence of both a maintenance process and a reuse process in software development and evolution. Basili [8] proposed three maintenance process models: the quick-fix model, the iterative-enhancement model, and the full reuse-model. All these models consider the software reuse and so are reuse-oriented software development models.

However, these models require complete documentation of requirement, analysis and design which is a property for closed-source software. The full-reuse model shown in Figure 1 is used as an example. It first extracts reusable artifacts including requirement, analysis, design, code and test, and saves them in the repository. Then, the development of the new system is performed by identifying the artifacts and reusing them. In his research, Basili viewed maintenance as reuse-oriented software development. However, his models could not fit to open-source software. Because, (1) open-source software does not have complete documents on requirement, analysis, design, and systematic test; (2) open-source software does not promote planned reuse, there is no artifact repository for open-source software.

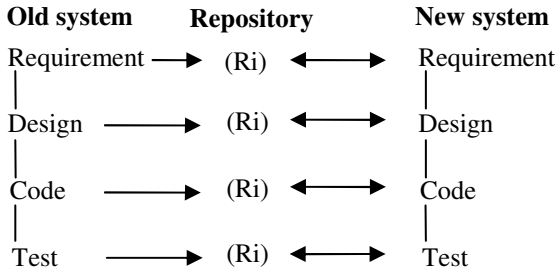


Figure 1. Full-Reuse model from Basili [8]

Another research that incorporates maintenance and reuse in software evolution is done by Schach and Tomer [9] [10]. In their series publications, they built a three dimensional development, maintenance, and reuse model for software product line as shown in Figure 2. The core assets as well as the product are developed, reused, and updated (maintained) in the product line process. The three dimensional model represents the concurrency of three activities. Significant research has been done to study software reuse alternatives based on this model [11]. However, this model cannot be applied to open-source software. It is built on product line software, while most open-source software systems are not.

Our literature review shows that previous researches on incorporating maintenance and reuse in software evolution are limited to closed-source software. There is a need to study this issue in open-source software.

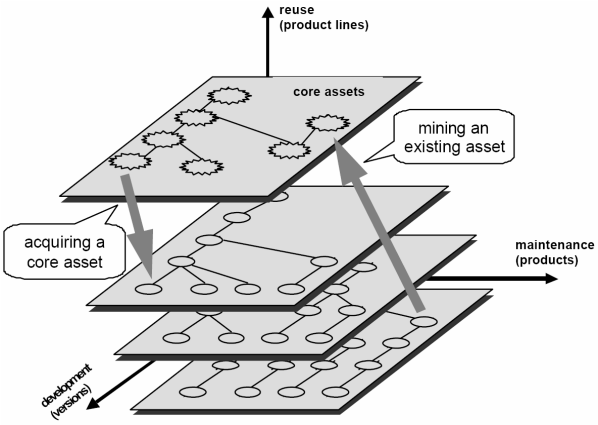


Figure 2. Software product line Evolution [10]

3. Evolution model for open-source software

As we mentioned before, open-source software is characterized by its frequent release property. Many versions exist for one open-source software system. Each new released version is based on a previous version. If we consider the evolution of the entire open-source software system, we can model it as an evolutionary streamline. Suppose an open-source software product consists of nine consecutive versions, V1 through V9, its evolution can be represented as a one-dimensional streamline, which is shown in Figure 3. (To simplify the problem, in this paper, we ignore the branching and merging of software versions, but only consider consecutive versions.)

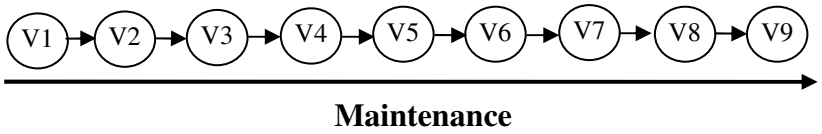


Figure 3. One-dimensional evolution process for open-source software

Figure 3 shows the evolution of the entire software system from the perspective of maintenance. Every new release (version) is based on the modification of the previous version. This is the widely accepted view of open-source software evolution. More complete model may include version branching and merging, which will be a two-dimensional tree structure. However, both the one-dimensional streamline model and the two-dimensional tree model are maintenance-based and the reuse process is largely ignored.

Figure 3 shows the evolution of the product as a whole. However, different components within the product may evolve differently. For example, some components may never been changed while others may be changed many times. Our study of open-source software evolution is component-based, which means we are more interested in the evolution of individual software components.

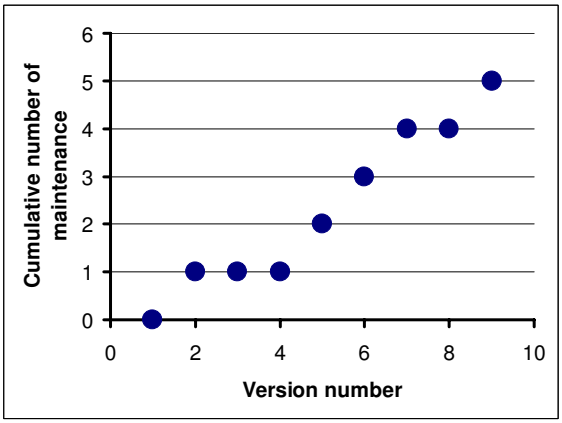


Figure 4. The maintenance-based evolution model for component C1 in nine versions of the product

Figure 4 shows our maintenance-based evolution model for a single component C1. The horizontal axis represents the version of the product. The vertical axis represents the cumulative number of maintenance, which sums up the total number of maintenance from the first version to current version.

Software reuse always happens in software evolution. Software system consists of a collection of components. In each release, some components are modified, others may remain unchanged. To study software evolution from the perspective of reuse, we propose a reuse-based evolution model. In the model, the reuse of software components is classified into two categories: white-box reuse and black-box reuse. In one software evolution step, which is marked by a new version release, the components that are modified in the new release are classified as white-box reuse and those that are unchanged are classified as black-box reuse. Therefore, the evolution of a component in the software lifetime can be represented as a trace of a two-dimensional diagram. Figure 5 shows the evolution of a component C1 in nine consecutive versions of the software. The value in the vertical axis indicates how many times that a component is black-box reused, while the value in the horizontal axis gives times that a component is white-box reused in the research period.

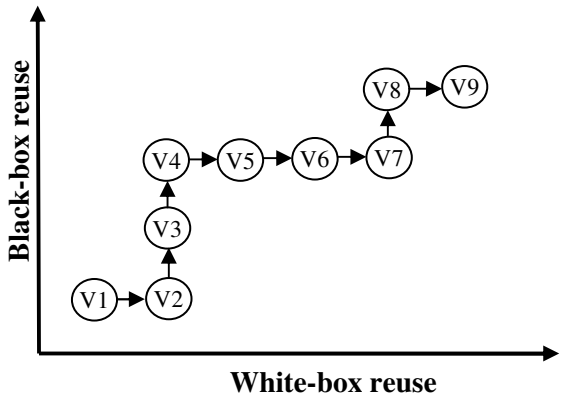


Figure 5. The reuse-based evolution model for component C1 in nine versions of the product

Different components in a product may be modified in different evolution steps. Accordingly, they should have different evolution traces. The reuse-based evolution model depicts the evolution trace of each specific component in the product instead of the entire software product. Comparing to the maintenance-based model, the reuse-based model has the following advantages: (1) the steepness of a component trace indicates the potential that this component can be reused without change. Therefore, by studying the reuse-based evolution of software components, we can identify the most reusable components; (2) reuse-based evolution model incorporates the maintenance process, since white-box reuse of a component also represents the maintenance history of a component.

The following part of this paper presents a case study, in which we study the evolution of Linux by applying our maintenance-based and reuse-based evolution models.

4. The evolution of Linux components

Linux is one of the most active open-source projects. It has released about 600 versions. A lot of

research [4] [7] has performed on the evolution pattern of Linux, including the growth of size, changes of kernel structure, and community properties.

From version 1.0.0 to version 2.6.11, there are 597 formal releases of Linux. If considered as a maintenance process, the evolution of the whole Linux product can be represented as a streamline with 597 nodes (again, we ignore the branching and the merging). Due to the simplicity of this streamline structure, it does not provide too much information about the evolution pattern of Linux. Therefore, we choose to study the evolution of components in Linux. The most important components in Linux are kernel modules (components), which contain the most important functions of the operating system and are architecture independent.

The Linux 1.0.0 version has 23 kernel modules. In the evolution process, some kernel modules are removed, some new modules are added. In version 2.6.11, there are 58 kernel modules. In these 58 kernel modules, only 12 modules continuously evolve from version 1.0.0. This means 12 kernel modules experienced the entire evolution of Linux. Therefore, we decided to study the evolution of these 12 kernel modules in 597 releases.

The research was performed by studying the entire evolution of Linux. For the simplicity, the 576 versions are indexed from 1 to 576, which are shown in Table 1.

Table 1. The version index of Linux

Version	1.0	1.1	1.2	1.3	2.0	2.1	2.2	2.3	2.4	2.5	2.6
Index	1-12	13-108	109-122	123-223	124-264	265-397	398-424	425-477	478-509	510-585	586-597

First, we build the maintenance-based evolution models for 12 kernel modules as shown in Figure 6, which depicts the cumulative number of maintenance for each module. For example, the figure shows that, in the earlier release, more times of changes are made to *time.c* than to *printk.c*, while in the recent release, more times of changes are made to *printk.c* than to *time.c*. The pitch of the line indicates the frequency of modification to a module. Also, we can see from Figure 6 that the number of changes made to these 12 kernel modules is very different, with *dma.c* has the least numbers of changes and *sched.c* has the largest number.

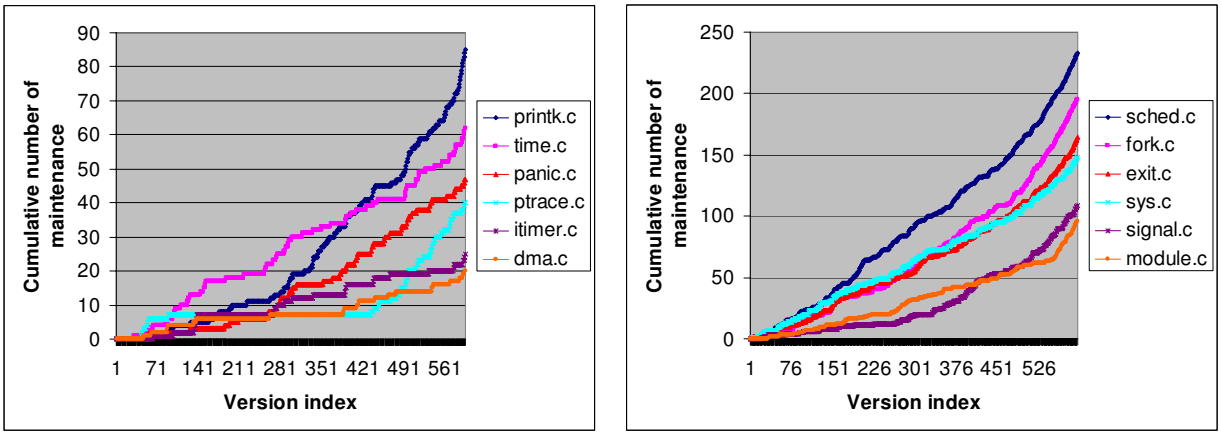


Figure 6: The maintenance-based evolution models for 12 Linux kernel modules [12]

Table 2: The number of white-box reuse and black-box reuse of 12 kernel modules

Number of	m1	m2	m3	m4	m5	m6	m7	m8	m9	m10	m11	m12
White-box reuse	21	165	196	26	97	48	86	41	234	110	149	63
Black-Box reuse	576	432	401	571	500	549	511	556	363	487	448	534

Key to modules: *m1: dma.cl; m2: exit.c; m3: fork.c; m4: itimer.c; m5: module.c; m6: panic.c; m7: ptrintk.c; m8: ptrace.c; m9: sched.c; m: signal.c; m11: sys.c; m12: time.c*

Next, we consider the evolution of these 12 kernel modules from the perspective of reuse. Table 2 shows the number of white-box reuse and black-box reuse of the 12 kernel modules in 597 releases. It shows that *dma.c* has the smallest number of white-box reuse and *sched.c* has the largest number of white-box reuse. This can also be deduced from Figure 6, because the number of white-box reuse equals the number of cumulative maintenance for a specific component.

Figure 7 shows the evolution trace of the 12 kernel modules under the reuse-based evolution model (Figure 5) described in Section 3. From the figure, we find that the traces of *dma.c, itimer.c, ptrace.c* and *time.c* have longer vertical paths than the others, which means they have more times of black-box reused in the past, and it is also reasonable to think that they also have more potential to be black-box reused in the future.

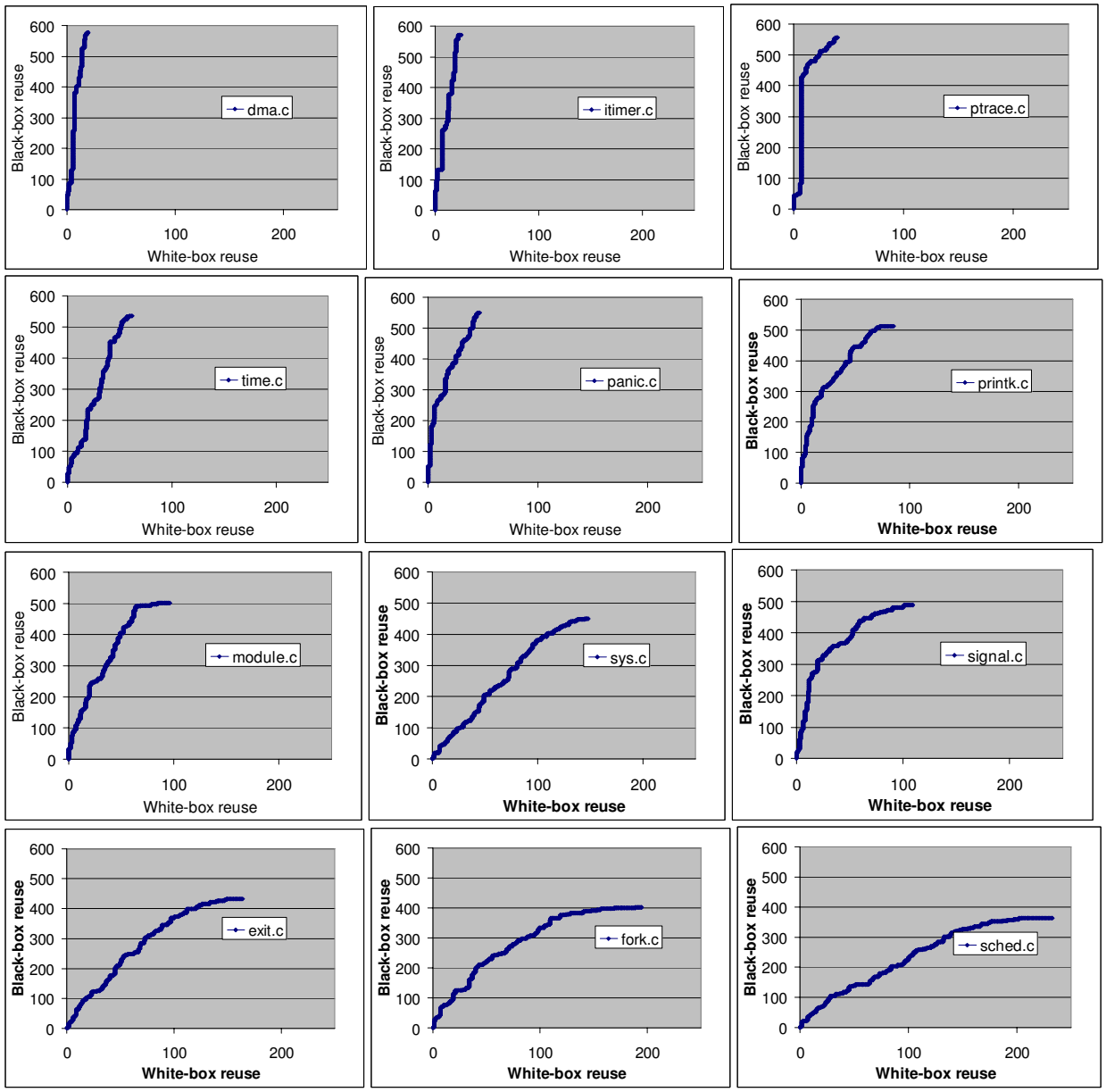


Figure 7: The evolution of 12 kernel modules from the perspective of reuse

Although we can derive the similar information from Figure 6 as from Figure 7, the reuse-base evolution model is more understandable. We can see from Figure 7 that, the recent versions of *fork.c* and *sched.c* are more frequently white-box reused than black-box reused. As we mentioned before, kernel modules in Linux are most important and architecture independent. They should be designed with more reuse potential. According to our study of 12 kernel modules which experienced the entire Linux evolution, different modules have different reuse history. This difference may be due to the reusability of these kernel modules.

More efforts are needed for white-box reuse than for black-box reuse. To reduce the reuse effort, black-box reuse is preferable over white-box reuse. Therefore, in order to improve the evolution process and reduce the reuse effort of Linux, modules such as *exit.c*, *fork.c*, and *sched.c* may need to be redesigned to support more black-box reuse.

This study was performed on Linux kernel modules (components). Similar study could be performed on other components of Linux and other open-source software systems. The models presented in this paper are simple yet powerful. It is easy to extract data for the models directly by using version control software or script programming.

5. Conclusions

Reuse-oriented development is becoming more and more common in closed-source software. Unfortunately, same as closed-source software, its reusable components are also considered private properties and are not accessible to outsiders. The widely available open-source software contains huge amount of software components and are free to use.

In this study, to address the evolution of open-source software, we studied its evolution from both the maintenance and reuse perspective. We proposed a maintenance-based evolutional model and a reuse-based evolution model, and concluded that the information contained in the first model can be covered by the second model. The reuse-based evolution model can help to identify reusable components and improve the software reuse process. This is our first step in identifying reusable open-source components. Future research will study the evolution dependencies of open-source components in order to identify large-scale reusable component clusters.

6. References

[1] M. M. Lehman, “Life Cycles and Laws of Software Evolution”, *Proceedings of IEEE (Special Issue on Software Engineering)*, 1980, pp. 1060–1076.

[2] D. E. Perry, “Dimensions of Software Evolution”, *Proceedings of International Conference on Software Maintenance*, Sorrento, Italy, 1994, pp. 296–303.

[3] L.J. Arthur, *Software Evolution: the Software Maintenance Challenge*, John Wiley and Sons, New York, NY, USA, 1988.

[4] E. S. Raymond, *The Cathedral & the Bazaar*, First Edition, O’Reilly, February 2001.

[5] K. Chen, S. R. Schach, L. Yu, J. Offutt, and G. Z. Heller, “Open-Source Change Logs,” *Empirical Software Engineering*, vol. 9, no. 3, 2004, pp. 197–210.

[6] M. W. Godfrey and Q. Tu, “Evolution in Open Source Software: A Case Study”, *Proceedings of International Conference on Software Maintenance*, San Jose, CA, 2000 pp. 131–142.

[7] S. R. Schach, B. Jin, D. R. Wright, G. Z. Heller, and A. J. Offutt, "Maintainability of the Linux Kernel", *IEE Proceedings—Software,* 149, 2002, pp. 18-23.

[8] V.R. Basili, “Viewing Maintenance as Reuse-Oriented Software Development”, *IEEE Software*, vol. 7, no. 1, 1990, pp. 19-25.

[9] A. Tomer and S.R. Schach, “The Evolution Tree: A Maintenance-Oriented Software Development Model”, *Proceedings of the Fourth European Conference on Software Maintenance and Reengineering*, Zurich, Switzerland, February/March 2000, pp. 209–214.

[10] S. R. Schach and A. Tomer, “Development/ Maintenance/Reuse: Software Evolution in Product Lines”, *Proceedings of the First Software Product Lines Conference*, Denver, USA, August, 2000, pp. 437–450.

[11] A. Tomer, L. Goldin, T. Kuflik, E. Kimchi, and S. R. Schach, “Evaluating Software Reuse Alternatives: A Model and its Application to an Industrial Case Study”, *IEEE Transactions on Software Engineering,* vol. 30, no. 9, 2004, pp. 601–612.

[12] L. Yu, “Understanding Component Co-evolution with a Study on Linux”, *Empirical Software Engineering*, 2006.

Reviewers' Index

A

Silvia Abrahão
Silvia Teresita Acuña
Anneliese Andrews
Carmelo Ardito
Juan Carlos Augusto
Mikhail Auguston

B

Doo-Hwan Bae
Xiaoying Bai
Maria Teresa Baldassarre
Luciano Barezi
F. B. Bastani
Fevzi Belli
Michael Berger
Sami Beydeda
Pankaj Bhawnani
Alessandro Bianchi
Cornelia Boldyreff
Jean-Michel Bruel
Lei Bu

C

Danilo Caivano
Gerardo Canfora
Joao W. Cangussu
Giovanni Cantone
Heeseo Chae
Christine W. Chan
W.K. Chan
Ned Chapin
Mei-Hwa Chen
William Chu
Peter Clarke
Panos Constantopoulos
Kendra Cooper
Massimo Cossentino
Maria Francesca Costabile

D

Yi Deng
Deepak Dhungana
Oscar Dieste
Damiano Distante
Jin Song Dong
Jing Dong
Vicky Dritsou

F

Davide Falessi
Fausto Fasano
Yuzhang Feng
Marta Lopez Fernandez
Eduardo Fernandez-Medina
Xavier Ferre

G

Felix Garcia
Carlo Ghezzi
Holger Giese
Sandra Gómez
Des Greer
Eric Gregoire
Volker Gruhn
Paul Grunbacher
Sheng-Uei Guan

H

Syed Waseem Haider
Ngai Cheuk Han
Jingsha He
Xudong He
Mei Hong
Hai Hu
Gang Huang
Gordon Huang
Ying Huang

I

Ricardo Imbert
Peter In

J

Sang-Uk Jeon
Wenpin Jiao
Zhi Jin
Dehua Ju
Natalia Juristo

K

Yoshiaki Kakuda
Gabor Karsai
Jun Kong
Alfred H. Kromholz

L

Rosa Lanzilotti
Hee-Jin Lee
Bin Lei
Tao Li
Yuan Fang Li
Huimin Lin
Zhenjiang Lin
Jiming Liu
Jian Lu
Zhongyu (Joan) Lu
Giuseppe A. Di Lucca
Carlos José Pereira de Lucena
Michael R. Lyu

M

Neil Maiden
Antonio Maña
Xinjun Mao
Tegegne Marew
Sergio Di Martino
Jim McElroy
Nelson Medinilla
Hong Mei
Daniel Mellado
Rym Mili
Abdallah S. Mohamed
Sandro Morasca
Ana M. Moreno
Dante Carrizo Moreno
Antonio Munoz

N

Michele Nappi
Elisabetta Di Nitto

O

Rocco Oliveti
Mehmet Orgun

P

Chris Pantazis
Buono Paolo
Georgios Papaioannou
Manish Parashar
Seung-Hun Park
Massimiliano Di Penta
Mario Piattini
Antonio Piccinno
Macario Polo

R

Rick Rabiser
David Riaño
Steve Roach
Guenther Ruhe
Francisco Ruiz

S

Omolade Saliu
Francisco Sanchez-Cid
Maria-Isabel Sanchez-Segura
Salvatore Alessandro Sarcià
Beata Sarna-Starosta

Walt Scacchi
Sahra Sedigh-Ali
Daniel Serrano
Norbert Seyff
Tony Shan
Yidong Shen
Tianjun Shi
Martin Solari
George Spanoudakis
Nenad Stankovic
Kurt Stirewalt
Giancarlo Succi

T

Philip S. Taylor
Paitoon Tontiwachwuthikul
Genny Tortora
Jeffrey Tsai
T. H. Tse

U

M. Irfan Ullah
Rainer Unland

V

Sira Vegas
Giuseppe Vissagio

W

Chan Pik Wah
Qianxiang Wang
Christiane Gresse von Wangenheim
Victor Winter
Guido Wirtz
Claes Wohlin
Eric Wong
Y. M. Wong
Gangshan Wu
Ye Wu

X

Liang Xiao
Bing Xie
Fei Xing
Baowen Xu
Dianxiang Xu

Y

Hongji Yang
Sheng Yang
Kyung-A Yoon
Huiqun Yu

Z

Vassilis Zafeiris
Du Zhang
Kang Zhang
Lu Zhang
Wenhui Zhang
Haiyan Zhao
Jianjun Zhao
Yangfan Zhou
Hong Zhu
Jianke Zhu
Eugenio Zimeo
Andrea Zisman

Authors' Index

E

F

G

H

J

K

L

M

N

O

SEKE 2007 CALL FOR PAPERS

The Nineteenth International Conference on Software Engineering and Knowledge Engineering

Hyatt Harborside at Logan Int'l Airport, Boston, USA
July 9 - July 11, 2007

Organized by

Knowledge Systems Institute Graduate School

The Nineteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'07) will be held in Hyatt Harborside at Boston's Logan Int'l Airport, Boston, USA, July 9-11, 2007. The hotel's website is at:
http://harborside.hyatt.com/hyatt/hotels/index.jsp
Hotel address: 101 Harbor Dr., Boston, Massachusetts, USA 02128
Phone: 617-568-1234 or 1-800-233-1234 Fax: 617-567-8856

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:
Adaptive Systems
Artificial Intelligence Approaches to Software Engineering
Automated Reasoning
Automated Software Design and Synthesis
Automated Software Specification
Component-Based Software Engineering
Computer-Supported Cooperative Work
Databases
Design Methods
Embedded and Ubiquitous Software Engineering
Education and Training
Electronic Commerce
Formal Methods
Human-Computer Interaction
Industrial Applications
Integrity, Security, and Fault Tolerance
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
Meta-CASE
Mobile Data Accesses
Multimedia and Hypermedia Software Engineering
Object-Oriented Technology
Ontologies and Methodologies
Patterns and Frameworks
Process and Workflow Management
Programming Languages and Software Engineering
Program Understanding
Reflection and Metadata Approaches
Reliability
Requirements Engineering
Reverse Engineering
Soft Computing
Soft Media/Digital Media
Software Architecture
Software Domain Modeling and Meta-Modeling
Software Engineering Decision Support
Software Maintenance and Evolution
Software Process Modeling
Software Quality
Software Reuse
System Applications and Experience
Time and Knowledge Management
Tools
Tutoring, Help, Documentation Systems
Uncertainty Knowledge Management
Validation and Verification
Web-Based Knowledge Management
Web-Based Tools, Systems, and Environments
Web and Data Mining

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/seke07/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'07 will be reviewed electronically. The users (webmaster, program chair, reviewers...) can login using the following URL: http://conf.ksi.edu/seke07/review/pass.php.

If you have any questions or run into problems, please send e-mail to: seke@ksi.edu.

SEKE'2007 Conference Secretariat
Knowledge Systems Institute Graduate School
3420 Main Street
Skokie, IL 60076, USA
Tel: +1-847-679-3135
Fax: +1-847-679-3166
E-mail: seke@ksi.edu

IMPORTANT DATES

February 20, 2007	*Paper submission due*
April 1, 2007	*Notification of acceptance*
May 1, 2007	*Camera-ready copy*