A Lightweight Formalism for the Integration of BPMN Models with Domain Ontologies

Giuseppe Della Penna[1], Roberto Del Sordo[3], Benedetto Intrigila[2], Nicoló Mezzopera[3], and Maria Teresa Pazienza[2]

[1] Dept. of Engineering, Computer Science and Mathematics, University of L'Aquila, L'Aquila, Italy
[2] Dept. of Enterprise Engineering, University of Rome Tor Vergata, Rome, Italy
[3] SinergieIT srl, Rome, Italy

Abstract. The widespread use of BPMN to describe business processes is highlighting the need to integrate the description of the operational flow with domain specific information. In most cases the domain knowledge is already represented in domain ontologies or can be derived from the existing documentation. To preserve the simplicity of the original BPMN model specifications our approach is to integrate the semantic and BPMN information through a separate view that can be semi-automatically built relying on the capabilities of the SyBeL modeling language. This unified view can be used to produce different artifacts to support the implementation phases of the executable process, while keeping track of the formal specifications.

Keywords: Ontology; BPMN; Modeling Formal Language

1 Introduction

The Business Process Modeling Notation (BPMN) is the new standard to model business process flows and web services [8]. Created by the Business Process Management Initiative (BPMI), the first goal of BPMN is to provide a notation that is readily understandable by all business users.

In particular, the business analysts can directly operate with the BPMN to create the initial drafts of the new processes to be realized. Therefore their ideas can easily and precisely conveyed to the technical developers responsible for implementing the technology that will perform those processes. It is the mentioned combination of easiness of use with the precision of a formally well defined notation system which is responsible for the enormous success of BPMN. [2]

Being a formalism for the modelization of *processes*, BPMN gives a limited support to the *modelization of data*. This is not an issue when the data models needed to implement the processes can be routinely constructed from a direct analysis of the business components involved. Here we focus on software projects that require a deeper *semantic* understanding of complex aspects of the domain.

In many cases, the required semantic information can be obtained from previously defined ontologies (see, e.g., [11]). In other cases, one has to rely on documents. However existing tools, such as Semantic Turkey [9], are making easier and easier the task of formalizing into ontologies the knowledge dispersed in the documents. Semantic Turkey is a free open-source platform for Semantic Bookmarking and Ontology Development.

Users can adopt Semantic Turkey to keep track of relevant information from textual documents and organize collected content according to imported/personally edited ontologies. Domain experts and ontology developers can build ontologies starting from the very raw source of information which they find on the web, without any need of interconnecting different heterogeneous tools and applications.

Therefore it is reasonable to have a formal ontology as a source of the needed knowledge about the domain. Now the problem arises on how to combine the information formalized by means of ontological descriptions with the BPMN graphical models. To this aim several approaches are possible. The most direct one is to extend the BPMN formalism to allow semantic annotations [10, 1]. This solution has the advantage of a more complete modelization at the cost of the complexity of building up a unique comprehensive model.

Here we want to explore a different approach, which is directed towards the implementation layer. We use the SyBeL formalism [3] to suitably merge the process and ontological information in a single, *integrated view*. With this approach, we do not modify or substitute any of the source formalisms (i.e., ontologies and BPMN descriptions), but only create an intermediate view which is useful to drive the implementation without loosing traceability with respect to the original specifications.

Indeed, SyBeL has been devised to support the modelization of software systems with a strong focus on the behavioural aspects and a limited but completely formal support for the modelization of data. These characteristics (that we synthetically expose in the next section) allow, on one hand, a direct translation of the BPMN models and, on the other, the extraction of limited data models from the ontology level. In our opinion, such a compromise is suitable when the implementation does not require a relevant modelization effort but there is still the need of a good documentation of the system for an appropriate maintenance of the system itself.

2 The SyBeL Modeling Language

SyBeL is an XML based language to specify the overall behaviour of a software system. Thanks to underlying XML technologies, SyBeL can be read and written by humans, but it also represents a formal, unambiguous specification that can be easily manipulated by a machine. In this way, SyBeL tries to be a good compromise between natural language descriptions and formal, compilable specifications, i.e., it preserves the advantages provided by a natural language description, while it is also formal enough to avoid ambiguity and allow a systematic reuse of the specifications throughout different phases of the software development process.

In particular, SyBeL includes a *behaviour description language*, which allows to encode dynamic processes as flows of user-triggered and system-generated events called *interactions* and *happenings*, respectively. Each event is defined in terms of an action executed on a particular entity, and variables can be used to capture action results and reuse them later, or to declare flow parameters. Finally, flows can contain conditional branches (*flow alternatives*) and unconditioned jump instructions to resume the execution from an arbitrary step of the flow.

Therefore, it is clear that SyBeL behaviours can easily encode any BPMN-defined process. The BPMN-to-SyBeL translation is very straightforward and can be performed without loss of information, thus SyBeL behaviours can be also used to regenerate the corresponding BPMN process description.

SyBeL also allows to declare data structures and types using the SyBeL *domain description language.* The language defines five basic simple types (string, number, boolean, binary and any) and two type constructors, i.e., *collections*, which represent lists of recursively typed objects, and *entity types*, that are sets of recursively typed properties and (possibly parametric) actions. Entity types can be also derived from other entity types in a object-oriented fashion. These features together create a quite complete but still abstract type system, allowing to declare types and data structures which can be easily translated in a variety of object-oriented languages. However the SyBeL type system, especially entity types, was also designed with semantics in mind. Indeed, SyBeL entities can be seen as a simplified, programmer-friendly view of ontologies, whose attributes and relations are encoded using properties, actions and type derivations. This semantic connection is made explicit by the modelReference attribute, which can be used to annotate entity types as well as single properties and actions with resource identifiers referencing elements of a semantic model.

Therefore, we can use the SyBeL domain language to extract suitable “views” from the information given by the ontologies, generating data structures with semantic annotations which encode the aspects most relevant to the implementation and are then used to give a type to any element (entity, variable, parameter, etc) referenced in the process description.

Putting all together, SyBeL can be used to fill the gap between BPMN-specified business processes, which describe the dynamic evolution of a system without data types, and ontologies, which in turn are well suited to describe data structures, but only statically. Moreover, by maintaining references to the original BPMN and ontologic sources, SyBeL does not try to substitute such formalisms, but only to offer an unified view which is useful both for documentation purposes and, most notably, to drive the implementation without loosing traceability with respect to the formal specifications.

3 Case Study

To explain our approach, we consider a case study which is of relevant interest in its own right, the *Alternative Dispute Resolution* (ADR) system, also called *Mediation.* Mediation is a system of resolution of disputes under civil and commercial law which is increasingly adopted to facilitate access to justice: here we refer to European and Italian legal systems ([7] and, respectively, [4]).

In the mediation a third party, called the *mediator*, assists the parties to negotiate a settlement. Mediation process is applicable to disputes in a variety of domains, such as commercial, legal, workplace, community and family matters. Mediation can also be seen as a powerful mechanism of eGovernment as a large part of the mediation process can be performed on the Web. It is clear that the software implementation of such an important system is *critical* with respect to several aspects, e.g., security, availability and reliability.

For space reasons, we consider here only the case of *condominium disputes.* It is nevertheless a very important category, in particular in Nova Scotia (Canada) there exists a specific ADR system for such kind of disputes [5] and in Italy there is an estimated number of one million condominium disputes [6].

Also for space reasons, we limit ourselves to consider the subprocess of the initial phase of the mediation request. The main actors involved in the subprocess are the following:

– Proposer: the specific party, between the parties involved in the legal dispute, that requires the mediation.
– Mediator: any third person who is asked to conduct a mediation in an effective, impartial and competent way.
– Coordinator: in the Italian mediation system the person who assesses the mediation request and assigns it to a mediator.

The BPMN diagram of the subprocess is illustrated in Figure 1.

3.1 Ontological Aspects

A simple analysis of the starting subprocess shows that the only critical aspect is the description of the object of the dispute. Indeed, in an eGovernment approach, we cannot assume that the parties are legal experts so that they are able to insert an appropriate description and - what is more relevant - an appropriate documentation. The risk is that the mediation request is poorly formulated so that a lengthly and costly process of revision is needed to arrive to a decision. Therefore, a valid system support is needed to ensure that the correct documentation is enclosed to the request. It is natural to base such support on an ontological description of the condominium disputes able to relate the different types of disputes with the appropriate evidence required for the mediator decision. A suitably simplified version of such an ontology, inspired by [6], is sketched in Figure 2. So, for instance, an aesthetic dispute needs photographic evidence. Further information can also make clear that the author of the photographs should be the condominium administrator or a person appointed by

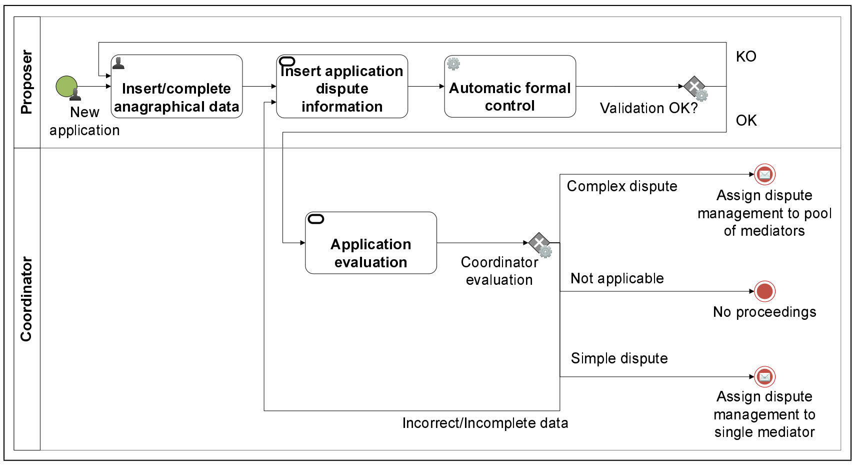

Proposer
Coordinator
New application
Insert/complete anagraphical data
Insert application dispute information
Automatic formal control
Validation OK?
KO
OK
Application evaluation
Coordinator evaluation
Complex dispute
Assign dispute management to pool of mediators
Not applicable
No proceedings
Simple dispute
Assign dispute management to single mediator
Incorrect/Incomplete data

Fig. 1. BPMN of the subprocess

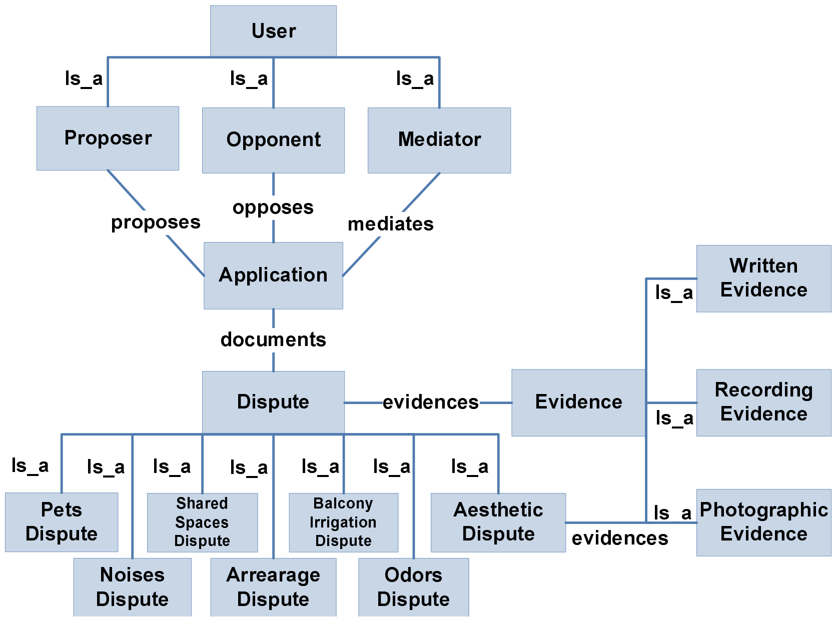


Fig. 2. Ontology describing the data used in the subprocess

the administrator. It is clear that with such support we make a step towards the basic aim of mediation, to ensure a facilitated access to justice.

3.2 SyBeL modeling

By integrating the information given by the ontology with the BPMN process description, we build SyBeL domain describing the data model needed to support the process.

```
<entityType name="DisputeType" specializations="AesteticDisputeType
     NoiseDisputeType PetDisputeType" modelReference="cc:dispute">
 <properties>
  <property name="request"><baseType>string</baseType></property>
  <property name="evidences" modelReference="cc:dispute/cc:evidences">
   <collectionType>
    <entityType>EvidenceType</entityType>
   </collectionType>
  </property>
  <property name="evidenceValidation" modelReference="cc:dispute/
       cc:evidencevalidation"><baseType>any</baseType></property>
</properties></entityType>

<entityType name="EvidenceType" specializations="
     PhotographicEvidenceType" modelReference="cc:evidence">
 <properties>
  <property name="author" modelReference="cc:evidence/cc:author">
   <baseType>string</baseType></property>
  <property name="date" modelReference="cc:evidence/cc:date">
   <baseType>string</baseType></property>
</properties></entityType>

<entityType name="AesteticDisputeType" base="DisputeType" modelReference
     ="cc:aestheticdispute">
 <properties>
  <property name="evidences" modelReference="cc:dispute/cc:evidences">
   <collectionType>
    <entityType>PhotographicEvidenceType</entityType>
   </collectionType></property>
</properties></entityType>

<entityType name="PhotographicEvidenceType" base="EvidenceType"
     modelReference="cc:photographicevidence">
 <properties>
  <property name="photo" modelReference="cc:photographicevidence/
       cc:photo"><baseType>binary</baseType></property>
</properties></entityType>
```

Fig. 3. A fragment of the SyBeL data model

An excerpt of the domain XML definition is shown in Figure 3, where the *DisputeType*, *EvidenceType*, *AesteticDisputeType* and *PhotographicEvidenceType* entities are declared through the entityType construct. Note that the hierarchy defined at ontological level by the "is_a" relations is transformed in a set of suitable entity derivations using the specializations and base attributes. Other semantic relations are transformed to properties referencing the corresponding entities: as an example, the *evidences* relation between the *dispute* and the *evidence* becomes the *evidences* property of the *DisputeType* entity, which is defined as a collection of *EvidenceType* entities. Finally, all the relevant elements are annotated with the modelReference attribute which connects the SyBeL constructs with the corresponding concepts of the semantic model.

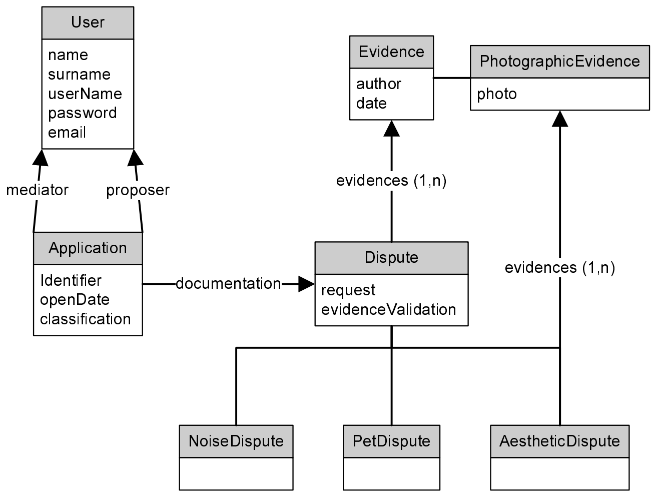


Fig. 4. Graphical view of the SyBeL data model

A graphical representation of the SyBeL model is shown in Figure 4, which can be easily compared with the ontology shown in Figure 2.

Moreover, thanks to the tools provided by SyBeL, it is possible, e.g., to automatically generate useful artifacts from the domain description, which can be directly used to drive the data model implementation. As an example, Figure 5 shows the Java declarations generated from the domain fragment of Figure 3.

Now we are ready to generate a SyBeL behaviour from the BPMN process, using the domain entities to type the involved objects. Figure 6 shows a fragment of this file, in particular some parts of the main execution flow. Here, the BPMN steps are transformed in corresponding actions executed on the domain entities and encapsulated in interaction or happening events. Actions may have parameters (such as *details*) and return values (such as *applicationValidation*), which are also typed using the domain definitions. BPMN process branches are encoded in alternative flows (such as *applicationNotValid*), which contain the corresponding guard condition. Note that each event can be also textually described, to better link it to the corresponding BPMN step.

Also in this case, the SyBeL tools provide an automatic way of generating useful derived artifacts from the behaviour description. As an example, Figure 7 shows the HTML documentation of the process, which can be easily read and interpreted both by domain experts, which may use it to further validate the process, and programmers, which may take advantage from this typed, procedural view to drive the process implementation.

```
// Semantic reference: cc:dispute
// Subclasses: AesteticDisputeType NoiseDisputeType PetDisputeType
class DisputeType {
 public String request;
 // Semantic reference: cc:dispute/cc:evidences
 public Collection<EvidenceType> evidences;
 // Semantic reference: cc:dispute/cc:evidencevalidation
 public Object evidenceValidation;
};
// Semantic reference: cc:evidence
// Subclasses: PhotographicEvidenceType
class EvidenceType {
  // Semantic reference: cc:evidence/cc:author
 public String author;
 // Semantic reference: cc:evidence/cc:date
 public String date;
};
// Semantic reference: cc:aestheticdispute
// Base class: DisputeType
class AesteticDisputeType extends DisputeType {
  // Semantic reference: cc:dispute/cc:evidences
 public Collection<PhotographicEvidenceType> evidences;
};
// Semantic reference: cc:photographicevidence
// Base class: EvidenceType
class PhotographicEvidenceType extends EvidenceType {
  // Semantic reference: cc:photographicevidence/cc:photo
 public byte[] photo;
};
```

Fig. 5. Java code generated from the SyBeL data model

4 Conclusions

In this paper we have presented our approach to integrate the description capabilities of the BPMN with semantic information based on ontologies.

The consideration we started with is that, in most cases, it is necessary to add to the process description some semantic information useful to define the domain in which the process will operate, and that we can assume such semantic information always codified by ontologies, preexisting or suitably generated by tools such as Semantic Turkey.

Instead of directly integrating the BPMN model with semantic information, which could overcharge the model making it more difficult to read by business executives, we use the features of the lightweight SyBeL modeling language to merge the process information provided by the BPMN with the information derived from domain ontologies in an integrated view, which preserves the original specifications. Then, thanks to the tools provided in SyBeL, it is possible to produce a variety of artifacts and documents that can facilitate the subsequent implementation steps of the executable process.

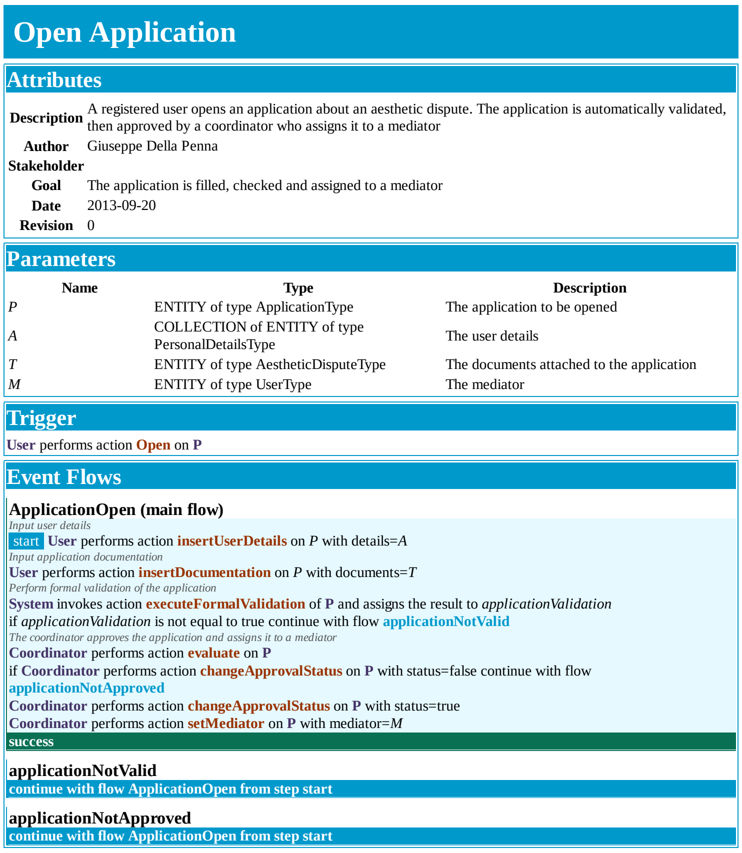
Acknowledgments. Authors would like to thank Andrea Palazzo for his contribution to the analysis of the case study.

```
<flow main="true" name="ApplicationOpen">
 <interaction stepName="start">
  <description>Input user details</description>
  <actor>User</actor>
  <actionExecute action="insertUserDetails">
   <variable>P</variable>
   <argument parameter="details">
    <variable>A</variable>
   </argument>
 </actionExecute></interaction>
 ...
 <happening resultVariable="applicationValidation">
  <description>Perform formal validation of the application</description
      >
  <actor>System</actor>
  <actionExecute action="executeFormalValidation">
   <entity>P</entity>
  </actionExecute>
 </happening>
 <alternative flow="applicationNotValid">
  <condition><comparison operator="ne">
    <variable>applicationValidation</variable>
    <baseValue>true</baseValue>
 </comparison></condition></alternative>
 ...
 <interaction>
  <actor>Coordinator</actor>
  <actionExecute action="setMediator">
   <entity>P</entity>
   <argument parameter="mediator">
    <variable>M</variable>
   </argument>
 </actionExecute></interaction>
 <success/>
</flow>
```

Fig. 6. A fragment of the SyBeL behaviour

References

1. Born, M., Drr, F., Weber, I.: User-friendly semantic annotation in business process modeling. In: Springer (ed.) Web Information Systems Engineering WISE 2007 Workshops. Lecture Notes in Computer Science, vol. 4832, pp. 260–271 (2007)
2. Business Process Trends: Case studies and success stories, http://www.bptrends.com/ resources_publications.cfm? publicationtypeID= DFFB84D1-1031-D522-339E8B42679D513F
3. Della Penna, G., Intrigila, B., Magazzeni, D., Orefice, S., Del Sordo, R., Cardinale Ciccotti, G.: SyBeL: a system modelling language enhancing automatic support in the software development process. International Journal of Software Engineering and Knowledge Engineering 23(2), 223–257 (3 2013)
4. Gazzetta Ufficiale della Repubblica Italiana: Legislative decree 4/3/2010, n. 28: mediation in civil and commercial matters, http://www.gazzettaufficiale.it/eli/id/2010/03/05/010G0050/sg
5. Government of Nova Scotia: Condominium disputes applying for dispute resolution, http://www.gov.ns.ca/snsmr/access/individuals/consumer-awareness/condominiums/applying-for-dispute-resolution.asp
6. Istituto Nazionale per la Mediazione e L'Arbitrato: Il mediatore condominiale, http://www.inmediar.it/il-mediatore-condominiale/



Open Application

Attributes

Description	A registered user opens an application about an aesthetic dispute. The application is automatically validated, then approved by a coordinator who assigns it to a mediator
Author	Giuseppe Della Penna
Stakeholder	
Goal	The application is filled, checked and assigned to a mediator
Date	2013-09-20
Revision	0

Parameters

Name	Type	Description
P	ENTITY of type ApplicationType	The application to be opened
A	COLLECTION of ENTITY of type PersonalDetailsType	The user details
T	ENTITY of type AestheticDisputeType	The documents attached to the application
M	ENTITY of type UserType	The mediator

Trigger

User performs action **Open** on **P**

Event Flows

ApplicationOpen (main flow)

Input user details
start **User** performs action **insertUserDetails** on *P* with details=*A*
Input application documentation
User performs action **insertDocumentation** on *P* with documents=*T*
Perform formal validation of the application
System invokes action **executeFormalValidation** of **P** and assigns the result to *applicationValidation*
if *applicationValidation* is not equal to true continue with flow **applicationNotValid**
The coordinator approves the application and assigns it to a mediator
Coordinator performs action **evaluate** on **P**
if **Coordinator** performs action **changeApprovalStatus** on **P** with status=false continue with flow **applicationNotApproved**
Coordinator performs action **changeApprovalStatus** on **P** with status=true
Coordinator performs action **setMediator** on **P** with mediator=*M*
success

applicationNotValid
continue with flow ApplicationOpen from step start

applicationNotApproved
continue with flow ApplicationOpen from step start

Fig. 7. HTML documentation generated from the SyBeL behaviour

7. Official Journal of the European Union: European directove 2008/52/ec: on certain aspects of mediation in civil and commercial matters, http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=OJ:L:2008:136:0003:0008:En:PDF
8. OMG: Business process model and notation (bpmn), http://www.bpmn.org/
9. Pazienza, M.T., Scarpato, N., Stellato, A., Turbati, A.: Semantic turkey: A browser-integrated environment for knowledge acquisition and management. Semantic Web Journal 3(2) (2012)
10. Seitz, C.: Patterns for semantic business process modeling. Tech. Rep. 2008-07, Institut fr Informatik, University of Augsburg (2008)
11. W3C: Ontology driven architectures and potential uses of the semantic web in systems and software engineering, http://www.w3.org/2001/sw/BestPractices/SE/ODA/