Fernando Silva Parreiras, Jeff Z. Pan, Uwe Assmann, Jakob Henriksson (Eds.)

TWOMDE 2008

First International Workshop on Transforming and Weaving Ontologies in Model Driven Engineering

Preface

The interest in integrating Ontologies and Software Engineering has gained more attention with commercial and scientific initiatives. The Semantic Web Best Practice and Deployment Working Group (SWBPD) in W3C included a Software Engineering Task Force (SETF) to explore how Semantic Web and Software Engineering can cooperate. The Object Management Group (OMG) has an Ontology Platform Special Interest Group (PSIG) aiming at formalizing semantics in software by knowledge representation and related technologies.

In counterpart, as Model Driven Engineering spreads, disciplines like model transformation and domain specific modeling become essential in order to support different kinds of models in an model driven environment. Understanding the role of ontology technologies like knowledge representation, automated reasoning, dynamic classification and consistence checking in these fields is crucial to leverage the development of such disciplines.

The TWOMDE 2008, affiliated with the 11th International Conference on Model Driven Engineering Languages and Systems (MoDELS2008), brought together researchers and practitioners from the modeling community with experience or interest in MDE and in Knowledge Representation to discuss about: (1) how the scientific and technical results around ontologies, ontology languages and their corresponding reasoning technologies can be used fruitfully in MDE; (2) the role of ontologies in supporting model transformation; (3) and how ontologies can improve designing domain specific languages.

The objective of the workshop was: (1) to present success cases of integrated approaches; (2) to present state-of-the-art researches covering ontologies in MDE; (3) and to encourage the modeling community to explore different aspects of ontologies.

Toulouse,
September 2008

Fernando Silva Parreiras, Jeff Z. Pan
Uwe Assmann, Jakob Henriksson

Workshop Organization

Programme Chairs

Fernando Silva Parreiras
Uwe Assmann
Jeff Z. Pan
Jakob Henriksson

Programme Committee

Colin Atkinson
Kenneth Baclawski
Saartje Brockmans
Luís Ferreira Pires
Dragan Gašević
Giancarlo Guizzardi
Peter Haase
Gerti Kappel
Elisa Kendall
Holger Knublauch
Harald Kühn
Daniel Oberle
Alexander Paar
Bernhard Rumpe
Steffen Staab
Phil Tetlow
Gerd Wagner
Andreas Winter

Local Organization

External Reviewers

Dirk Reiss
Marvin Schulze-Quester
Srdjan Zivkovic

Table of Contents

Potential applications of ontologies and reasoning for modeling and software engineering

Andreas Friesen, SAP Research, CEC Karlsruhe

Vincenz-Prießnitz-Str.1, D-76131 Karlsruhe
andreas.friesen@sap.com

Abstract. In the last few years SAP introduced Service-oriented Architecture as a blueprint for an adaptable, flexible, and open IT architecture for developing services-based, enterprise-scale business solutions. An Enterprise Service is typically a series of Web services combined with business logic that can be accessed and used repeatedly to support a particular business process. Aggregating Web services into business-level enterprise services provides a more meaningful foundation for the task of automating enterprise-scale business scenarios. At the same time, SAP Research was investigating in numerous research projects how ontologies, reasoning, semantic web services technologies, and advanced business process modelling technologies can be applied in order to improve technical foundation behind SAP SOA and business modelling. In this extended abstract we describe some selected business process composition and integration scenarios identified as potential candidates for application of business process composition techniques, semantic technologies, ontologies, and reasoning. We further identify some challenges linked to application of such advanced technologies in the context of modelling and software engineering.

Introduction

The advent of Service-oriented Architecture (SOA) and Web Services (WS) opened new possibilities for smooth Enterprise Application Integration (EAI), i.e., enabling of cross-system message flow automation, in A2A and B2B scenarios in a loosely-coupled manner. In principle, Web Services enabled enterprise applications can be used by anyone, from anywhere, at any time, and on any type of platform. Additionally, SOA opens also possibilities to innovate on top of already available *best practices business processes* implemented in existing enterprise applications. This can be achieved through adoption, extension, and composition of business processes across enterprise applications. Such composites can again be made available for composition or integration scenarios through SOA-enabling.

With respect to enterprise applications, there is obviously a relationship between Business Process Management (BPM) and SOA. BPM as a management discipline helps business organizations to standardize and continuously optimize the operational processes throughout the complete business processes lifecycle. BPM as a technology provides organizations with a framework of tools to compose, model, deploy, execute, and monitor processes that include human and system tasks or that span across different business applications and require a broad set of integration capabilities. From a technical point of view, a business process is a "collection of interrelated tasks, which accomplish a particular goal".[1] Hence, SOA is an enabler of BPM and at the same time, BPM provides value on top of a service-enabled enterprise application.

The lack of formally represented semantic meaning in the WS technology stack causes the tasks of discovering, selecting, composing, and binding Web Services being considered as manual steps performed by a human. The recent advent of the Semantic Web and Semantic Web Services (SWS) promises new standardized means to formally capture the representation of the semantic meaning of data and interfaces. This enables the machines to automatically reason and to draw conclusions about the "intended meaning". The so-called Semantic Web Services promise a higher degree on automation concerning discovery, invocation, composition, and monitoring of Web Services.

Similarly, the lack of formally represented meaning in the BPM technology stack causes the tasks of defining, classifying, discovering, selecting, composing, integrating and refining (adopting, extending) business processes as well as binding them to a SOA-enabled IT platform a challenging task requiring a broad set of human skills and expertise.

[1] http://en.wikipedia.org/wiki/Business_process

Modeling at SAP

SAP introduced Service-oriented Architecture as a blueprint for an adaptable, flexible, and open IT architecture for developing services-based, enterprise-scale business solutions. In order to achieve this, service interfaces need to be clearly defined and stable, make use of interoperable (global) data types, and must follow clearly defined communication and behavioral patterns. Furthermore, the underlying business model (interaction between business objects) needs to be transparent to establish common ground in terms of relationships between business objects which play a role when calling services.

Services that fulfill the above criteria are called Enterprise Services and are published in an Enterprise Services Repository. Enterprise Service is typically a series of Web services combined with business logic that can be accessed and used repeatedly to support a particular business process. Enterprise services that SAP delivers are grouped in Enterprise Service Bundles (ES Bundles) based on specific business scenarios. This provides a meaningful foundation for the task of automating enterprise-scale business scenarios. The use of Enterprise Services or ES Bundles for integration/automation purposes can be seen as "service consumption". For instance, the usage of Web Services residing on top of Private Business Process Layer in order to create a composite application or usage of Web Services residing on top of Public Business Process Layer in order to create collaborative business process means service consumption (Figure1). Furthermore, it is possible to implement business applications (composites) on top of Enterprise Services and other third party services. Composites benefit from the reuse of existing assets and process flexibility. This means that it is possible to replace, remove, or rearrange steps in a process but also to implement new collaborative scenarios. The creation and packaging and publication of a composite application as ES bundle can be seen as "service provisioning" (Figure1).

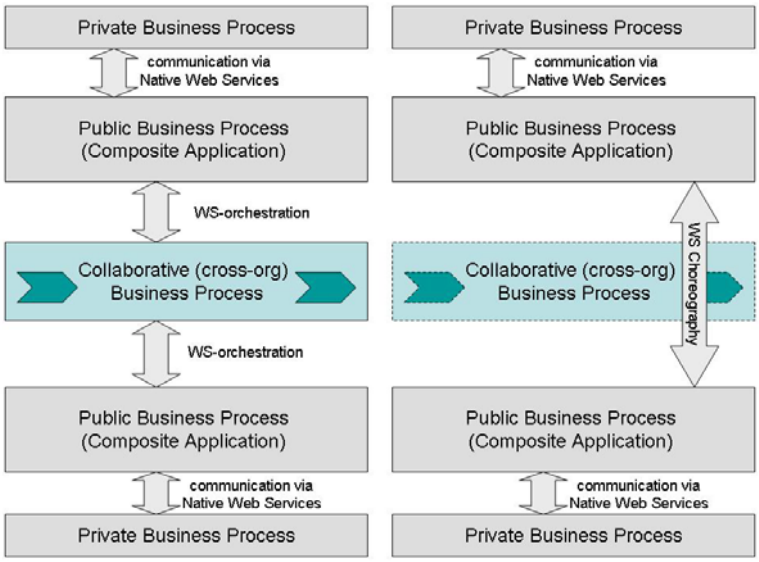


Figure 1 SOA-enabled EAI

BPM provides value on top of SOA by providing business value and business semantics to the Enterprise Services. BPM provides business process modeling capabilities identifies and classifies business scenarios, business processes and their variants and links them to integration scenarios. Thus, BPM helps to determine the granularity and required behavior of Enterprise Services and to give them business semantics. SOA is an enabler for BPM as it provides support for creating different types ob business processes, e.g., human-centric/system-centric composite business processes/applications as well as integration processes, and defines their accessibility and extensibility points.

In overall, SOA at SAP can already be seen as a convergence point between business and IT. The lack of formally represented semantic meaning in both BPM and SOA technology stacks causes all tasks to be performed within this infrastructure to be manual steps performed by a human.

Potential applications for ontologies and reasoning

In the last few years, SAP Research was investigating in numerous research projects how ontologies, reasoning, semantic web services technologies, and advanced business process modelling technologies can be applied in order to improve technological foundations behind SOA and BPM.

In order to give an idea about the scope covered within these projects, we give a short overview of the most relevant projects before we introduce two examples for potential application of ontologies, reasoning and business process composition technologies:

ATHENA (Advanced Technologies for Interoperability of Heterogeneous Enterprise Networks and Their Application) – main scope on Enterprise Interoperability, e.g., the concept Private/Public/Collaborative Processes as shown in Figure1 built one of the core concepts

DIP (Data, Information and Process Integration with Semantic Web Services) – main scope on Semantic Web Services Lifecycle (WSMO, WSML, WSMX)

SUPER (Semantics Utilized for Process Management within and between Enterprises) – main scope on Semantic Business Process Management, i.e, bridge the gap between Business Experts Perspective and IT Implementation Perspective (sBPMN, sBPEL, WSMO)

FUSION (Business Process Fusion using Semantically-enabled Service-oriented Business Applications) – main scope on Semantically-enabled Process-oriented Enterprise Application Integration (Private-Public Abstraction Layer for Data/Functionality/Behavior, EAI Ontology, Semi-automated Process Integration, SA-WSDL, BPEL)

MOST – Marrying Ontologies with Software Technologies (Semantically-enabled MDA - Ontology-driven Software Engineering, Guidance and Traceability in the Software Development Process and in Solution Domain (MOF, UML, BPMN, OWL, DSLs)

Example 1: Shipper Carrier Integration for delivery of goods

Let's assume shippers and carriers are SOA-enabled (in the sense of ES Bundles for the delivery step of let's say Order2Cash process), i.e. technically enterprise application integration is not a problem. Additionally, a shipper would like to automate the discovery, integration and selection of carrier services according to business rules that specify his preferences from the business perspective. This is obviously not possible without shared and common understanding of the business scope within the logistics domain of carriers and shippers and standardized means to describe the capabilities of the carriers and requirements of the shippers. We solved this problem by introducing a logistics ontology specified in OWL-DL and using SA-WSDL annotations to link WSDL to the concepts in the ontology. The complete solution is described in [1].

Example 2: Semi-automated process integration of enterprise and composite applications

Let's assume we have a SOA-enabled HR System providing an ES Bundle for a recruiting process. A department in a company would like to extend this process in order to introduce department specific recruiting procedures that a not supported by best practice process in the HR System. This can be done by developing a composite application on top of the ES Bundle provided by the HR System. However, all steps have to be done manually. We could semi-automate creation of a composite application with respect to the data and control flow tasks in the composite application by semantically annotating the input and output messages of the Enterprise Services and their behavior. The only steps that still need to be done manually are: the modeling of the new behavioral interface for the applicant and the internal workflow of the composite application. The integration of the data and control flow between the applicant interface, the HR System and the internal business process of the composite application is a guided semi-automatic process. The complete background used in this solution is described in [2], [3].

Challenges and future work

The above as well as many, many other examples show that application of ontologies, reasoning and advanced business process composition techniques in the area of integration and composition of enterprise applications is reasonable and feasible. However, there are only bits and pieces based on different conceptual assumptions and technologies. The challenge is to bring all loosely-coupled models and ontologies under an umbrella of a unified conceptual framework that combines the classical modeling techniques with ontological modeling in one hybrid meta-model. The framework must

address the business level as well as IT level and provide a link from business to IT that ideally supports round-trip engineering supported by guidance and traceability methodologies for both the software engineering process and the solution domain. SAP Research started to investigate these issues within the EU-funded project MOST (http://www.most-project.eu/).

Reference

[1] Andreas Friesen, Kioumars Namiri, Towards Semantic Service Selection for B2B Integration, Proceedings of The 6th International Conference on Web Engineering (ICWE 2006), Menlo Park, CA, July 2006

[2] Jens Lemcke, Andreas Friesen, Composing Web-Service-like Abstract State Machines (ASMs), Proceedings of the 2007 IEEE Congress on Services (ICWS/SCC 2007), July 2007, Salt Lake City, USA

[3] Jens Lemcke, Andreas Friesen, Considering Realistic Web Service Features for Semi-automatic Composition, In the Proceedings of the 3rd South-East European Workshop on Formal Methods (SEEFM 2007), November 2007, Thessaloniki, Greece, ISBN 978-960-89629-4-1

Using an Ontology to Suggest Software Design Patterns Integration

Dania Harb, Cédric Bouhours, Hervé Leblanc

IRIT – MACAO
Université Paul Sabatier
118 Route de Narbonne
F-31062 TOULOUSE CEDEX 9
{harb, bouhours, leblanc}@irit.fr

Abstract. To give a consistent and more valuable feature on models, we propose that model-driven processes should be able to reuse the expert knowledge generally expressed in terms of patterns. In order to formalize and use them, some design pattern ontologies have been developed. To share them on the Web they have been implemented using the OWL language. They can be easily interrogated with dedicated query languages. Our work has consisted in extending a design pattern intent ontology with "alternative model" and "strong points" concepts, which partially refers "anti-patterns". We validate this approach in tooling a step of a design review activity, we have proposed. This activity, directed by design patterns, is adapted to a model driven process, for the need to improve object-oriented architecture quality.

Keywords: OWL, SPARQL, Software Design Pattern, Design Review Activity, MDE, MDA

1 Introduction

The emergent MDE community, aiming at giving a productive feature on models, has proposed model-driven process development. However, to obtain guarantees on model relevance at the end of each activity, these processes should promote the reuse of the knowledge of experts generally expressed in terms of analysis [1], design [2] or architectural [3] patterns approved by the community. Given the existence of "code review" activities [4] in some development processes, we have specified a "design review" activity [5] directed by design patterns and oriented to model quality. In this activity, we propose to parse models to find fragments substitutable with software design patterns and to replace them if the intent of the designer matches with the intent of the pattern and if the architectural qualities of the pattern are needed. Our activity is situated after the design stage, and its purpose is to urge and to help the designer to integrate design pattern in his design.

Thanks to their *Design Pattern Intent Ontology* (DPIO), Kampffmeyer et al. [6] have developed a wizard enabling designers to efficiently retrieve software design patterns applicable for their design problems, during the design stage. Our approach

has not the same timing. It is situated after the design stage, and it verifies if there is no known bad design practices in a model. So, the designer is not in need of identifying design problems, it is the activity which finds the lacks in his models and suggests design patterns integrations instead. However, the DPIO [6] is an interesting start point for the formalization of our concepts because it links software design pattern to some problem concepts. So, in reusing this ontology backwards (from the pattern to the design problems), and in adding our concepts, we are able to establish a dialog with the designer.

In this paper, after presenting the design review activity, we explain how we have reused and extended the DPIO [6]. We illustrate the execution of our activity on a "file system management" example.

2 The Design Review Activity

The design review activity, presented in [5], may be decomposed into four steps (see Fig. 1).

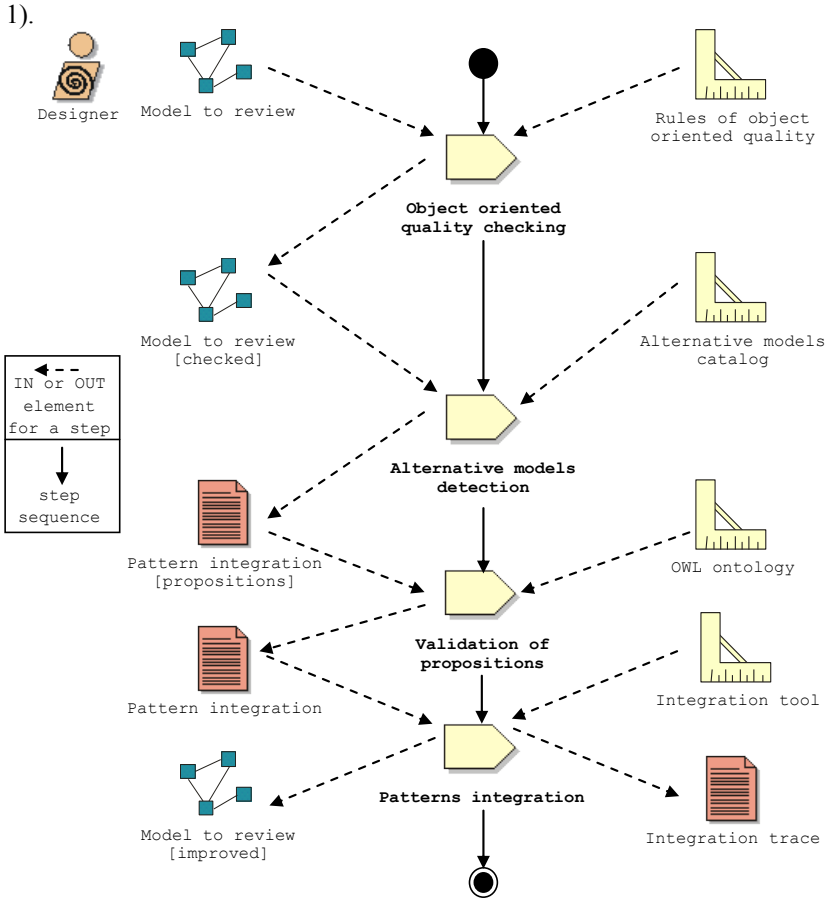


Fig. 1. Design Review Activity

In order to work with models in a "sufficient" quality, the first step checks good basic object-oriented design practices.

When the model to review is checked in a "sufficient" quality state, the second step consists in an automatic research of model fragments which are candidate to a substitution with a pattern. This research is based on structural similarities detection with "alternative models". An "alternative model" is a model which solves inadequately the same problem as a pattern [5]. That means there is a better solution to solve this problem. Our work hypothesis is that a software design pattern is the best solution for a given design problem. According to the taxonomy proposed by Chikofsky and Cross [8], our detection technique can be connected to a redocumentation technique as to permit model restructuring. Our "alternative models" catalog is presented in [9], with the experiments used to constitute it.

Each "alternative model" detected in the model represents propositions of fragments substitutable with a design pattern. Since we need the designer opinion in the third step, our ontology will help him determine if his intent matches with the suggested pattern intent and whether the propositions are needed in the model to review.

With the designer authorization, the last step consists in integrating the validated propositions into the model. This integration is done thanks to an automatic model refactoring.

3 Reusing and extending an existing ontology

In order to improve the design of object oriented models, our work relies on detecting all instances of alternative models in UML design models and substituting them, if necessary, with appropriate design patterns. Each class of the instances detected is connected in the same manner as the classes of the alternative model. So, since the detection is only structural, the instances detected must be checked by the designer before any substitution with a pattern. Therefore, after the detection step, propositions of patterns integration consist of sets of model fragments representing a possible substitution. These sets may be large where some fragments may not be relevant with a substitution. So, to help the designer in filtering the fragments, we need an ontology that formalizes intent of design patterns (is the substitution have a sense?) and our characterizations of "alternative models" in terms of quality features (is the effort of the substitution balanced by improved architectural qualities?).

For this purpose, we choose OWL, the Web Ontology Language [10], to import an existing ontology on design patterns intent and extend it by adding our knowledge on "alternative models". We validated our new knowledge base using a specific query language to interrogate it and filter out the pertinent information.

3.1 Requirements

Our catalogue is composed with “alternative models”, introduced in Section 2, and their characterization. We have constituted our catalog in asking students to solve some design problems. These problems were simply solvable with designs patterns, but, as the students chosen have no knowledge on design patterns, they solve the problems without using design patterns. In following our work hypothesis, their solutions were not the best solution for the problem, and so, the models produced had some design defects. The characterization of these defects consists in a valuation of the “strong points” of the pattern. “Strong points” are criteria of object-oriented architecture or software engineering quality, partially deduced from the “consequences” section of the GoF [2] catalogue and from our study on the design defects of “alternative models”. As pattern injection may alter some object-oriented metrics [11], “strong points” allow us to compute dedicated pattern metrics to classify the “alternative models” and to help the estimation of the pertinence of pattern injection in a design model. Each “alternative model” perturbs the “strong points” of its associated pattern.

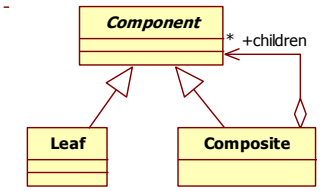
Since we need to formally describe design patterns, “alternative models” and “strong points” in a machine readable format, we start with the DPIO ontology. These concepts must be constructed in a way that allows querying based on the “alternative model” detected.

Intent: Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

Applicability: Use the Composite pattern when:

- you want to represent part-whole hierarchies of objects.
- you want clients to be able to ignore the difference between compositions of objects and individual objects. Clients will treat all objects in the composite structure uniformly.

Structure:



Strong points:

1 Decoupling and extensibility
  1.1 Maximal factorization of the composition
  1.2 Addition or removal of a Leaf does not need code modification
  1.3 Addition or removal of a Composite does not need code modification
2 Uniform processing
  2.1 Uniform processing on operations of composed object
  2.2 Uniform processing on composition managing
  2.3 Unique access point for the client

Fig. 2. Composite Pattern and its "Strong Points"

In Fig. 2, we present one of the GoF patterns, named *Composite*. The intent, the applicability and the structure are provided directly from the GoF book while the "strong points" are deduced from our experiments by comparing solutions to specific design problem implemented by the *Composite* pattern and its "alternative models". Fig. 3 shows the structure and the design defect valuation of an "alternative model" to the *Composite* pattern. We have named it "Development of the composition on Composite with process conformance" in reference of its design defects. Then an "alternative model" can be considered as a "chipped pattern".

So we have made two major hypotheses about "alternative models". First, each "alternative model" is attached by the valuation of their design defects to a unique design pattern. Second, each "alternative model" has one or more strong points perturbed. We assume that the same structure of an "alternative model" can be duplicated in our catalog, but with a different name, a different valuation and some different components.

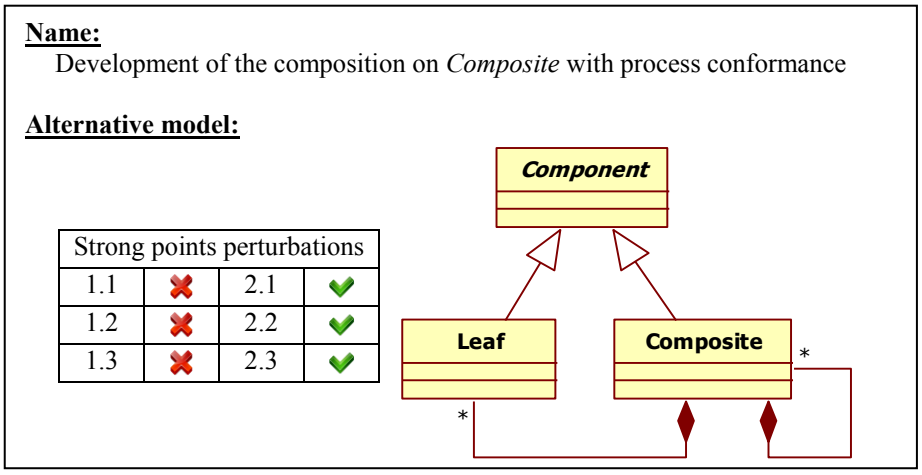


Fig. 3. Characterization of an "Alternative Model"

3.2 Existing ontology: the Design Pattern Intent Ontology

Design patterns have been used successfully in recent years in the software engineering community in order to share knowledge about the structural and behavioural properties of software. Most of the existing approaches to formalizing design patterns are based on structural aspects. For example, the work of Dietrich et al. [12] uses the OWL to formally describe the structure of design patterns and then transform it in first-order logic predicates which are reuse as an entry for a scanner pattern. However, there is more lightly approaches concentrated in the usability of design patterns according to the design problems they solve. Kampffmeyer and Zschaler [6] define the intent of the 23 GoF design patterns [2] using OWL. Their

work was based on the work of Tichy [13], who developed a catalogue of more than hundred design patterns classified according to the problems patterns solve.

The core structure of the DPIO, provided from the paper [6], is presented in Fig. 4 by UML classes and associations. Kampffmeyer and Zschaler chose to represent their ontology with UML diagram because they consider that is easily to understand. To read the diagram, they indicate: "The relations between *DesignPattern*, *DPProblem* and *ProblemConcept* classes are depicted using UML-notations. UML classes symbolize OWL classes and UML associations symbolize OWL object properties. Each *DesignPattern* is a solution to one or more design pattern problem *DPProblem*. The association between them indicates an object property *isSolutionTo* which is an inverse property of *isSolvedBy*. *DPProblem* is defined that is the root node for more specific problems. The association class *Constrains* indicates an OWL object property that can be specialized also by subproperties. *DPProblem* is a set of classes that describe a problem by constraining a *ProblemConcept"*. The DPIO contains the vocabulary for describing the intent of design patterns.

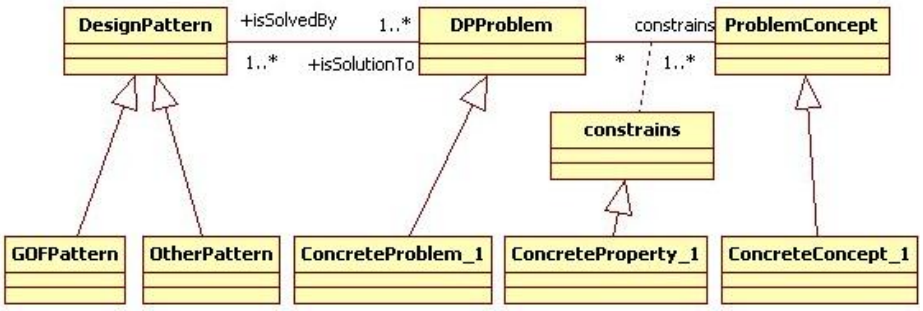


Fig. 4. Graphical overview of the core structure of the DPIO

All the 23 GoF patterns inherit from the DesignPattern class. DPProblem and ProblemConcept are the root classes of the other hierarchies.

Based on the work of [6], and instead of finding the design pattern for a given problem, we retrieve the intent of a design pattern. It is much like reversing the query to get the pertinent data from the same ontology. So we can benefit from their existing work and their published ontology.

3.3 Method and Results

Now to determine the scope of our new ontology, there are kinds of questions called "competency questions" the ontology should be able to answer [14]. Our work could be defined in 3 steps: first, when an "alternative model" is detected, we need to interrogate our knowledge base to know which design pattern could replace it. Second, we will verify with the designer if his "alternative model" detected has a similar intent as the corresponding design pattern. Last, in this case, we will show him

the lack in his model by displaying the perturbed "strong points". Then, if the designer finds the need to improve his model, his fragment will be substituted with the design pattern. Therefore, the three competency questions are as follow:

1. Which design pattern could replace a given "alternative model"?
2. What is the intent of the corresponding design pattern?
3. Which are the "strong points" perturbed using this "alternative model"?

In designing the structure of the new ontology, we took into consideration all the possible relations between the classes in the DPIO model and the classes we want to add:

1. Each"*alternative model"* could be replaced by one and only one *Design Pattern*. But a *Design Pattern* will replace one to many "alternative models".
2. An "*alternative model*" *perturbs* at least one "s*trong point"* of the *Design Pattern* that can replace it.

From this analysis, we extend the DPIO by adding our new concepts.

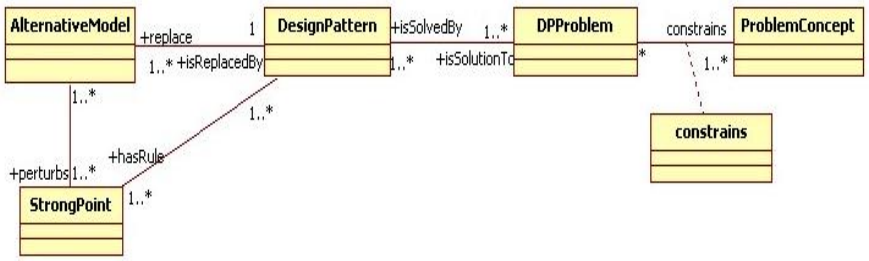


Fig. 5. Graphical overview of the structure of the extended ontology

Fig. 5 represents the new structure of the extended ontology. Based on this structure and the relations between classes, we extended the existing ontology with OWL classes and properties as follow:

1. Two new OWL classes:
    a. *AlternativeModel*: the root class of all "alternative models". They are grouped by the design pattern that could replace them. For example, we find six "alternative models" for the *Composite* pattern. They inherit all from the *Composite_AM* class (Fig. 6). They have the name of their super class followed by their numeration in the catalogue.
    b. *StrongPoint*: the root class of all the "strong points". They are attached to a design pattern. For example, we find two main "strong points" for the *Composite* pattern: *Composite_Rule_1* and *Composite_Rule_2* (Fig. 6); each one of them was précised by three sub features. They have the name of their super class followed by their numeration in the catalogue.

2. Four new OWL properties:
    a. *isReplacedBy*: links an *AlternativeModel* to his corresponding *DesignPattern*.
    b. *Replace*: the inverse of *isReplacedBy.*
    c. *Perturbes*: links an *AlternativeModel* to the valuation of the corresponding pattern "strong points" (*StrongPoint*).
    d. *hasRule*: links a *DesignPattern class* to one of its *StrongPoint.*

Fig. 6 shows a detailed structure of the extended base concerning the *Composite* pattern. The "alternative model" presented in Fig. 3 perturbs the three subfeatures of the first "strong point" of the *Composite* pattern that concerned in the *Decoupling and extensibility*. More precisely, for each OWL class concerning our concepts, we have:

OWL Classes	***rdfs:comment***
Composite_AM_5	Development of the composition on "Composite" with protocol conformance
Composite_Rule_1	Decoupling and Extensibility
Composite_Rule_2	Uniform processing
Composite_Rule_1.1	Maximal factorization of the composition
Composite_Rule_1.2	Adding or removing a Leaf does not need a code modification
Composite_Rule_1.3	Adding or removing a Composite does not need a code modification
Composite_Rule_2.1	Uniform processing on operations of composed objects
Composite_Rule_2.2	Uniform processing on compositions management
Composite_Rule_2.3	Unique access point for the client

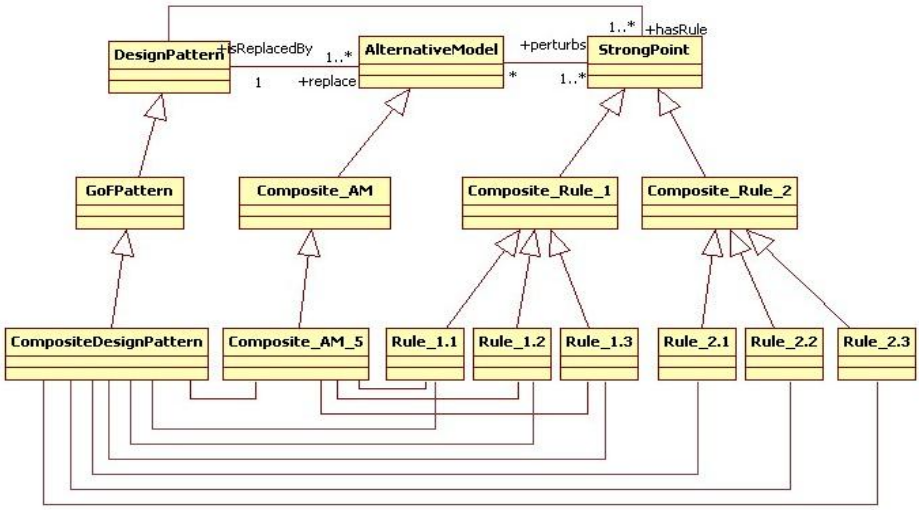


Fig. 6. Detailed structure of the extended ontology

For presentation reasons, we have omitted the name of the design pattern in each sub feature.

We used Protégé [15], an open source ontology editor and knowledge-base framework, to load the existing ontology and add our new classes, properties, property characteristics, and interrogate it using queries. We referred to a user guide [14] on how to develop an ontology using Protégé and the OWL Plug-in. We created our OWL classes, linked them by OWL properties, and interrogated the knowledge base by generating SPARQL (SPARQL Protocol and RDF Query Language) [16] queries to answer our competency questions.

SPARQL is a W3C Candidate Recommendation towards a standard query language for the Semantic Web. Its focus is on querying RDF graphs at the triple level. *SPARQL* can be used to query an RDF Schema or OWL model to filter out individuals with specific characteristics.

4 Illustration on a "File System Management" Design

After adding our new concepts to the DPIO, the knowledge base could now be interrogated according to the competency questions we mentioned earlier. Standard ontology reasoning is used to retrieve the results responding to queries. In order to illustrate the use of the ontology, we execute the whole activity on an example. It was found in a subject of an object-oriented programming supervised practical work. It aims to implement a file management system represented in the Fig. 7 below.

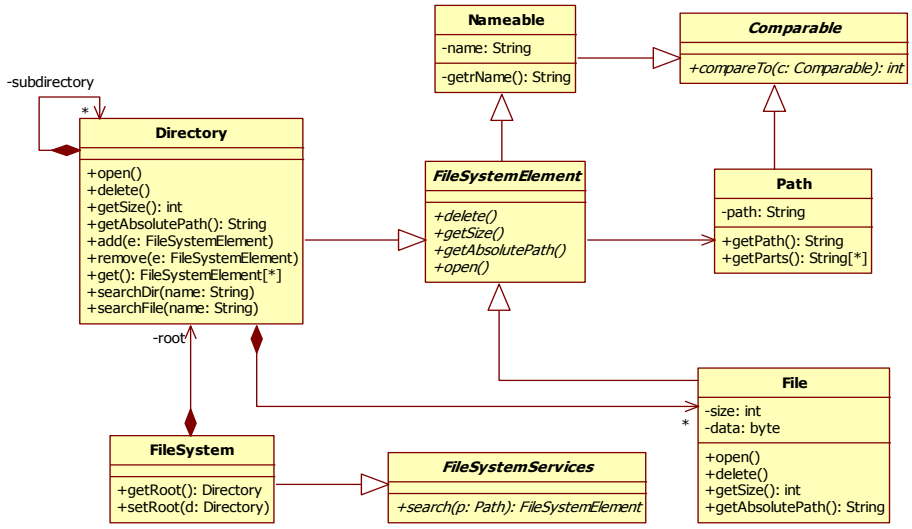


Fig. 7. Model to Review: File System Management

This static UML model represents a basic architecture for a File System Management. Authors of this model are interested in the presentation of some object concepts:

- Inheritance between classes and abstract classes. A uniform protocol for every *FileSystemElement* is encapsulated by a corresponding abstract class. *Directories* and *Files* must respect this protocol via inheritance relationship. We can note that all concrete classes are derived directly or indirectly from an abstract class. This rule enforces the emergence of reusable protocols.
- Management of references, here composition links, between container and components. A *Directory* object manages some references to *Files* and *Directories* objects.

Nevertheless, this model contains a misconception. Although there is a uniform protocol owned by the class *FileSystemElement*, the management of composite links along a hierarchical structure is duplicated. Indeed, *Directory* class manages independently links on *Files* and *Directories*. Now, we consider two evolution scenarios. The first is adding new Terminal types in the tree structure, for example, symbolic links in UNIX systems. This evolution requires the management of this new type of links by the *Directory* class and then requires code modification and code duplication in this class. The second is adding new non Terminal types in the tree structure, for example archive files in UNIX or in Java environment. We can consider that an archive file has the same functionalities as a *Directory*. This evolution requires a reflexive link on an archive file class and the duplication of all links that represent composition links in the tree structure. Then it requires duplication of management of composition and modification in the *Directory* class, it must manage another type on *FileSystemElement*. These two scenarios show a decoupling problem (each container manages a part of the composite structure) and an extensibility limitation (it requires existing code modification for adding new type of terminal or non terminal element of the composition hierarchy). Therefore, this model can be improved. Furthermore, when the authors have implemented this model, they realized that there were defects, and they adapted their code to improve it.

4.1 Object-Oriented Quality Checking

Visually, there is no design mistake: each class of the model presents a reusable protocol. Composition links are used here as delegation between *Directory* and *File*. And messages sent between them have the same selector.

4.2 “Alternative Models” Detection

This step consists in the execution of all queries corresponding at each “alternative model” of the base. In this example, the query of the fifth *Composite* “alternative model” returns theses matching classes:

1. The *Directory* class is able to play the role of the *Composite* class.
2. The *File* class is able to play the role of the *Leaf* Class.
3. The *FileSystemElement* is able to play the role of the *Component* class.

This means that we detected an "alternative model" for the *Composite* pattern because they have the same structural features (cf. Fig. 8).

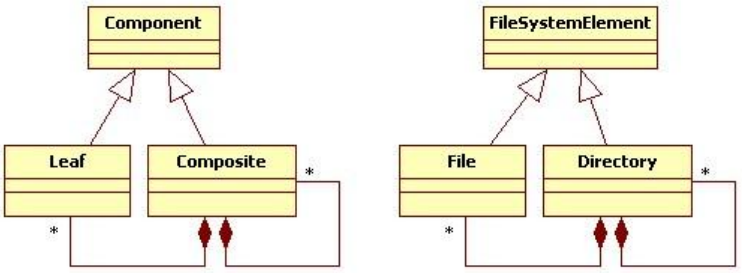


Fig. 8. The fifth *Composite* "Alternative Model" its Instantiation in the Model

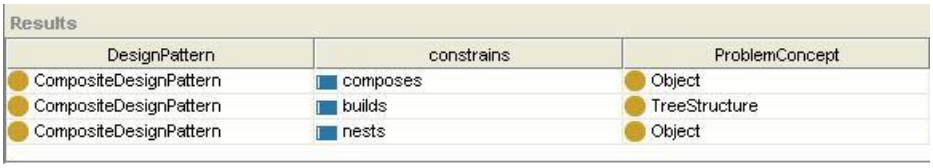
4.3 Designer/Machine dialog

At this step, the designer must verify the substitutability of the detected fragment. Firstly, he must verify if the intent of the fragment matches with the proposed design pattern. To do so, we build a question thanks to a SPARQL query we have coded (cf. Listing 1). This query retrieves the intent of the design pattern in using the "alternative model" detected (here *Composite_AM_5*). Indeed, we consider that the intent of the pattern is described with a list of couples (constraint – ProblemConcept) in the ontology (see Fig. 5).

```
SELECT ?DesignPattern ?constrains ?ProblemConcept
WHERE{
          ?DesignPattern rdfs:subClassOf ?x.
          ?x rdf:type owl:Restriction.
          ?x owl:onProperty :replace.
          ?x owl:someValuesFrom: Composite_AM_5.
          ?DesignPattern rdfs:subClassOf ?y.
          ?y rdf:type owl:Restriction.
          ?y owl:onProperty :isSolutionTo.
          ?y owl:someValuesFrom ?pbconcept.
          ?pbconcept rdfs:subClassOf ?z.
          ?z rdf:type owl:Restriction.
          ?z owl:onProperty ?constrains.
          ?z owl:someValuesFrom ?ProblemConcept.
}
```

Listing 1 SPARQL query to retrieve the intent of the *Composite* pattern that could replace the "alternative model" *Composite_AM_5*

Based on the results (cf. Fig. 9) of this query, we will proceed in dialoguing the designer with the first question: *We have detected in your design an alternative model of the CompositeDesignPattern. Is the fragment {FileSystemElement, File, Directory} composes Object, builds TreeStructure and nests Objects?*


Results

DesignPattern	constrains	ProblemConcept
CompositeDesignPattern	composes	Object
CompositeDesignPattern	builds	TreeStructure
CompositeDesignPattern	nests	Object

Fig. 9. Screenshot of Protégé after executing the query (Listing 1)

We can note that the intent of {*FileSystemElement, File, Directory*} is a recursive composition: "*Directories* are composed with *Files* or *Directories* which are composed with…". So the answer to the previous question is positive.

Now, we must check the interest to replace the fragment with the pattern. Thanks to the perturbation of the "strong points", we can present to the designer the advantage to use the pattern. We retrieve the perturbed "strong points" with a SPARQL query (Listing 2):

```
SELECT ?Strong_Points ?Sub_Features
WHERE{
          :Composite_AM_5 rdfs:subClassOf ?x.
          ?x rdf:type owl:Restriction.
          ?x owl:onProperty :perturbs.
          ?x owl:someValuesFrom ?SF.
          ?SF rdfs:subClassOf ?SP.
          ?SP rdfs:comment ?Strong_Points.
          ?SF rdfs:comment ?Sub_Features.
} ORDER BY ASC(?Strong_Points)
```

Listing 2 SPARQL query to retrieve the "strong points" perturbed by *COMPOSITE_AM_5*

The second question is built with the results (cf. Fig. 10) of the previous query: *Our analysis shows that you have problems of "Decoupling and Extensibility"; your model is unable to satisfy those points:*

1. *Maximal factorization of the composition.*
2. *Addition or removal of a leaf does not need code modification.*
3. *Addition or removal of a composite does not need code modification.*

In injecting the CompositeDesignPattern, you will improve all of these points. Do you want to refactor the identified fragment {FileSystemElement, File, Directory} ?

Results

Strong_Points	Sub_Features
Decoupling and Extensibility	Maximal factorization of the composition
Decoupling and Extensibility	Addition or removal of a Leaf does not need code modification
Decoupling and Extensibility	Addition or removal of a Composite does not need code modification

Fig. 10. Screenshot of the result window presenting the "strong points" perturbed

As we consider that the model may evolve, it is useful to guarantee that there are extensibility and decoupling possibilities. Therefore, the fragment must be substituted with the pattern.

4.4 Patterns Integration

At this step, the identified fragment is replaced by the suggested design pattern like the Fig. 11 below:

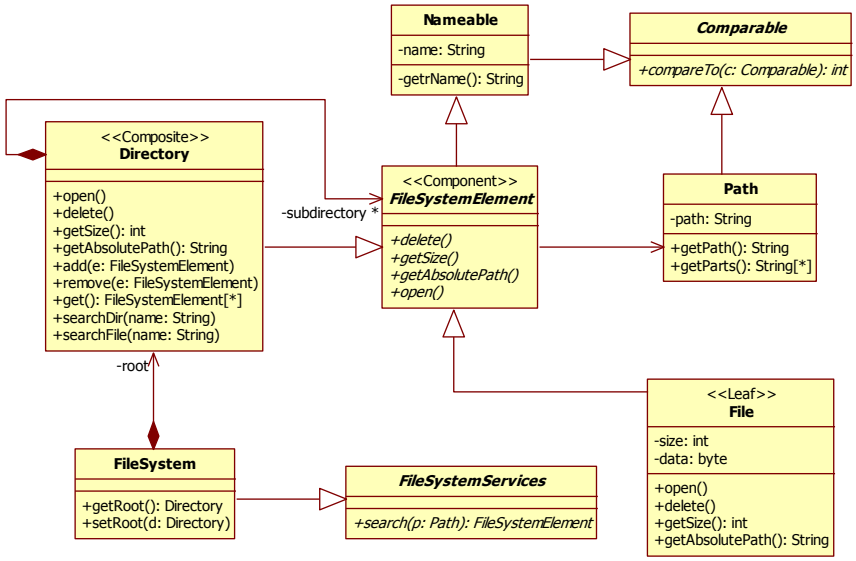


Fig. 11. Model to Review Improved

To do so, a suite of simple model refactoring suffices to integrate the pattern. Here, it consists in:

- Remove composition link between Directory and File.
- Move the end of the recursive composition link from Directory to FileSystemElement.

These inter-classes refactorings can be automatically deduced with an operation of "differentiation" between the "alternative model" and the pattern structure.

At the end of the activity, we can say that this model is improved, because we have substituted a fragment (with "weak points") with a pattern (with "strong points"). This transformation may appear as non fundamental in the model, but at the code level, the implications are substantial. Every hierarchy traversal methods are simpler to implement, and there is less code to write. Moreover, in case of extensions, there is no code modification of existing classes.

5. Conclusion and Perspectives

The approach of reusing and extending an existing ontology corresponding to our requirements was successfully applied. From the existing DPIO ontology, we have plugged our concepts on "alternative models" and "strong points". These concepts are fundamental for tooling our Design Review Activity. Accurately, at the step named validation of substitution propositions, we have simulated a dialog with a designer by interrogating the extended base using queries. These queries will be generated automatically by a template process. The integration of this work into a tool dedicated to the design review activity is envisaged.

Finally, we conclude with some perspectives:

- Take into consideration the relationships between patterns. For example, the *Decorator* pattern can be applied to the *Composite* pattern structure.
- Take into consideration the applicability of each pattern. For example, referring to the GoF book, one of the applicability of the Composite pattern is: *you want clients to be able to ignore the difference between compositions of objects and individual objects*. We notice that this sentence cannot be part of the pattern intention but can be considered as a "strong point".
- Optimize our knowledge base by sharing common "strong points" between patterns. For example, the Composite, the Decorator and the Bridge pattern have a same "strong point" concerning the maximal factorization between specific classes.
- Use inference rules to find new concepts when adding new "alternative models" or "strong points". This could help us improving our knowledge on patterns and particularly, our knowledge on the good practices on object oriented architecture.

Acknowledgements

We are grateful to Mrs. Nathalie Aussenac-Gilles for her precious advices during this work.

References

1. Fowler M., “Analysis patterns: reusable objects models”, Addison Wesley Longman Publishing Co, Inc., 1997.
2. Gamma E., Helm R., Johnson R., Vlissides J., “Design Patterns: Elements of Reusable Object-Oriented Software”, Addison Wesley Professional, 1995.
3. Buschmann F., Meunier R., Rohnert H., Sommerlad P., Stal M., “Pattern-Oriented Software Architecture”, John Wiley & Sons, August 1996.
4. Dunsmore A.P., “Comprehension and Visualisation of Object-Oriented code for Inspections”, Technical Report, EFoCS-33-98, Computer Science Department, University of Strathclyde, 1998.
5. Bouhours C., Leblanc H.., Percebois C., “Alternative Models for a Design Review Activity”. In : *Workshop on Quality in Modeling* - ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, NASHVILLE, TN (USA), 30/09/2007-05/10/2007, Ludwig KUZNIARZ, Jean-Louis SOURROUILLE, Miroslaw STARON (Eds.), Springer, p. 65-79, October 2007.
6. Kampffmeyer H., Zschaler S., Engels G., Opdyke B., Schmidt D. C., Weil F., “Finding the Pattern You Need: The Design Pattern Intent Ontology”, in *MoDELS*, Springer, 2007, volume 4735, pages 211-225.
7. Guéhéneuc Y. G., Albin-Amiot. H., “Using Design Patterns and Constraints to Automate the Detection and Correction of Inter-Class Design Defects”, in *Proceedings conference TOOLS*, July 2001, pages 296-305.
8. Chikofsky E. J., Cross J. H., “Reverse engineering and design recovery: A taxonomy”, in *IEEE Software*, 7(1), page 13-17, January 1990.
9. Bouhours C., Leblanc H., Percebois C., “Alternative Models for Structural Design Patterns”, research report, IRIT/RR--2007-1--FR, IRIT, December 2007, http://www.irit.fr/recherches/DCL/MACAO/docs/AlternativeModelsForStructuralDesignPatterns.pdf.
10. D.L. McGuinness and F. van Harmelen: OWL Web Ontology Language Overview, 2004.http://www.w3c.org/TR/owl-features/
11. Huston B., “The effects of design pattern application on metric scores”, in *Journal of Systems and Software*, 58(3), Elsevier Science, September 15, 2001, pages 261-269.
12. Dietrich, J., Elgar, C.: A formal description of design patterns using OWL, in: Australian Software Engineering Conference (ASWEC'05), pp. 243-250. IEEE Computer Society, Los Alamitos, 2005. http://doi.ieeecomputersociety.org/10.1109/ASWEC.2005.6
13. Tichy, W.F.: A catalogue of general-purpose software design patterns. In: TOOLS'97. Proceedings of the Tools-23: Technology of Object-Oriented Languages and Systems, IEEE Computer Society, Washington, DC, USA, 1997.
14. Noy, N.F., McGuinness, D.L.: Ontology development 101: A guide to creating your first ontology. Technical Report KSL-01-05, Knowledge Systems Laboratory, Stanford University, Stanford, CA, 94305, USA, March 2001.
15. Protégé ontology editor and knowledge acquisition system (2006). http://protege.stanford.edu/
16. Prud'hommeaux E., Seaborne: SPARQL Query Language for RDF, January 2008. http://www.w3.org/TR/rdf-sparql-query/

Using Ontologies in the Domain Analysis of Domain-Specific Languages

Robert Tairas[1], Marjan Mernik[2], Jeff Gray[1]

[1] University of Alabama at Birmingham, Birmingham, Alabama, USA
{tairasr,gray}@cis.uab.edu

[2] University of Maribor, Maribor, Slovenia
marjan.mernik@uni-mb.si

Abstract. The design stage of domain-specific language development, which includes domain analysis, has not received as much attention compared to the subsequent stage of language implementation. This paper investigates the use of ontology in domain analysis for the development of a domain-specific language. The standard process of ontology development is investigated as an aid to determine the pertinent information regarding the domain (e.g., the conceptualization of the domain and the common and variable elements of the domain) that should be modeled in a language for the domain. Our observations suggest that ontology assists in the initial phase of domain understanding and can be combined with further formal domain analysis methods during the development of a domain-specific language.

Keywords: Domain Analysis, Domain-Specific Languages, Ontology

1 Introduction

The development of a Domain-Specific Language (DSL) requires detailed knowledge of the domain in which the language is being targeted. Paradigms such as *Generative Programming* [3] and *Domain Engineering* [5] also require an understanding of the target domain, which is done through a process called domain analysis that produces a domain model. An important theme in the domain analysis used by both paradigms is the need to determine elements that can be reused. The reusable components or software artifacts form the building blocks for developing new software systems. In DSL development, in addition to the overall knowledge of the domain, the domain model can reveal important properties that will influence the way the language is shaped. In particular, the search for reusability in domain analysis can be translated into realizing the *commonalities* and *variabilities* of a domain. This information can assist in pointing out elements in the domain that can be fixed in the language and those that must provide for variabilities; hence, domain analysis has the potential to be beneficial if used during DSL development. However, clear guidelines for the use of established domain analysis techniques in the process of DSL development are still lacking [11].

Ontology development is one approach that has contributed to the early stages of domain analysis [5]. This paper investigates the use of ontology during domain analysis in DSL development and how it contributes to the design of the language. The rest of the paper is organized as follows: Section 2 describes the potential connection between ontology and DSL development. Section 3 provides a case study on the use of ontology in the development of a DSL for air traffic communication and Section 4 provides some observations on ontology in DSL development based on the case study. Related work, a conclusion, and future work are described in Sections 5 and 6.

2 Early Stage DSL Development

Chandrasekaran et al. [2] propose two properties related to ontologies: the first is a representation vocabulary of some specialized domain. This vocabulary represents the objects, concepts, and other entities concerning the domain. The second is the body of knowledge of the domain using this representative vocabulary. This knowledge can be obtained from the relationships of the entities that have been represented by the vocabulary. Ontologies seek to represent the elements of a domain through a vocabulary and relationships between these elements in order to provide some type of knowledge of the domain.

An interesting connection can be observed between ontology and DSL design. As it relates to DSL development [11], a domain model is defined as consisting of:

- a domain definition defining the scope of the domain,
- the domain terminology (vocabulary, ontology),
- descriptions of domain concepts, and
- feature models describing the commonalities and variabilities of domain concepts and their interdependencies.

Not only is an ontology useful in the obvious property of domain terminology, but the concepts of the domain and their interdependencies or relationships are also part of the properties of an ontology [2]. The knowledge of the commonalities and variabilities of the domain concepts can further provide crucial information needed to determine the fixed and variable parts of the language. This part is a more open question as to the potential of finding commonalities and variabilities through information obtained from the ontology.

As it relates to the DSL development process as a whole, the insertion of ontology development in the early stages of DSL development can potentially provide a structured mechanism in the part of DSL development that is still lacking attention. The early stages of DSL development (i.e., domain analysis) have not received as much attention compared to the latter stages of development (i.e., language implementation). Various DSL implementation techniques have been identified in [11], including interpreter or compiler development and embedding in a General-Purpose Language (GPL). In contrast, only four out of 39 DSLs evaluated in [11] utilized a more formal domain analysis, such as FAST [14] and FODA [8]. These formal approaches have shown to result in good language design, but their use is still

limited and it has yet to be seen how well they will be adopted by the community. The question is whether other less formal approaches, such as Object-Oriented Analysis (OOA) or ontology, can be reused in the early stages of DSL development. In order to promote interest in the domain analysis stage of DSL development, this paper advocates the use of ontology in DSL development, which is observed through a case study of a DSL for air traffic communication.

3 Case Study

Ontology development to assist in the design of a DSL is described through a case study in this section. Section 3.1 provides a summary of the air traffic communication problem domain. The ontology and its related competency questions are given in Sections 3.2 and 3.3. The development of a class diagram, object diagram, context-free grammar, and sample program related to the DSL and obtained from the ontology is given in Section 3.4.

3.1 Air Traffic Communication

A case study was selected to apply the ontology development process and observe its usefulness in domain analysis related to DSL development. The case study selected focuses on the communication that occurs between the air traffic control (ATC) at an airport and the pilot of an aircraft. More specifically, the communication is between the ground controller that is responsible for the traffic between the runways and the ramps containing gates in an airport, and the pilots of an aircraft that has just arrived or is in the process of departure. The purpose is to develop a DSL that can standardize the language for the communication between the two individuals. English is the standard language in this domain, but more often the controllers or pilots of non-English speaking countries may experience problems communicating in English. A DSL that standardizes the communication can be translated into the native tongue of the controller or pilot for better comprehension. A separate functionality could check and verify the path that is given to a pilot by a ground controller. An example communication sequence that highlights the potential communication problem is given in Listing 1. The controller is asking the captain to hold short of taxiway "MikeAlpha," but the pilot continually assumes it is taxiway "November."

Listing 1. Example of air traffic communication

```
  ATC:  Make the right turn here at Juliette. Join Alpha. Hold short
        MikeAlpha.
Pilot:  Right on Juliette hold sh ... Taxi Alpha. Hold November [...] Can
        we taxi now?
  ATC:  Make the right turn here at Juliette. Join Alpha. Hold short of
        MikeAlpha.
Pilot:  Roger, join right Juliette. Join Alpha. Hold short November.
  ATC:  OK, I'll say it again. Hold short of Mike Alpha "M" - "A"
        MikeAlpha, not November.
Pilot:  OK, hold short of MikeAlpha.
```

3.2 Ontology Development

Following the ontology development process outlined by Noy and McGuinness [13], competency questions are selected that serve as the purpose of the ontology. In order to obtain a domain model as defined in Section 2, two competency questions were selected: "What are the concepts of the domain and the interdependencies of these concepts?" and "What are the commonalities and variabilities of the domain?"

Both the Ontolingua[1] and DAML[2] ontology libraries were searched for existing ontologies related to the domain in this case study, but no related instances contained the vocabulary necessary for the domain. Although a new ontology is needed for this case study, the availability of an existing ontology in other cases provides a head start to the development of a domain model as the important terms and relationships have been determined already for the domain and can be used toward the subsequent steps of DSL development.

Table 1. Listing of classes and associated slots

Class	Description	Slots		
		Name	Description	Values
`Aircraft`	Arriving or departing aircraft	`Airline ID`	Name of the airline	Two letters
		`Flight Number`	Flight Identification	Integer
`GroundControl`	Controller in charge of airport ground traffic			
`Tower`	Controller in charge of take-offs and landings			
`Runway`	Available take-off and landing locations	`Runway Number`	Runway Identification	1 – 36 (i.e., runway heading 10° – 360°)
		`Runway Orientation`	To distinguish parallel runways	Class `Left` or `Right`
`Taxiway`	Paths connecting runways to ramps	`Taxiway Name`	Taxiway Identification	One or two letters (digits)
`Ramp`	Aircraft parking area	`Ramp Name`	Ramp Identification	String
`Gate`	Passenger embarkation and disembarkation	`Gate Letter`	Gate Identification	One letter
		`Gate Number`	Gate Identification	Integer
`Turn`	Command to turn	`Direction`	Turning direction	Class `Left` or `Right`
		`Taxiway`	Taxiway Identification	Class `Taxiway`
`HoldShort`	Command to hold short of a runway or taxiway	`Runway`	Runway Identification	Class `Runway`
		`Taxiway`	Taxiway Identification	Class `Taxiway`
`Contact`	Command to contact a separate controller	`ATC`	Controller to contact	Class `Tower` or `GroundControl`
`Follow`	Command to follow behind another aircraft	`Aircraft`	Aircraft Identification	Class `Aircraft`

[1] Ontolingua Ontology Library, http://www-ksl.stanford.edu/knowledge-sharing/ontologies/html
[2] DAML Ontology Library, http://www.daml.org/ontologies

Utilizing the tool introduced by Noy and McGuinness [13] called Protégé 2000[3], the ontology for the case study was developed. The terms in Protégé 2000 are stored as classes. This allows for terms to be considered subclasses of other terms. In addition to classes, Protégé 2000 also contains slots and instances. Slots are the properties and constraints of the classes. Slots define the properties of classes and also determine the values that can be set for each property. Instances are actual instances of the classes in the ontology. These can be used to determine how well the ontology is representing a domain.

Table 1 contains a selection of classes and slots of the ontology that was developed in Protégé 2000 for the case study. In addition to the classes and slots in Table 1, instances of these classes were also determined. These instances are based on the information from a simplified diagram of the Birmingham International Airport (BHM) as shown in Figure 1. For example, instances of the `Runway` class are 6, 24, 18, and 36. Instances of the `Taxiway` class are A, B, F, G, H, M, A1, A2, A3, A4, B1, G1, H2, and H4. The `Ramp` class consists of Cargo and Terminal.

3.3 Competency Questions Revisited

The usefulness of the ontology in Table 1 can be measured by how well the ontology assists in answering the previously specified competency questions from Section 3.2. Regarding the first question, the ontology provides the concepts of the domain through the classes. The interdependencies between the concepts can be derived from the constraints of the slots of the classes. For example, the `HoldShort` class is dependent on either the `Runway` or `Taxiway` classes, as this command is always followed by the location in which the pilot is to hold short.

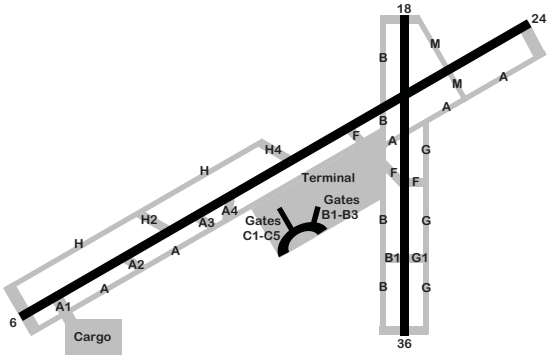


Fig. 1. Simplified Diagram of Birmingham International Airport (BHM)

Answering the second question related to commonalities and variabilities is less evident if observing only the ontology's structure of classes and slots. Information regarding the variabilities can be extracted by including the instances of classes, such

[3] Protégé 2000, http://protege.stanford.edu

as the instances from BHM. Classes `Runway` and `Taxiway` consist of many instances, which could mean these classes have the variabilities property. Moreover, instances that represent airports other than BHM will also contain different values for these classes, which could also be interpreted as containing variabilities. The classes not containing instances, such as most of the commands (i.e., `Turn`, `HoldShort`, and `Contact`), could be interpreted as common concepts in all instances. These commands are common in the ATC domain and represent standard commands that are used in all airports. However, the specific airport elements (i.e., collection of instances of runways and taxiways) may change depending on the airport.

3.4 Conceptual Class Diagram

The ontology process is similar to the process of object-oriented analysis [1]. However, one distinction is that ontology design is mainly concerned with the *structural* properties of a class, whereas object-oriented analysis is primarily concerned with the *operational* properties of a class [13]. The focus here is a methodology that can assist in determining the domain concepts for DSL development by reusing an approach from general software engineering.

Figure 2 presents a conceptual class diagram that was manually generated from the structural information of the classes in the ontology from Table 1. In this case, the development of the class diagram has been assisted by the information obtained from the ontology. In Figure 2, similar classes are grouped together. For example, classes `Gate`, `Ramp`, `Runway`, and `Taxiway` represent physical structures in the airport. Such groupings identified the need for a generalized class for each group. A generalized class was included in the diagram for `Runway` and `Taxiway`, because from the slot properties of class `HoldShort`, two possible values can be used (i.e., `Runway` and `Taxiway`). In the diagram, this is represented by abstract class `Way`. The classes at the bottom of the diagram represent communication commands. These are associated with other classes through their respective slot properties. Generalizations such as `Command` and `Way` were not part of the original ontology and were only introduced during the development of the class diagram. These classes in turn can be used to update the ontology to further improve the structure of the ontology. This can be seen as part of the process of iteratively refining the ontology to better represent the domain.

From the class diagram in Figure 2, an initial context-free grammar (CFG) for the DSL can be generated, as shown in Listing 2. This CFG was manually obtained from the conceptual class diagram to CFG transformation properties defined in [12]. Relationships such as *generalization* and *aggregation* in the class diagram are transformed into specific types of production rules in the CFG. For example, a generalization where classes `Runway` and `Taxiway` are based on class `Way` is transformed into the production rule `WAY ::= RUNWAY | TAXIWAY`. An aggregation where class `Gate` is part of class `Ramp` is transformed into the production rule `RAMP ::= GATES`. In this case the non-terminal `GATES` is used, because the cardinality of this aggregation is zero or more gates on a ramp (i.e., 0..*). An additional production rule is generated to represent this cardinality (i.e., `GATES ::= GATES GATE | ε`).

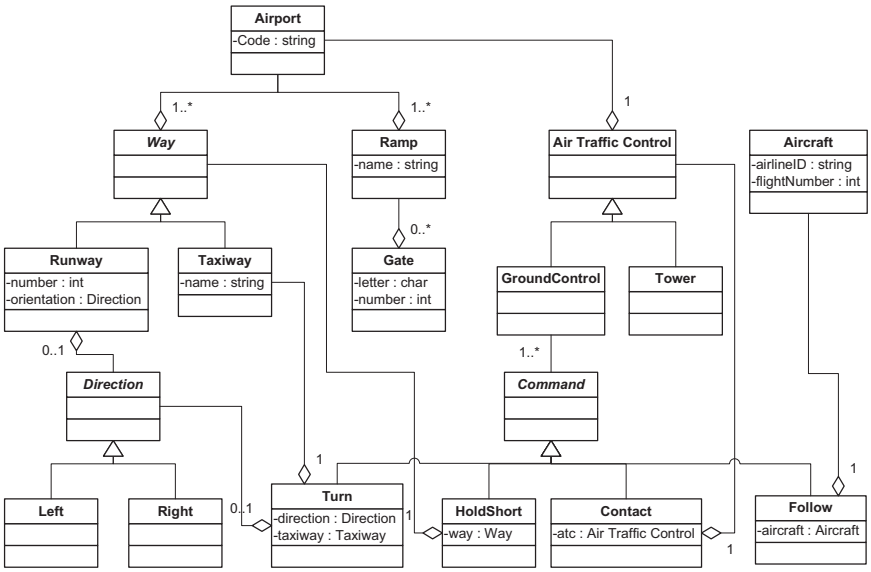


Fig. 2. Conceptual class diagram obtained from the ontology

Listing 2. Transformation of conceptual class diagram to context-free grammar

```
AIRPORT        ::= WAYS RAMPS ATC
WAYS           ::= WAYS WAY | WAY
WAY            ::= RUNWAY | TAXIWAY
RUNWAY         ::= number DIRECTION
TAXIWAY        ::= name
RAMPS          ::= RAMPS RAMP | RAMP
RAMP           ::= name GATES
GATES          ::= GATES GATE | ε
GATE           ::= letter number
ATC            ::= GROUNDCONTROL | TOWER
GROUNDCONTROL  ::= COMMANDS
COMMANDS       ::= COMMANDS COMMAND | COMMAND
COMMAND        ::= CONTACT | FOLLOW | HOLDSHORT | TURN
CONTACT        ::= ATC
FOLLOW         ::= AIRCRAFT
HOLDSHORT      ::= WAY
TURN           ::= DIRECTION TAXIWAY
DIRECTION      ::= LEFT | RIGHT | ε
AIRCRAFT       ::= airlineID flightNumber
```

The transformation of the class diagram into the CFG above, albeit manual, followed a predefined collection of transformation rules. The manual transformation of the ontology into the class diagram is less formal, but was done by connecting the properties of the classes in the ontology with the graphical representation of the class diagram. In order to provide a more automated transformation between the ontology and the class diagram, developing a transformation between an Web Ontology Language (OWL) instance for the ontology and a textual representation of the class diagram could be considered. Related to this, UML-based ontology development has been proposed [6]. Specifically for this case, the transformation between an XML-based OWL file into a class diagram represented in XMI could assist in the

automation of the ontology to class diagram step. After the transformation to a CFG, some keywords have been added to the CFG for easier human parsing, as shown in Listing 3.

Listing 3. Addition of keywords and production refactoring

```
AIRPORT        ::= WAYS RAMPS ATC
WAYS           ::= WAYS WAY | WAY
WAY            ::= runway RUNWAY | taxiway TAXIWAY
RUNWAY         ::= number DIRECTION
TAXIWAY        ::= name
RAMPS          ::= RAMPS RAMP | RAMP
RAMP           ::= ramp name GATES
GATES          ::= GATES GATE | ε
GATE           ::= gate letter number
ATC            ::= GROUNDCONTROL | TOWER
GROUNDCONTROL  ::= COMMANDS
COMMANDS       ::= COMMANDS COMMAND | COMMAND
COMMAND        ::= CONTACT | FOLLOW | HOLDSHORT | TURN
CONTACT        ::= contact ATC
FOLLOW         ::= follow AIRCRAFT
HOLDSHORT      ::= hold short WAY
TURN           ::= turn DIRECTION on TAXIWAY
DIRECTION      ::= left | right | ε
AIRCRAFT       ::= airlineID flightNumber
TOWER          ::= tower
```

An example of a program written in this DSL is shown in Listing 4 and is based on the CFG of Listing 3. Even from this simple DSL for ground control, it can be seen that some simple verification of aircraft path control can be checked. The development of the DSL has been aided by the ontology that was initially produced, which in turn assisted in the generation of a class diagram. This provided a means to understand the domain in the early stages of DSL development, which provided input to the subsequent structure of the DSL, as seen in the grammar in Listing 2.

Listing 4. An example program

```
// description of BHM airport
runway 6 runway 24 runway 18 runway 36
taxiway A taxiway A1 taxiway A2 taxiway A3 taxiway A4 taxiway B  taxiway B1
taxiway F taxiway G  taxiway G1 taxiway H  taxiway H2 taxiway H4
ramp Cargo
ramp Terminal gate B1 gate B2 gate B3 gate C1 gate C2 gate C3 gate C4 gate C5

// commands from Ground Control
turn right on A
turn left on M
hold short runway 18
contact tower
```

An object diagram of the example program in Listing 4 is illustrated in Figure 3. Airport-related structures such as gates, ramps, runways, and taxiways are represented by multiple objects that will differ among various airports. However, the types of commands issued by the ground control remain the same. The specific attributes of the command objects are based on the objects of the structures of a particular airport, e.g., taxiway A and M, and runway 18. As described in Section 3.3, evaluating the instances of the classes provides information regarding the elements of the domain that are common (or fixed) and those that are variable.

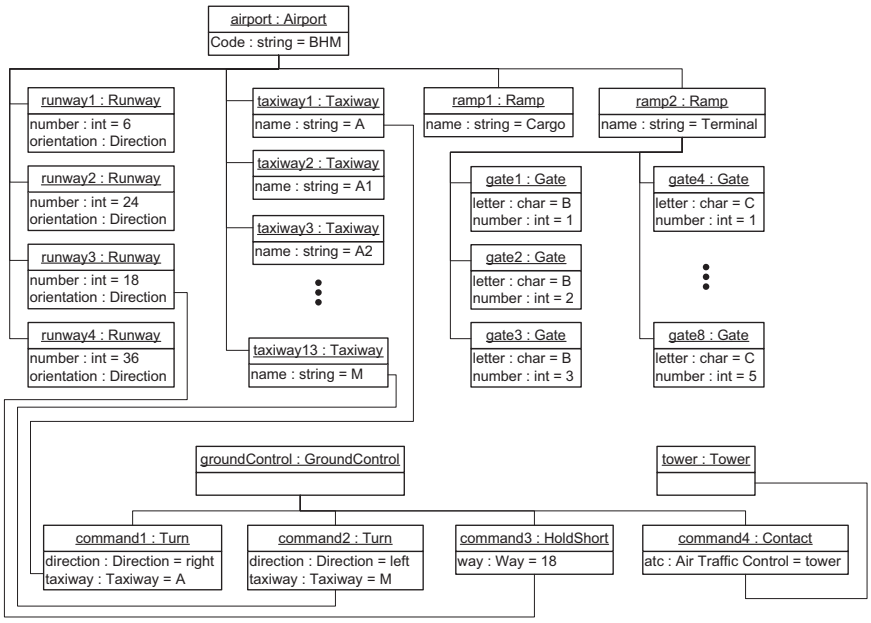


Fig. 3. Object diagram from example program

4 Ontologies in DSL Development

Section 3 summarized the development of a preliminary ontology using the standard development process as seen in literature using a well-known tool called Protégé 2000. The usefulness of the ontology was measured by answering several competency questions that were selected to match the goals of domain analysis. Domain concepts and their interdependencies were determined. In addition, commonalities and variabilities as they relate to the DSL can be determined by observing the instances of the classes in the ontology. It should be noted that the ontology and class diagram went through several iterations before reaching the state described in Section 3. However, further refinements may help to provide more satisfactory answers to the competency questions. The ontology was then used to manually generate a conceptual class diagram, which in turn produced an initial context-free grammar for the proposed DSL.

The case study has shown the potential usefulness of ontology in the development of a DSL specifically during the early stages of development. An ontology can provide a well-defined and structured process to determine the concepts of a domain and the commonalities and variabilities for a DSL, which can result in the generation of a class diagram and subsequently a CFG from the information. Two further observations highlight the benefits of an ontology-based approach. First, if an ontology is already available for a domain, then this existing ontology can be used to

initiate the development of a DSL without the need to start from scratch. This was not the case for the air traffic communication domain described in Section 3, but ontologies for other domains could already exist and be utilized in the DSL development for those domains. Second, the entire process outlined in Section 3 could be used as an alternative to a more formal domain analysis technique such as FODA. In a separate direction, the ontology alone can be combined with formal domain analysis techniques (e.g., proposed by Mauw et al. in [10]) and be used as a supplier of a well-defined input of domain concepts and relationships for further analysis.

5 Related Work

De Almeida Falbo et al. describe the use of ontology in domain engineering that has the purpose of developing software artifacts for *reuse* [5]. A more recent publication demonstrates the use of ontology in engineering design requirements capture [4]. Both cases propose methodologies of utilizing ontology in terms of providing the knowledge about a specific domain, albeit more in a general case of engineering. However, the utilization of ontology in domain analysis in these works translates well to the similar effort in DSL development. Guizzardi et al. associate ontology with the development of Domain-Specific Visual Languages (DSVL) [7]. The ontology is used to assist in developing a representative language for a specific domain that is correct and appropriate. Similarly, our initial investigation described in this paper utilizes ontology as part of the main goal of developing a representative language for a domain such as air traffic communication. However, in addition to this, the common and variable elements of the domain are also considered through the ontology in order to determine the structure of the domain-specific textual language (i.e., fixed and variable language constructs).

Gašević et al. describe efforts to associate the two technical spaces of Model-Driven Architecture (MDA) and ontology, which include the utilization of MDA-based UML in ontology development [6]. We follow a similar approach where a connection is made between the ontology in Table 1 and the UML class diagram in Figure 1. However, in addition to this association, we perform manual transformations on the class diagram to obtain a context-free grammar for the DSL.

6 Conclusion and Future Work

An initial investigation of the usefulness of ontology in domain analysis in DSL development was described in this paper. A case study demonstrated the insertion of ontology development in the DSL development process, where a class diagram was obtained from the ontology and subsequently a CFG was produced. The ontology assisted in answering questions related to the domain, such as the main concepts and their interdepencies, and the common and variable parts related to the DSL. The ontology also provided a structured input to the subsequent stages of DSL development. Continued exploration of ontology-driven domain analysis may provide further proof of effectiveness in the analysis of domains for DSL development.

The class diagram in Figure 2 that was generated from the ontology can also serve as the basis for creating a metamodel. Slight adaptations of this diagram could represent the metamodel for a tool like the Generic Modeling Environment (GME) [9], which provides a domain-specific modeling language that has a concrete syntax that resembles concepts from the domain. Thus, the results of the domain analysis and the observed ontology can inform technologies of both grammarware and modelware. This direction will be explored as future work. In addition, the transformations that were performed were done manually based on predefined transformation properties. A possibility for a more automated step is the transformation of the Web Ontology Language (OWL) representation into a Backus-Naur Form (BNF) representation for the DSL. Such a transformation may map similar elements and perform some alterations between the representations. This direction will also be considered in future work.

Acknowledgments. This project is supported in part by NSF grant CPA-0702764.

References

[1] Booch, G.: Object-Oriented Development. IEEE Transactions on Software Engineering 12, 211--221 (1986)

[2] Chandrasekaran, B., Josephson, J., Benjamins, V.: What Are Ontologies, and Why Do We Need Them? IEEE Intelligent Systems 14, 20--26 (1999)

[3] Czarnecki, K., Eisenecker, U.: *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Boston (2000)

[4] Darlington, M., Culley, S.: Investigating Ontology Development for Engineering Design Support. Advanced Engineering Informatics 22, 112--134 (2008)

[5] De Almeida Falbo, R., Guizzardi, G., Duarte, K.: An Ontological Approach to Domain Engineering. In: International Conference on Software Engineering and Knowledge Engineering (SEKE), pp. 351--358, Ischia, Italy (2002)

[6] Gašević, D., Djurić, D., Devedžić, V.: *Model Driven Architecture and Ontology Development*. Springer, Berlin (2006)

[7] Guizzardi, G., Ferreira Pires, L., van Sinderen, M.: Ontology-Based Evaluation and Design of Domain-Specific Visual Modeling Languages. In: International Conference on Information Systems Development, Karlstad, Sweden (2005)

[8] Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University (1990)

[9] Lédeczi, Á., Bakay, Á., Maróti, M., Völgyesi, P., Nordstrom, G., Sprinkle, J., Karsai, G.: Composing Domain-Specific Design Environments. IEEE Computer 34, 44--51 (2001)

[10] Mauw, S., Wiersma, W., Willemse, T.: Language-Driven System Design. International Journal of Software Engineering and Knowledge Engineering 14, 625--664 (2004)

[11] Mernik, M., Heering, J., Sloane, A.: When and How to Develop Domain-Specific Languages. ACM Computing Surveys 37, 316--344 (2005)

[12] Mernik, M., Črepinšek, M., Kosar, T., Rebernak, D., Žumer, V.: Grammar-Based Systems: Definition and Examples. Informatica 28, 245--255 (2004)

[13] Noy, N., McGuinness, D.: Ontology Development 101: A Guide to Creating Your First Ontology. http://www-ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness.pdf.

[14] Weiss, D., Lay, C.: *Software Product Line Engineering*. Addison-Wesley, Boston (1999)

MDE for publishing Data on the Semantic Web

Guillaume Hillairet, Frédéric Bertrand, Jean Yves Lafaye

{guillaume.hillairet01, fbertran, jylafaye}@univ-lr.fr
Laboratoire Informatique Image Interaction,
University of La Rochelle, FRANCE

Abstract. Publishing local data on the Semantic Web entails providing a shareable semantic data representation. We present a complete MDE approach that allows importing data sources into an RDF repository. At a preliminary stage, the object domain model is mapped to an ontology and to a persistence model. Specifying mappings requires a model annotation performed by a domain expert. All other processes are automated and data transformations are generated from the mappings via model weaving techniques.

Keywords: MDE, Semantic Web, Ontology, RDF

1 Introduction

Due to the expansion of the Semantic Web [2], three technical spaces presently rise up in the information system design landscape. The object space, which evolves into the Model Driven Engineering (MDE) space, allows implementing software applications and uses several languages and libraries. The second space concerns data persistence, mainly through relational databases management systems. The third space deals with resource description and is the core of the Semantic Web. It allows publishing RDF data on the Web as well as OWL ontologies, and ensures that machines can interpret and combine data.
Publishing existing data on the Semantic Web supposes defining bridges between technical spaces. Data persistency is mainly achieved through relational databases systems. In order to ease the use of relational data in enterprise application, several approaches, mainly based on the Active Record pattern [9], propose an object relational mapping (ORM) solution [1] [4]. This is an example of such a bridge, tending to cope with the so-called ‘impedance mismatch’ between distinct formalisms. More recently, some academic and industrial tools started offering quite homologous solutions for bridging relational and RDF data representations [3] [5] [10] [14].
Our opinion is that the gap between relational and RDF data models, is too wide for being crossed over in a single step, i.e. with a direct mapping. In fact, we advocate for introducing an object-oriented domain model as an intermediate between the relational and RDF layers. Despite the differences existing between the object and RDF model [16], their similarities (notion of classes and class hierarchy) remain higher than those existing between the relational and the RDF model. We assume that

the first step which maps the relational and object models is already conveniently addressed by the literature (as evoked above). So, we can focus on the second step that aims at filling the gap between object and RDF data models. The core of our proposal consists of specifying the mapping between the object-oriented domain model and one or more ontologies. The general use case for our work is the following: within the context of Software and Information System design, we assume that an object-oriented domain model does exist, which stands as a reference model for both software applications and database management systems. In order to raise interoperability and data sharing, we propose an object-ontology mapping tool that allows two kinds of data access. The first one follows the ETL [13] (extract, transform and load) pattern for publishing objects (possibly loaded from relational database) as RDF triples. The second data access offer by our tool is an on-demand mapping that translates SPARQL [19] queries into HQL [1] (Hibernate Query Language) that can be executed over the object-oriented domain model.
The remainder of the paper is organized as follows. Section 2 presents our motivation for developing this work and provides a small example that will be kept as an illustration until the end of the paper. Section 3 describes the overall architecture of our proposal. Section 4 outlines the mapping language between the object and ontology modelling spaces and gives some examples. Section 5 presents the ETL approach for publishing objects and relational data as RDF resources. Section 6 presents the on-demand mapping implementation. Related works are quoted in Section 7 before concluding in Section 8.

2 Motivation

Developing the Semantic Web entails publishing existing relational database content in an OWL/RDF format. In this paper, we present a model driven approach for publishing data that have been created, or loaded, by object-oriented applications and that are finally stored in RDF repositories. We show how MDE allows a better coping with such an architectural complexity. Most of the computer applications presently rely on object-oriented modeling, while most of the data are stored in relational databases. We take account of this situation and propose to use the object-oriented domain model as the basis for defining a mapping between object-oriented data and RDF data sources.
Before going further, let's clarify our insight on the usage of domain ontologies and object-oriented domain models. Despite one common objective which is to capture the main concepts of a domain, and in addition to technical and syntactic discrepancies, the viewpoints and final use are differing. The object-oriented domain model is a basis for designing both robust software application and persistent data layers. The matter is not so to give unambiguous definitions of terms (which are usually shared by domain users) as to list the prominent elements and specifies the constraints and mutual relationships [15]. Domain ontology, also capture main concepts and terminology but are fitted to reasoning, browsing and querying which are the facilities required in the semantic web context. Conversely, they generally are not adapted to software and database design. The solutions brought to ontology data

persistency are efficient for semantic queries but lack integrity constraint definition and checking; they are not convenient for update insert and overall delete operations. Practically, the question of creating RDF data (individuals) conforming to a special ontology arises in two contexts

- In case an existing ontology is available that fits the data, direct data sharing is facilitated. Since existing ontologies essentially are consensual, then local data then prove to be presented through a shared knowledge view and vocabulary.
- In case no existing ontology is satisfactory, a special new ontology has to be created so as to account for the local data semantics. However, in order to ensure that this new ontology semantics is not ambiguous and can practically be shared, all genuine concepts should extend actual concepts of an existing ontology. This is easily achieved by using such ontology matching facilities as, for instance, those provided in OWL.

Since an ontology may capture some aspects of the data but fails to capture others, one domain model may need to be mapped to several ontologies. Whatever the choice, defining a correspondence between the ontologies and the data to be published is mandatory. We claim that the domain model is a key representation, standing as a pivot model halfway between the persistence and ontology models. Let's notice that the persistence layer may indifferently be implemented in any manner, say relational or XML, with no special impact upon the overall approach.

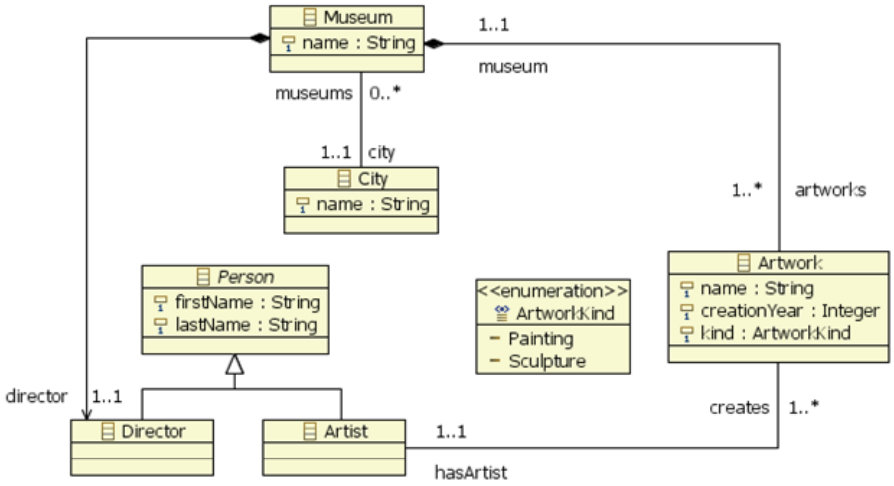


Fig. 1: An object-oriented domain model for a cultural application *(excerpt).*

A short example depicted in Fig. 1 shows the domain model of a museum while Fig. 2 gives an excerpt of the object-ontology mapping between the domain model and several ontologies. Let's now consider the museum example, and illustrate our overall approach. Using appropriate functions provided by modeling tools, the object-oriented domain model may lead to provide a relational schema which is made of tables that are associated to domain model classes. The resulting database is populated by domain data. In order to publish the data on the Semantic Web, an

ontology is needed that will account for the concepts underlying the domain model. Fig. 2 shows what links can be specified when weaving the domain model and the ontology to be generated. In this example, the resulting ontology merges three existing ontologies *(e.g. foaf, dbpedia, geonames)* and a novel one *(museum)*.

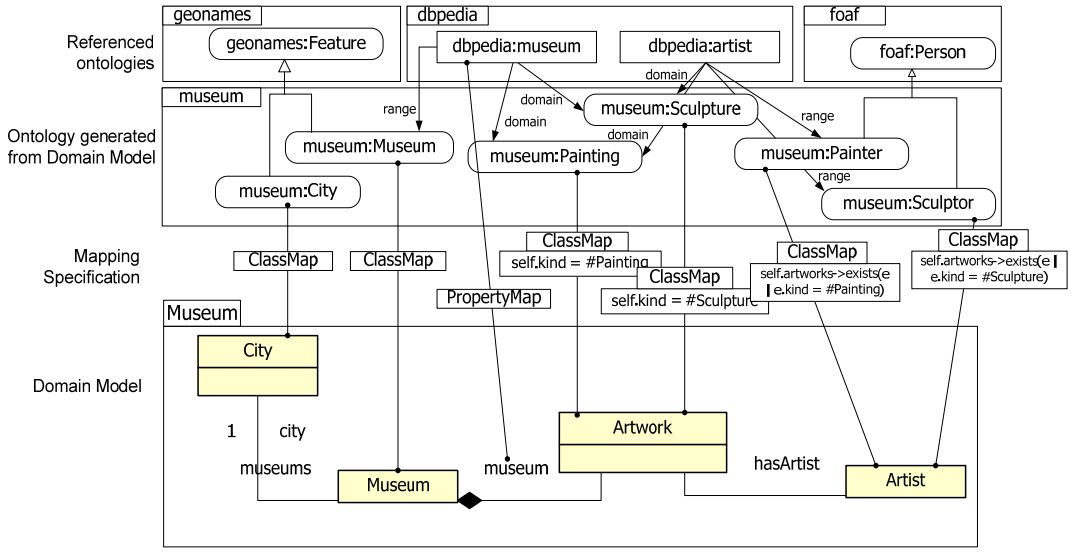


Fig. 2: Mapping the object-oriented domain model with ontologies.

3 Overall Approach

We already evoked the object-relational mapping that links the domain model to the corresponding database. We pointed out the so-called impedance mismatch. We take advantage of the important academic work on this subject and do not account for how the object-relational mapping is achieved. Conversely, we focus on the object-ontology mapping which is quite similar and has less been studied. Our approach can be split into three steps:

1. Specifying the correspondence between domain model elements and ontology concepts by means of the mapping language we developed. This step requires a human domain expert who operates in a specially designed software environment that aids and controls his actions. The role of the expert is mainly to select, constrain and combine elements in the domain model, using operators such as restriction, intersection, union, subsumption, equivalence… and map or create corresponding concepts in the ontology. The result is a mapping specification.
2. Actual generation – on the basis of the mapping specification - of the ontology that describes the data semantics.

3. Populating the knowledge base which is associated to the ontology (individuals) from the relational data. This step involves both the object-relational and object-ontology mappings.

According to our approach, the object-relational mapping also is represented by a weaving model. It links the domain model to the database. We choose to consider the domain model as a metamodel (EMF model) at the M2 level (MDE). Doing so, the domain data may be represented at the M1 level. Then, domain model instances appear as M1 model elements that comply to their M2 metamodel. In contrast, using plain UML class diagrams with instances and classes appearing at a same level would have led to an unnecessary complication of the weaving and transformation processes. Our mapping language between an object model and several ontologies is defined by a textual grammar. It is then processed and translated into weaving models, themselves being specified with the AMW model weaver [6] (Atlas Model Weaver).
Importing and transforming the data from the database so as to populate the knowledge base is achieved via a series of model transformations, implemented in ATL [11] (Atlas Transformation Language), as shown in Fig. 3.

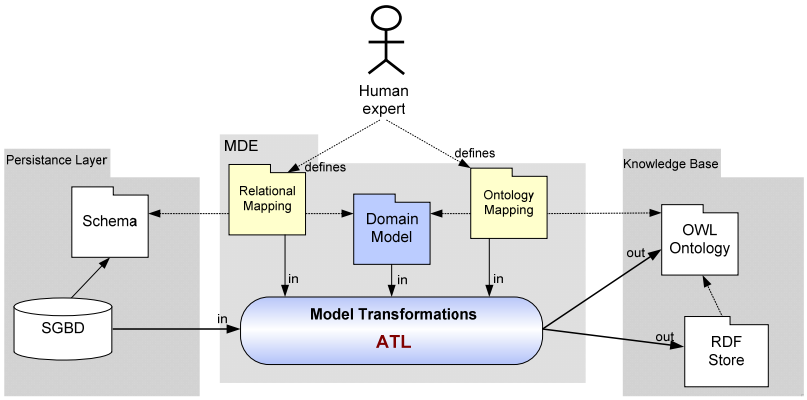


Fig. 3: Linking the modelling spaces to one another

4 Defining ontological views over domain models

This section presents the implementation and use of the mapping language we defined so as to specify the correspondence between the domain model and one (or more) ontology. For sake of conciseness, the complete language grammar cannot be presented here.

4.1 The Domain model / Ontology mapping Language by example

We use the object-oriented domain model in Fig. 1 as an example that outlines our mapping language. The ontology in Fig. 2 appears as a view being defined upon the

domain model thanks to the mapping language. The mapping language implementation we outline hereunder links an EMF metamodel to several OWL ontologies.
A mapping specifies correspondences between compatible elements, such as: a package and an ontology; an EMF class and an OWL class; an EMF attribute and an OWL datatypeProperty, *etc*. More complex mappings are allowed, in order to express 1-N or N-1 correspondences between compatible elements. In the following, we give examples of such simple and complex mappings.

4.1.1 Simple Mapping

A simple mapping accounts for a 1-1 correspondence between a domain model element and an ontology concept, as shown in the listing below. A mapping instance contains the keyword *map* followed by the element to be mapped (*package, class, attribute or reference*). The mapping body is similar to the object language structure, i.e. a class mapping should be nested in a package mapping, a property mapping within a class mapping etc…

```
1   prefix foaf: "http://xmlns.com/foaf/0.1/";
2   prefix geonames: "http://www.geonames.org/ontology#";
3   map package Museum with museum: "http://museum#" {
4     map class Museum with museum:Museum {
5           uriPattern = "http://museum/" + self.name;
6           subClassOf = {geonames:Feature}
7           properties = {
8             map attr name with museum:name;
9             map ref artworks with museum:artworks;
10            map ref city with geonames:locatedIn;
11    }
12  }
13    map class Director with foaf:Person {
14          properties = { map attr lastName with foaf:family_name; }
15    }
16  }
```

Listing 1: An example of simple mappings

The package mapping specifies which the corresponding target ontology is. In the *museum* example, the target ontology is given the same name as the source object model (*museum*), and a special URI. Let's notice that the museum ontology does not yet exist. It is still an abstract ontology in wait of being generated at a further step, in accordance to the specified mapping. When the package is to be mapped to an existing ontology, the latter should be declared via its namespace in the prefix header of the mapping. In the example above, the FOAF ontology[1] is referenced that way, and all FOAF concepts hence are made available for further mappings. EMF class mapping examples are given, such as for *Museum* and *Director*, which are respectively mapped to the museum class in the novel ontology and to Person in FOAF. A class mapping comprises the following clauses:

[1] http://foaf-project.org

- **uriPattern:** specifies the URI of the RDF resource corresponding to the ontology class target. The URI definition is specified via an OCL expression that returns a String. OCL allows browsing the domain model for pruning relevant attributes on which OCL functions can be applied for eventually build the required URI pattern.
- **subClassOf** is an OWL keyword (from the ontology matching language) and here indicates that the mapped element in the new ontology refers to a yet existing ontology concept. Other OWL keywords can be used to express other appropriate kinds of relationship. (e.g. : *museum* in our *museum* ontology is a kind of *Feature* in the *geonames* ontology)
- **properties**: specifies the correspondences between properties of the model class (attribute, reference) and the ontology properties. The property mapping clause distinguishes between *map attr* that links an *attribute* to a *datatypeProperty* and *map ref* that links a *reference* (association) to an *objectProperty*.

4.1.2 Complex Mapping

A complex mapping represents a 1-N or N-1 correspondence between model and ontology elements. Our language, accepts these mappings in a simple way. Complex mappings appear as a series of simple mappings being defined within a same context (see Listing 2 where museum is mapped twice).

```
1   map class Artwork (self.kind = #Painting) with museum:Painting {
2     uriPattern = "http://museum/painting/" + self.name;
3     properties = {
4         map attr name with dbpedia:title;
5         map attr creationYear with museum:creationYear;
6         map ref museum with dbpedia:museum;
7         map ref museum with geonames:locatedIn;
8         map ref hasArtist with dbpedia:artist;
9     }
10  }
11  map class Artwork (self.kind = #Sculpture) with museum:Sculpture {…}
12  map class Artist(self.creates->forAll(e | e.kind = #Painting))
    with museum:Painter {
13    uriPattern = "http://museum/person/"
                 + self.firstName + self.lastName;
14    subClassOf = {foaf:Person}
15    properties = {
16        map attr firstName with foaf:firstName;
17        map attr lastName with foaf:family_name;
18    }
19  }
20  map Artist(self.creates->forAll(e | e.kind = #Sculpture))
     with museum:Sculptor {…}
```

Listing 2: A complex mapping example

All Class mapping clauses with source *Artwork* belong to the same context and then define a complex 1-N mapping. More precisely, the Artwork domain class is split into two ontology target classes that distinguish between *painting* and *sculpture*. The opposite mapping is consequently typed N-1. In such complex mappings of classes,

the selection of the subsets that define the concrete classes in the ontology is achieved by means of OCL constraints whose context is the source class in the domain model. Complex mappings may also concern properties. For instance, in Listing 2, the *museum* reference in the *Artwork* class is mapped to both *dbpedia:museum* and *geonames:locateIn* properties. This 1-N mapping allows setting two distinct views (cultural *vs* geographical) upon the same domain model property.

4.2 Weaving between Domain Models and Ontologies

The language presented here, conforms to a textual syntax that aids an expert in specifying correspondences between the domain model and an ontology. We ground our proposal upon MDE principles. The mapping is viewed as a special model, namely a weaving model that records the set of correspondences. Building the weaving model involves several model transformations.

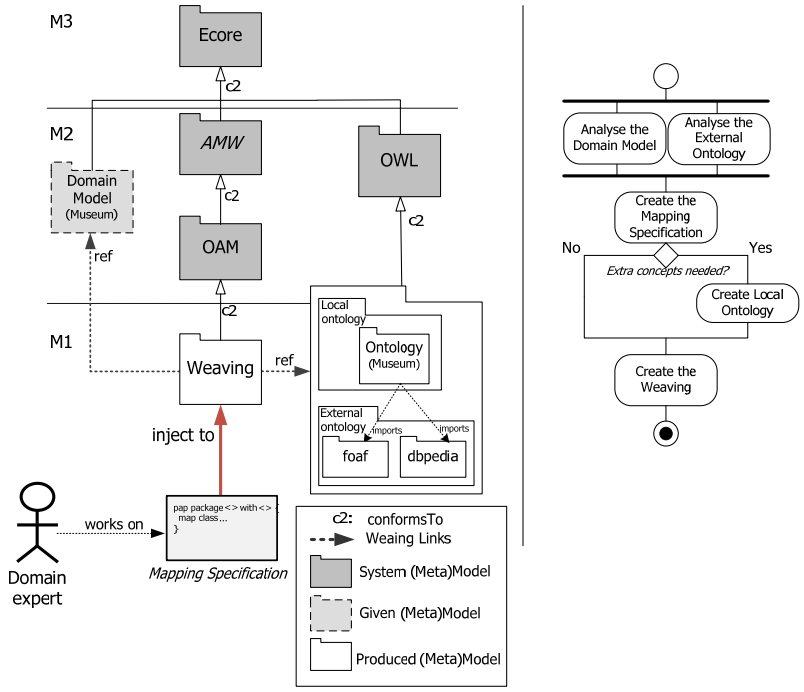


Fig. 4: Several processes are combined for building the mapping file.

1. The mapping possibly is fully defined by the expert. Nevertheless, a basic default mapping may be provided on request. It results from applying standard transformation rules from UML to OWL (UML2OWL[2]), as given in the ODM specification [17]. We implemented these rules *via* a model transformation

[2] http://www.eclipse.org/m2m/atl/usecases/ODMImplementation

process. Starting from this basic original mapping model, the expert is then only in charge of validation and extension tasks, so as to build the final mapping he wants.

2. In case existing ontologies are not sufficient to provide all needed concepts underlying the domain model referenced in the mapping and of interest for data publication, a novel local ontology should be generated. This is achieved by a model transformation that takes both the domain model and the mapping model as inputs and provides the desired OWL ontology as an output.
3. The last step consists in translating the mapping model to a weaving model representation. We use the AMW model weaver environment to implement the weaving process. So, we had to build a weaving metamodel specific to AMW. This process merges a series of transformations (text to model, model to model).

5 Populating the knowledge base

Up to now, we worked at an abstract level, dealing with classes and concepts. From now on, in order to effectively publish data on the Semantic Web, concrete data should be made reachable through a web interface. The natural solution is to make an import from the data source into an RDF repository so that the information to publish could be accessed *via* a web browser. We argued for putting the domain model as a pivot in our method, in consequence, the importation splits up in several consecutive operations we sketch hereafter. First we provide an object view upon the data in the data source according to the domain model, and then we give an ontology view upon the object data. Views are defined as mappings at the data level that lead to weaving models. The weaving models permit to generate transformation rules that can perform the importation process. Using such a high-order transformation is mandatory since the object model at hand cannot be known *a priori*.
The two following subsections respectively address one of the two steps described above. The main benefit of MDE is that it allows linking transformations and dealing in a modular way with the numerous models we need here, with no significant increase of the complexity. Besides, it supplies traceability of the transformations. The example deals with relational persistent data, but dealing with – say – XML data would only require fitting the mapping of the domain model to the new persistent format, all other concerns remaining unchanged.

5.1 From Object data to RDF data

The first step concerns the translation of objects (represented at the M1 layer) into RDF resources. Here we see the benefits of putting the domain model at the M2 layer. Doing so, we have a clear separation between objects (M1 layer) and domain classes (M2 layer). This leads to a simplification of the transformation rules specification. Otherwise, we would have been faced to dealing with UML domain classes and objects at the same level (M1 layer).
The implementation is done using a high-order transformation technique, in order to generate the required transformation. Let *MM2RDF.atl* be the transformation that

converts a M1 model into an RDF model. As depicted in Fig 5, *MM2RDF.atl* is generated by the high-order transformation *OAM2ATL.atl*. The latter takes the mapping model and its related models (domain model and ontology model) as input. It is here made clear that the use of a high-order transformation is inescapable since *MM2RDF.atl* rules depend of the domain model which is a variable model.
The role of *OAM2ATL.atl* is to screen the weaving model, and for each class mapping clause (map class), it generates a specific ATL rule in the *MM2RDF.atl* file. This resulting ATL transformation extracts an RDF data representation from a model conforms to the domain model.

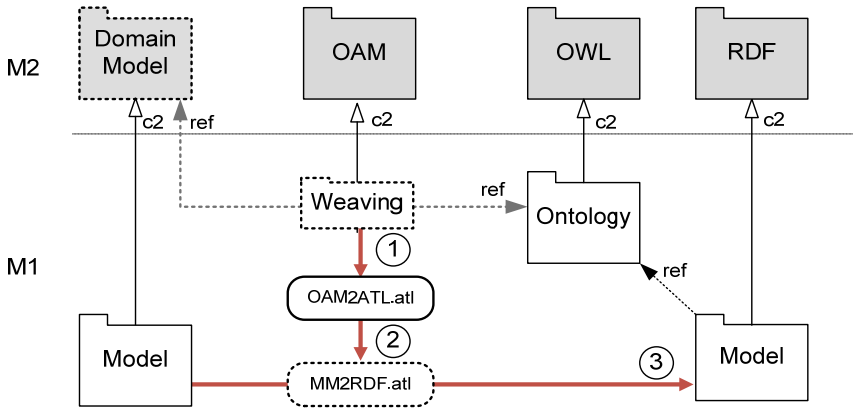


Fig. 5: Model Transformation process for obtaining an RDF repository from a model.

The following listings show some significant excerpts of the *MM2RDF.atl* transformation. Listing 3 shows how an instance of the Museum class is translated into an RDF Resource. For each domain model class that appears in the mapping, an equivalent rule is generated. The RDF metamodel used in this transformation is derived from the ODM specification, and has been extended in order to ease transformation specifications. Each RDF resource is the subject of some statements (see Line 7). For instance, the lazy rules, *makeDataStatement* and *makeObjectStatement*, respectively create the desired statement from a class attribute, and from an association (see Listing 4).

```
1   rule Museum2Resource {
2     from m : Museum!Museum
3     to r :
4       RDF!Resource (
5         uri <- m.getURI(),
6         subjectOf <- Sequence { type,
7           thisModule.makeDataStatement(m, m.name, 'name'),
8           m.artworks->collect(e |
                    thisModule.makeObjectStatement(m, e, 'artworks')),
9           thisModule.makeObjectStatement(m, m.city, 'city'))}
10      ),
```

```
11      type : RDF!Statement ( ... )
```

Listing 3 Generated ATL code excerpt for translating a Museum instance into an RDF Resource.

```
1  lazy rule makeObjectStatement {
2    from s : OclAny, o : OclAny, pname : String
3    to r : RDF!Statement (
4             subject <- s,
5             predicate <- p,
6             object <- o
7           ),
8       p : RDF!Property ( ... ),
9   }
```

Listing 4: Statements and properties are created by lazy rules.

5.2 From Relational data to Object data

In this paper, we assume that a relational database is *in situ* available, whose schema has been obtained by applying an object-relational mapping upon the domain model and the database schema. In our current implementation, we rely on the Hibernate[3] framework, for the object-relational mapping definition (more precisely the TENEO[4] project which is dedicated to EMF persistency). The weaving model that represents the object-relational mapping is thus obtained from the Hibernate mapping definition. The database schema is captured by applying a schema discovery process with the help of the Hibernate framework. It allows defining a projector from the database onto the MDE technical spaces. The concrete data are modelled accordingly, by sending queries to the database. We then apply a model transformation similar to the one presented in section 5.1. The last step that imports relational data into an OWL knowledge base can now be achieved through exploiting the sole object-ontology mapping.

6 Querying the RDF Store

The process we presented above publishes data as an OWL knowledge base. It can be viewed as implementing a data warehouse *via* the Extract-Transform-Load principle (ETL) [13], in which resident data are translated in order to fit the target system format. In our approach, the target system is an OWL ontology whose content is made of RDF triples. Data access is achieved by use of a RDF query language, e.g.: SPARQL [19]. An example of such a SPARQL query is given hereafter. Let's suppose the user wants to list artworks present in "Le Louvre".

```
1  prefix dbpedia: <http://dbpedia.org/property/>
```

[3] http://www.hibernate.org
[4] http://www.elver.org/

```
2  prefix museum: <http://museum#>
3  select ?art
4  where {
5     ?art dbpedia:museum ?museum .
6     ?museum museum:name "Le Louvre"
7  }
```

Listing 5: A SPARQL query on the museum ontology.

Here, we assume that the knowledge base has been generated and completely populated from an existing database. Performing a query then only involves the knowledge base that finally can be a RDF file or a Sesame[5] RDF database.

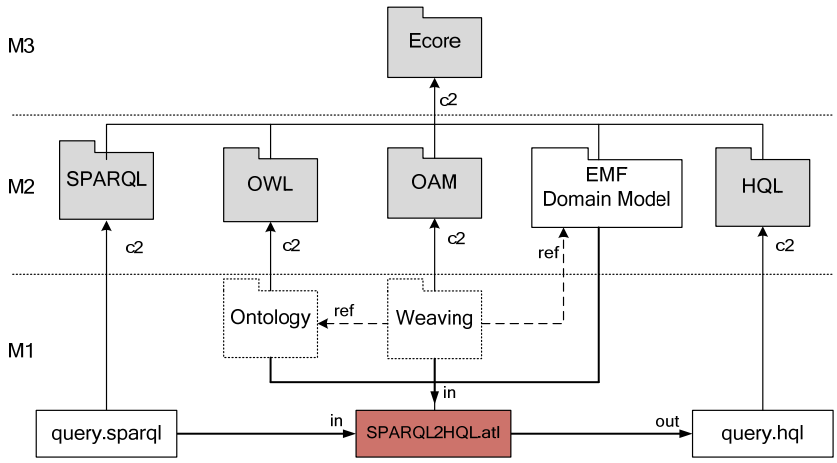


Fig. 6: Model Transformation process for Query rewriting

However, translating and importing the whole data base into an RDF repository, does not seem - in most cases - the best solution, since it implies to replicate data and carry out a synchronization process with heavy redundancy. We advocate for a better practice, which directly answers the ontology query by querying the original data source. Implementing this approach led us to specify an automated rewriting process that relies on the specification of the chain of mappings and finally translates a SPARQL query into a set of SQL statements.

The rewriting process naturally is developed within the MDE paradigm and potentially applies to any persistent data source format. Fig. 6 gives an overview of the series of model transformations that lead to the result. More precisely, on the museum example, the initial SPARQL query is first translated into an equivalent HQL query upon the domain model, which in turn is rewritten as a SQL query on the relational database. The model transformations that are responsible for that rewriting process take both the object-ontology and object-relational mappings and weaving models into account, in order to exploit the element correspondences for translating

[5] http://www.openrdf.org

the terms of one query into its counterpart in the target language. For example, the SPARQL query in Listing 3 gives the corresponding HQL query in Listing 4.

```
1  select art
2  from Museum museum, Artwork art
3  where art.museum = museum and museum.name = 'Le Louvre';
```

Listing 6: HQL query resulting from a rewriting process from SPARQL.

The resulting HQL query can be rewritten by Hibernate according to the data source format (e.g.: SQL).

7 Related Works

The works connected to our approach belong to various but complementary domains. Those that treat of the UML and RDF(S) correspondence [7] [8] [18], and especially the ODM specification from the OMG [17] that sets up basic useful although limited transformation rules, helped us in defining the abstract syntax of our mapping language between an object model and an ontology. Linking metamodels and ontology through weaving models has also been proposed by Klapper [12] who suggests the use of ontology alignment techniques in order to automatically build the weaving model. This seems of interest for us in case we try to map a model to an existing ontology, but is out of concern when creating a new ontology is the goal.
Another kind of related works treats of data integration by means of an ontological view, and more generally of the semantic web. The question still is to implement RDF knowledge bases providing access to existing data source. Bizer with D2RQ [3], Chen [5] and de Laborda [14] come with works having goals very close to ours. They propose a direct mapping of the knowledge base to the data source. Conversely, a part of our contribution is to rely on the domain model (and not on the persistence model) as a pivot element in the process of correspondence design.

8 Conclusion

This paper grounds on the MDE technique and presents a contribution that aids publishing data on the semantic web. A complete process for building a RDF/OWL knowledge base from existing possibly heterogeneous and distributed databases is specified. Our proposal is semi-automated, but the expert is only involved in the mapping definition step (between the domain model and the ontology). One novel aspect of our work is that we give a prominent role to the domain model in the overall process. It comes as a pivot model standing as a midterm between the data source and the ontology. It embodies a sizeable part of the business knowledge and bears more information about the domain terminology and concepts than – for instance - the flat relational model that generally results from complex normalization and de-normalization processes, far from any semantic concern. It also keeps simpler and

focuses on permanent and essential features, while an ontology is intended to also account for subsidiary concerns.
Finally, when splitting the mapping process by introducing the intermediate domain model, we gain in modularity, and robustness. Transformations are more explicit and simpler since the gap is less at each step. We also gain in being less dependent on the persistency technology. In case the database schema is modified, only one transformation step is involved and subject to update.
Our proposal not only permits to create an ontology that describes exported data, but also to create data that conform to an existing ontology. In order to get free from controlling the consistency between the database and the RDF triples, we presently implement an automated 'on the fly' rewriting process that translates SPARQL queries into SQL or XML queries. Consequently, the data can be kept in their original persistence system, with no replication in an RDF repository, while being subject to the queries a user can directly express in terms of the ontology.

References

1. Bauer, C., King, G.: Java Persistence with Hibernate. Manning Publications, 2007.
2. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic Web. Scientific American 284 (2001) 28-37
3. Bizer, C., Seaborne, A: D2RQ-Treating Non-RDF Databases as Virtual RDF Graphs. 3rd International Semantic Web Conference (ISWC2004), Hiroshima, Japan (2004)
4. Castro, P., Melnik, S., Adya, A.: ADO.NET entity framework: raising the level of abstraction in data programming, in: Proceedings of the ACM SIGMOD International Conference on Management of Data, 2007, pp. 1070–1072.
5. Chen, H., Wang, Y., Wang, H., Mao, Y., Tang, J., Zhou, C., Yin, A., Wu, Z.: Towards a Semantic Web of Relational Databases: a Practical Semantic Toolkit and an In-Use Case from Traditional Chinese Medicine.
6. Del Fabro, M.D., Bézivin, J., Jouault, F., Breton, E., Gueltas, G.: AMW: A Generic Model Weaver. Proc. of the 1ères Journées sur l'Ingénierie Dirigée par les Modèles (2005)
7. Djuric, D., Gasevic, D., Devedzic, V.: Ontology Modeling and MDA. Journal of Object Technology 4 (2005) 109-128
8. Falkovych, K., Sabou, M., Stuckenschmidt, H.: UML for the Semantic Web: Transformation-Based Approaches. Knowledge Transformation for the Semantic Web 95 (2003) 92-107
9. Fowler, M.: Patterns of Enterprise Application Architecture. Addison-Wesley, 2002.
10. Hu, W., Qu, Y.: Discovering Simple Mappings Between Relational Database Schemas and Ontologies. ISWC/ASWC 4825 (2007) 225-238
11. Jouault, F., Kurtev, I.: Transforming Models with ATL. Model Transformations in Practice Workshop at MoDELS Vol. 3844, Montego Bay, Jamaica (2005) 128–138
12. Kappel, G., Kapsammer, E., Kargl, and al., M.: Lifting metamodels to ontologies: A step to the semantic integration of modeling languages. ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems, Genova, Italy (2006)
13. Kimball, R., Margy, R.,. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling, 2nd edition, Wiley, 358-362. (2002)
14. de Laborda, C.P., Conrad, S.: Bringing Relational Data into the SemanticWeb using SPARQL and Relational. OWL. IEEE Computer Society Washington, DC, USA (2006)
15. Larman C., Applying UML and Patterns, An introduction to Object-Oriented Analysis and design and The Unified Process 2nd edition, Prentice Hall, 2001

16. Oren, E., Delbru, R., Gerke, S., Haller, A., Decker, S.: ActiveRDF: object-oriented semantic web programming. Proceedings of the 16th international conference on World Wide Web (2007) 817-824
17. OMG. Ontology Definition Metamodel OMG Adopted Specification, November 2007. http://www.omg.org/docs/ptc/07-09-09.pdf
18. Parreiras, F.S., Staab, S., Winter, A.: On marrying ontological and metamodeling technical spaces. Proceedings of the 6th joint meeting of the european software engineering conference and the 14th ACM SIGSOFT symposium on Foundations of software engineering (2007) 439-448
19. Prud'hommeaux, E., Seaborne, A., others: SPARQL Query Language for RDF. W3C Recommendation (2008)

Bringing Ontology Awareness into Model Driven Engineering Platforms[1]

Srdjan Živković, Marion Murzek, Harald Kühn

BOC Information Systems GmbH, Wipplingerstrasse 1,
1010 Vienna, Austria
{srdjan.zivkovic, marion.murzek, harald.kuehn}@boc-eu.com

Abstract. In state-of-the-art of MDE platforms semantic technologies such as ontologies are rarely used. Our aim is to understand the role of ontologies in supporting model-driven engineering, in particular MDE platforms. MDE platforms may benefit from semantic technologies in formal model semantics and automated reasoning on different levels of the metamodelling architecture. We present an ontology-aware MDE platform architecture and outline some application scenarios where ontologies and automatic reasoning may bring benefit to such platforms. Additionally, an example of using ontologies for verification checks of mapping models in the course of metamodel composition is illustrated.

Keywords: metamodelling, model-driven engineering (MDE), ontology-aware architecture, ontology application scenarios.

1 Introduction

In state-of-the-art MDE platforms semantic technologies such as ontologies are rarely used. Currently, such platforms focus on aspects such as metamodelling, metamodel composition, model integration, and model transformation [1]. We are convinced that MDE platforms may benefit from semantic technologies particularly in formalizing semantics of models on different metamodelling levels, which in turn allows for application of automated reasoning.
Based on this hypothesis, in this research-in-progress paper, we discuss first how a MDE platform architecture may be extended to be considered as ontology-aware (section 2). Referring to the introduced architecture, we describe some possible application scenarios where ontologies and automatic reasoning may bring benefit to such platforms (section 3). Out of these scenarios, we highlight in more detail an example from the metamodel composition domain, to illustrate how an ontology-based approach may enhance quality of process by supporting verification checks of

[1] *Acknowledgement: This research has been co-funded by the European Commission and by the Swiss Federal Office for Education and Science within the 7th Framework Programme project MOST N° 216691, http://most-project.eu.*

metamodel mappings (section 4). Finally, we summarize the discussion and give outlook for future work.

2 Architecture of Ontology-aware MDE Platforms

Metamodelling platforms are software environments providing means for the management of models and metamodels. Usually, such model-aware platforms allow for: (1) definition, storage and maintenance of models and modelling languages, (2) execution of mechanisms working on models, metamodels and the meta-metamodel and (3) guidance on how to apply a metamodelling language and/or modelling languages together with corresponding mechanisms to produce metamodels and/or models [2]. Besides these capabilities, metamodelling platforms need to meet other functional and non-functional requirements such as multi-productability, web-enablement, multi-client ability, adaptability, extensibility, scalability and interoperability [3].
The architecture for MDE platforms may be seen as an incarnation of the generic metamodelling platform architecture [2]. Furthermore, adding the ontology aspect, we envision an enriched MDE platform architecture including semantic technologies as depicted in fig. 1.

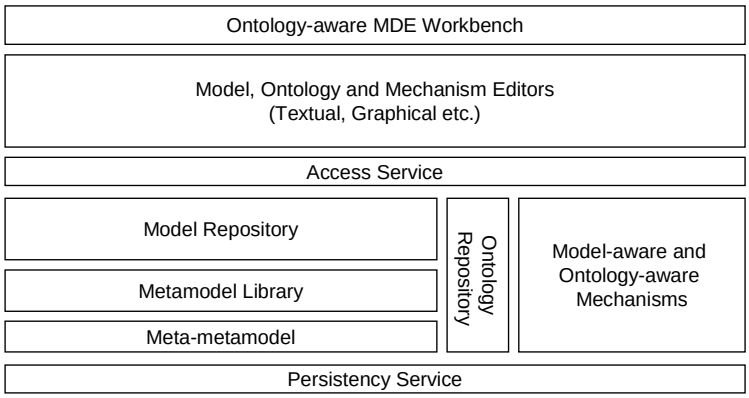


Fig. 1. Logical Architecture for Ontology-aware Model Driven Engineering Platforms

The root core architectural element is the meta-metamodel which defines the concepts available for the definition of modelling languages. Based on it, the metamodel library contains metamodels of defined modelling languages. The metamodel library conforms to the meta-metamodel and, in turn, forms the foundation of the model repository, in which all models are stored.
As an extension to models on different levels, the ontology repository serves as storage of the semantics of models, metamodels and the meta-metamodel. Semantics can be formally described by using the notion of ontology [4]. Reasoning on ontologies is part of the ontology-aware mechanisms.

Model and ontology editors are used for the definition and maintenance of models, metamodels, and ontologies.
All mechanisms used for evaluating and using models are stored in the mechanism base. A fundamental mechanism within MDE environments is model transformation. Further important mechanisms are model/metamodel integration, comparison and mapping mechanisms. The ability to manage different versions of models and metamodels or its parts is another key characteristic of model-aware systems, which should be enabled by means of model/metamodel versioning mechanisms. To support syntactically and semantically correct modelling, validation mechanisms on models and metamodels are used. Furthermore, querying mechanisms of ontology-enriched models and metamodels are needed features to allow various model analyses. Traceability of transformations, which should support guiding the software development tasks, is another needed mechanism within MDE platforms. Mechanism editors are used for definition, configuration and maintenance of mechanisms.
Guidance describes the application of modelling and metamodelling languages and mechanisms. Particularly in context of MDE, this can include guidance on what to consider when defining certain models or guidance on the decision what the next step in a particular model-driven software development process would be. Guidance information can be stored in process models or ontologies, and/or extracted out of existing modelling artefacts and/or traceability links.
Persistency services support the durable storage of models, metamodels, and ontologies. These services abstract from concrete storage techniques and permit storing of modelling information in heterogeneous data sources such as files, databases or web services.
Access services serve two main tasks. On the one hand they enable the open, bi-directional exchange of all metamodel, model and ontology information. On the other hand they cover all aspects concerning security such as access rights, authorization, and en-/decryption.
An ontology-aware MDE workbench serves as a common environment for integrating different editors.

3. Some Ontology Application Scenarios in MDE Platforms

The “ontology” concept, as the literature describes it, seems to enjoy as many definitions as there are attempts to define it. In the course of this paper, we will favour the one we believe to best match the context under consideration. Hence, we understand an ontology as an *explicit conceptual model extended by formal logic based semantics* [5]. Formal semantics expressiveness of ontological models is a de facto advantage compared to conceptual models in software engineering i.e. MDE. Model checking, model enrichment and dynamic classification are some of the identified usage scenarios when thinking about marrying ontological and metamodelling technical spaces [6].
In the following, we describe some scenarios, where ontologies may find its usage within MDE platforms. We concentrate particularly on the following core elements of the MDE platform (see section 2): the model, metamodel and meta-metamodel

element (section 3.1), the mechanism element (section 3.2) and the guidance element (section 3.3). Each section starts by introducing a problem domain, followed by an ontology application scenario description and finalized with a list of possible benefits gained by using semantic technology.

3.1 Ontologies and Models on M1, M2 and M3 Level

Problem Domain: The 3-layer MDE architecture enables definition of modelling languages, and based upon it, creation of models. The M3 model, i.e. the meta-metamodel, defines the syntax of the metamodelling language which is used for modelling metamodels on the M2 layer. Similarly, the M2 model, the metamodel, constructs the syntax of the modelling language for the application domain used on level M1. Even though the abstract syntax is structurally well defined, metamodelling language (M3) and modelling languages (M2) lack well-defined semantics. Currently, language semantics on M3 and M2 level may be expressed algorithmically in the core implementation of the M3 model or in the M2 models. Another approach is to use a declarative language, such as the Object Constraint Language (OCL [7]) to additionally define the semantics of M3 and M2 models.
Furthermore, modelling task on M1 level requires adequate knowledge about the subject under consideration. Such knowledge may currently be captured and reused via reference models or patterns. However, more sophisticated mechanisms to facilitate modelling task semantically are missing, which would prevent heterogeneity problems, model ambiguity etc.
Ontology Application Scenario: The syntax-rich languages (M2 and M3 languages) in MDE platforms may be extended by semantically more expressive ontology languages, in order to take advantage of automated reasoning. There are different approaches, which tackle the problem of converging languages from different technical spaces, such as UML+OWL integration [8]. Having integrated languages, their synergetic effect may be exploited. On the one side, automated reasoning may be used for model consistency checks which may outperform existing solutions in terms of semantic soundness. On the other side, ontology-based approaches may be used both for the definition of modelling languages (M2) and their models (M1). Relying on common domain ontologies in form of machine-readable and reusable domain knowledge, quality of modelled solutions may be raised.
Gained Benefit: Semantically enriched modelling languages and models; machine-readable formal semantics of models; enhanced quality of modelling solutions.

3.2 Ontologies and Mechanisms

Problem Domain: Mechanisms are applied on models residing on different abstraction levels (M1, M2 M3 models). According to the abstraction level, mechanisms may be generic, metamodel specific or hybrid. *Generic mechanisms* are defined on the M3 level, thus being independent of languages defined on the M2 level; e.g. generic import/export interface for model exchange. *Metamodel specific mechanisms* require explicit knowledge about metamodels, in order to work on their

underlying models on M1 level; e.g. business process model simulation mechanism. *Hybrid mechanisms* are a combination of the previous two. They are generic, but they are adaptable to specific metamodels; e.g., model transformation is a hybrid mechanism which uses M3 level constructs to define transformation rules between M2 models, which are, in turn, executed on models on M1 level. Other MDE mechanisms such as model integration, metamodel composition or model comparison fall into this category as well. The challenge of applying metamodel specific or hybrid mechanisms lies in their configuration. For example, to enable a simulation on a specific process language, mappings to generic process concepts need to be defined. Similarly, the specification of metamodel mappings for hybrid mechanisms such as transformation, metamodel composition or model integration is, in most cases, performed manually (see section 4 for a detailed example).
Ontology Application Scenario: Ontologies may be applied to fill the formal semantic gap towards support of automated configuration of metamodel-specific and hybrid mechanisms. For instance, in the case of model transformations, this would mean e.g. an ontology based model transformation approach. On the other side, metamodel specific mechanisms such as simulation would benefit from ontologies, by relying on e.g. generic, machine-readable process ontology. The prerequisite is that different simulation-enabled languages, i.e. process modelling languages have to conform to particular generic process ontology. Consequently, the process of configuring a simulation mechanism for different process languages may be automated by inferring mappings out of ontology.
Gained benefit: Reduced costs through automated configuration of mechanisms; increased potential for reuse; low efforts for substitution and extensions of mechanisms.

3.3 Ontologies and Guidance

Problem Domain: Guidance in the MDE context may include information on what to consider when defining certain models, or information of the next step in the model-driven software development process. Often, execution of a step within the software process is influenced by many factors, such as pre and post-conditions, defined rules etc.
Ontology Application Scenario: Ontologies and automated reasoning may leverage execution of process models, by formalizing its semantics in the form of *process guidance ontologies* that formalize rules, conditions and actions a software engineer has to conduct in specific situations. This way, reasoning technology would infer the next step within the process based on the process guidance ontology and by deriving implicit knowledge from corresponding modelling artefacts.
Gained Benefit: Flexibility of process definitions; enhanced quality of guidance.

4 An Example: “Verification of Mapping Models”

Mappings define correspondences between elements of different models. Particularly, metamodel mappings have an important role in the MDE approach by building

semantic bridges between different metamodels. They add knowledge about integrative usage of different modelling languages, still leaving the integrating languages independent. Bridging of metamodelling and ontology languages is done via mappings [10]. Rules for MDA based model transformations may be built based on existing mappings [11][12]. Furthermore, mappings are used as input for metamodel composition rules by stating about structural-semantic relationships of metamodel elements from different metamodels [13]. However, two problems arise when the mappings are applied: First, discovery of mappings by means of metamodel matching is a complex task, being a tedious work when executed manually and a challenging task for semi-automatic identification [14]. Second, mappings are managed as models and are built based on a mapping language. Thus, mappings need to be verified against their syntax and defined semantics. This may imply not only checking the mapping model, but also crossing the model boarder and diving into the semantics of metamodels being integrated, in order to verify certain mapping statements against e.g. cyclic generalization relationships, multiple inheritances, redundancy etc.

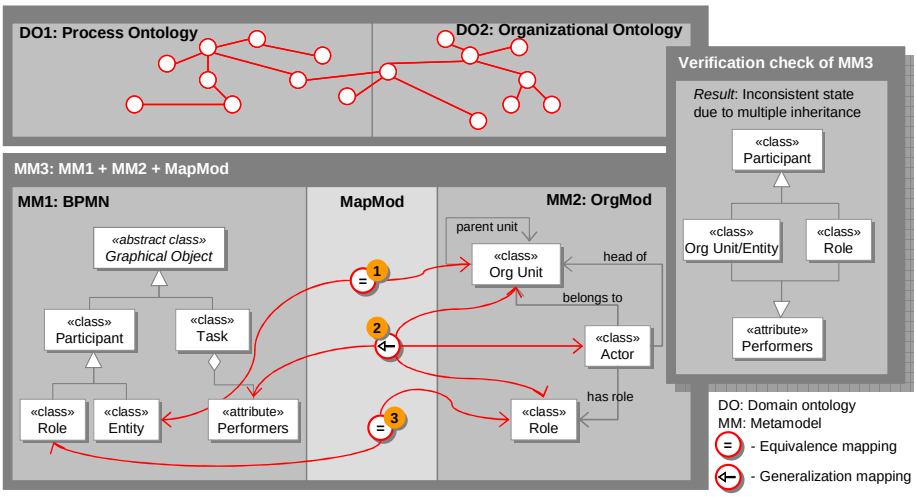


Figure 2 Ontology-aware Modelling and Verification of Mappings for Metamodel Composition

Figure 2 illustrates the simplified case of a metamodel composition using mappings. Let us assume that the MDE platform supports two modelling languages, e.g. BPMN [15] and the ADONIS Language for Organizational Modelling [16], separately. The envisioned integrated metamodel (MM3) should exhibit characteristics of a business process modelling language extended by the concepts for modelling organizational structures (in Fig. 2 as MM1 and/or MM2). A proposed approach is to assemble existing metamodels by utilizing metamodel mappings. The manual process of metamodel matching results in three identified mappings (Fig. 2, 1, 2 and 3), which are candidates for integration points. At first glance, everything seems correct, as the defined mappings conform to the mapping language which uniquely state correlations between elements. However, after generating the integrated metamodel (MM3) (see [13] for detailed integration rule definitions) a verification check founds an

unconformity against the meta-metamodel, which disallows multiple inheritance of elements. The drawback of the solution is, that the verification of a mapping model may only be done after having generated the integrated metamodel, since such cross-model correlations may not be examined in the mapping design time. Here, ontologies may be used guiding both metamodelling and modelling of metamodel mappings. If metamodels are designed with the support of corresponding ontologies, as depicted in the Figure 2, not only the metamodel matching process may be enhanced, but also an early verification of a mapping model in design time against integrated semantics stemming from both problem domains may be possible. In addition, large scale metamodel integration scenarios may especially benefit from the ontology based composition approach reducing ambiguities and improving quality of modelling solutions.

5 Summary and Outlook

In this research-in-progress paper we presented a possible extension of the generic architecture for MDE platforms [3] towards an ontology-aware MDE platform. Based on this architecture and its core elements, possible ontology application scenarios have been discussed. Some of their expected benefits are:

- Semantic-enriched models (on M1, M2, and M3 level).
- Machine-readable formal semantics of models.
- Semi-automated configuration of mechanisms.
- Increased potential for reuse of mechanisms.
- Flexibility of process definitions for guidance.

We are currently working on the refinement of the presented ontology-aware MDE platform architecture to support first prototype implementations. This includes the application of ontologies for enhanced software process guidance.

6 References

1. Bezivin, J., Jouault, F., and Touzet, D. 2005. Principles, Standards and Tools for Model Engineering. In Proceedings of the 10th IEEE international Conference on Engineering of Complex Computer Systems (June 16 - 20, 2005).
2. D. Karagiannis; H. Kühn: Metamodelling Platforms. Invited paper in: Bauknecht, K.; Tjoa, A Min.; Quirchmayer, G. (eds.): Proceedings of the Third International Conference EC-Web 2002 - Dexa 2002, Aix-en-Provence, France, September 2-6, 2002, LNCS 2455, Springer-Verlag, Berlin, Heidelberg.
3. H. Kühn, M. Murzek: Interoperability Issues in Metamodelling Platforms. In: Konstantas, D; Bourrières, J.-P.; Léonard, M.; Boudjlida, N. (Eds.): Proceedings of the 1st International Conference on Interoperability of Enterprise Software and Applications (I-ESA'05), Geneva, Switzerland, February 2005, Springer Verlag, pp. 215-226.
4. Gruber, T. R.: Toward principles for the design of ontologies used for knowledge sharing. In: Guarino, N.; Poli, R. (Eds.): Proceedings of the International Workshop of Formal Ontology, Padova, Italien, August 1993.

5. Oberle, D.: Semantic Management of Middleware, Volume I of The Semantic Web and Beyond Springer, New York (2006).
6. Fernando Silva Parreiras, Steffen Staab, Andreas Winter: On Marrying Ontological and Metamodeling Technical Spaces. 2007. ACM Press. Proceedings of the 6th joint meeting of ESEC/FSE, 2007, Dubrovnik, Croatia, September 3-7.
7. OCL – Object Constraint Language, http://www.omg.org/. December 2007.
8. Kiko, K. and Atkinson, C.: Integrating Enterprise Information Representation Languages. In: Proc. of Int. VORTE Workshop, VORTE 2005, Enschede, The Netherlands, 2005.
9. S. Roser and B. Bauer. An approach to automatically generated model transformations using ontology engineering space. In Proceedings of Workshop on Semantic Web Enabled Software Engineering (SWESE), Athens, GA, U.S.A., 2006.
10. Fernando Silva Parreiras, Steffen Staab, Andreas Winter: TwoUse: Integrating UML Models and OWL Ontologies. Universität Koblenz-Landau, Fachbereich Informatik. 2007. Arbeitsberichte aus dem Fachbereich Informatik.
11. Lopes, D, Hammoudi, S, Bézivin, J, and Jouault, F : Mapping Specification in MDA: from Theory to Practice. In: Proceedings of the First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA'2005). Springer-Verlag, pages 253—264. 2005.
12. G. Kappel, E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, W. Schwinger, and M. Wimmer. Lifting metamodels to ontologies: A step to the semantic integration of modeling languages. In Proc. of MoDELS? 2006, volume 4199 of LNCS, pages 528-□542. Springer, 2006.
13. Zivkovic, S.; Kühn, H.; Karagiannis, D.: Facilitate Modelling Using Method Integration: An Approach Using Mappings and Integration Rules. In: Österle, H.; Schelp, J.; Winter, R.; (Eds.): Proceedings of the 15th European Conference on Information Systems (ECIS2007) - "Relevant rigour - Rigorous relevance", St.Gallen, Switzerland. June 2007, pp. 2038-2050.
14. Kappel, G., Kargl H., Kramler G., Schauerhuber A., Seidl M., Strommer M., Wimmer M., Matching Metamodels with Semantic Systems - An Experience Report, BTW 2007 Workshop Model Management und Metadaten-Verwaltung, Aachen; March 2007.
15. BPMN – Business Process Modelling Notation, http://www.omg.org/spec/BPMN/1.1/. January 2008.
16. BPMS Method Handbook, ADONIS 3.9 Manuals, BOC Group, 2007.

Designing MAS Organisation through an integrated MDA/Ontology Approach*

Daniel Okouya[1] and Loris Penserini[1] and Sébastien Saudrais[2] and Athanasios Staikopoulos[2] and Virginia Dignum[1] and Siobhán Clarke[2]

[1] Universiteit Utrecht, The Netherlands,{maatari,loris,virginia}@cs.uu.nl
[2] Trinity College Dublin, Computer Science, Ireland
{Athanasios.Staikopoulos, Sebastien.Saudrais, Siobhan.Clarke}@cs.tcd.ie

Abstract. The increasing complexity of distributed applications, software services that can be dynamically deployed, adjusted and composed, paves the way for new challenges in software and service engineering. This paper describes a novel approach that combines the flexibility of MDE techniques to deal with the conceptual modelling of MAS and the expressive power of OWL based ontologies to deal with semantics constraints verification as well as domain knowledge provision of MAS models. We will illustrate these ideas through the modeling of a crisis management scenario, using a first prototype of our future Design tool: OperettA.

1 Introduction

Nowadays' distributed applications based on the notion of service-oriented systems –which can dynamically adapt, organise, and compose to satisfy with their networked stakeholders' needs– are fostering the software engineering research area with new challenges. As these distributed systems have to be deployed within real organisational contexts, adhering with organisational rules, and meeting stakeholders' expectations, it is crucial to characterise the software architectural and functional requirements in terms of their correlations with the actual environment. To deliver on this aim, a promising approach in software engineering has been to build methodologies along with conceptual modelling languages that better reflect and describe the complex social and human organisational context where the system-to-be has to be deployed [1][2][3]. For example, in [3] and [4], an ontology based on the Telos language has been presented to describe the conceptual model of the Tropos methodology.

In this paper, we describe some achievements and future improvements of a MAS development framework, OperettA [5], which is based on the OperA methodology [1]. Using a simplified crisis management scenario, we will illustrate how the Operetta's conceptual modelling language –which adheres to a

* This work has been performed in the framework of the FP7 project ALIVE IST-215890, which is funded by the European Community. The authors would like to acknowledge the contributions of their colleagues from ALIVE Consortium (http://www.ist-alive.eu)

Model Driven Engineering approach– can be integrated with ontology representation languages as OWL. Moreover, this approach allows designers for both the verification of the semantics and the provision of domain specific knowledge of the MAS design models.

Model Driven Engineering (MDE) refers to the systematic use of models as primary artefacts throughout the Software Engineering lifecycle. The defining characteristic of MDE [6] is the use of models to represent the important artefacts in a system, be they requirements, high-level designs, user data structures, views, interoperability interfaces, test cases, or implementation-level artefacts such as pieces of source code. The Model Driven Development promotes the automatic transformation of abstracted models into specific implementation technologies, by a series of pre-defined model transformations.

The paper is organised as follows: Section 2 presents the methodological context of our approach with a motivation example. Section 3 provides an overview of our approach. Section 4 summarises the main characteristics and benefits of the approach adopted, which is partially implemented within the OperettA tool.

2 A Methodological Context

2.1 OperA overview

OperA [1] is an engineering methodology based on organisational abstractions, suitable both to model and study existing societies, as well as to develop new systems that participate in an organisational context. The main focus of OperA enable a suitable balance between global aims and requirements agent autonomy, their coordination needs, and environmental stakeholders' needs.

The development framework for agent societies, proposed in OperA, is composed of three conceptual design models: *Organisational Model*, *Social Model*, and *Interaction Model*, as detailed in [1]. In this paper, we illustrate our ideas by only using some concepts and diagrams belonging to the*Organisational Model*. It contains the description of the roles, relations and interactions in the organisation. It is constructed based on the functional requirements of the organisation. The social model and the interaction model are the link between the organisational description and the executing agents.

2.2 Running scenario

Using an example taken from the Dutch procedures for crisis management, we provide a conceptual model based on organisational and social concepts. The modelling phase is conducted according to the OperA methodology [1] using the OperettA tool [5] for graphical representation and verification of the model. This scenario will be used along the paper to explain the development framework properties.

The structure diagram depicted in Figure 1 represents the crisis situation. It specifies the responsibilities and goals of each role, e.g., each time an emergency call occurs to the Emergency_Call_Center, this role alerts the Fire_Station entity, informing about the location in which the (possible) disaster is taking place. The

Fire_Station is responsible to build the appropriate fire brigade and depends on the established Firefighter_Team in order to achieve the objective extinguish the fire. When the team arrives at the accident location, it has to decide (based on its personal experience) the severity of the disaster. Only after this evaluation is reported, an intervention decision is taken. For example, according to local rules, the evaluation should comply with some standardised emergency levels, as established by the Dutch Ministry of Internal Affairs. For the sake of simplicity, we consider that Firefighter_Team sets up a strategic intervention based on the results of two evaluation criteria: *damage evaluation* and *fire evaluation*. Based on the number of wounded, Firefighter_Team decides on the necessity or not to ask for ambulance_service. Moreover, the Firefighter_Team checks if the damage involves building structures in which case police intervention is necessary to deviate traffic. From the fire evaluation criterion, Firefighter_Team can decide whether it is the case or not to ask Fire_Station for a Firefighting_Truck intervention. As described in [1], the Social Structure is further detailed by interaction structure diagrams to model activities among and within roles in order to achieve their objectives. Such activities are called scenes.

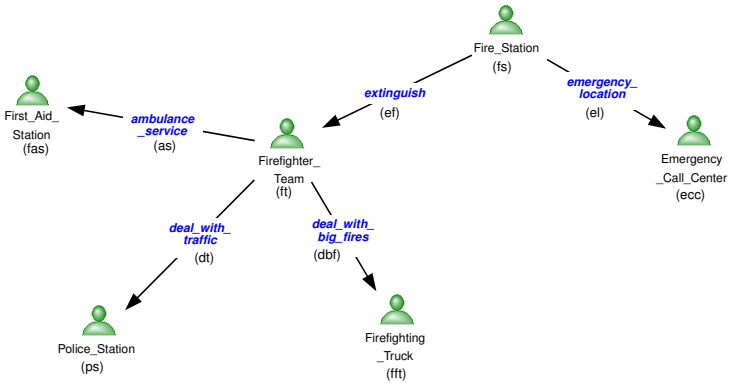


Fig. 1. Social Structure diagram: Fire Station organisation (O) example.

Notice that, the specification provided at this time is not sufficient to give a complete picture about the *know how* required to software systems to achieve the modelled organisational objectives. Nevertheless, the level of abstraction achieved provides enough anchor points for agents to coordinate their activity without fully pre-specifying the capabilities of the agents and therefore limiting flexibility.

3 Approach overview

3.1 A meta-level view of conceptual models

In order to effectively deal with the MAS development, the proposed approach takes into account both the syntax and the semantics of a MAS, through an

integration of MDA with Reasoning and Domain Knowledge specification Based on Ontology. Fig. 2 illustrates the architecture of our approach. The central part corresponds to the OperA metamodel, which provides the syntax of MAS. The right part provides the actual semantics of MAS, which is described by the OperA ontology. The MAS ontology instantiation will be automatically produced from the MAS model, which is created with the OperettA tool. Next, the MAS ontology instantiation will be semantically checked against the OperA ontology, see section 3.2. The left part provides an interaction between existing domain ontologies and MAS models. The interaction is maintained by defining transformations relation between the OperA metamodel and EODM[3]. EODM is an implementation of the ODM standard from OMG, defining metamodels for RDF(S) and OWL, see section 3.3.

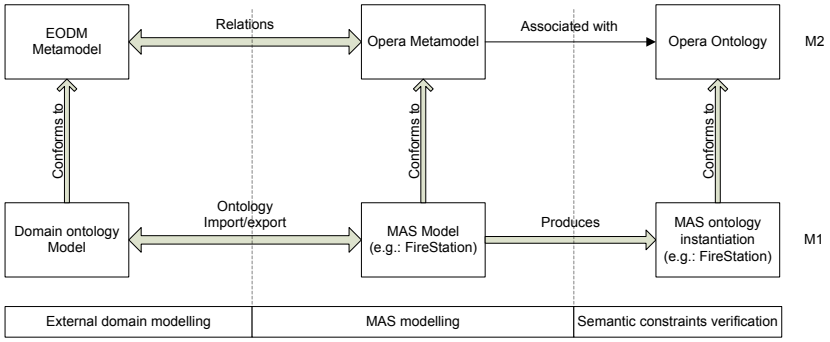


Fig. 2. Overview of our approach.

3.2 Reasoning with models compliant with OperA ontology

The first aspect of our integrated approach is directed toward reasoning on our models using logics, with description logic as our language mainly dedicated at reasoning on the structural aspect of our models. This will provide us with the ability to use the power of descriptive logic along with associated reasoners, combined with techniques to formally analyse our models and enhance their quality.

At the meta-level, the abstract syntax is defined through metamodelling and the semantics is based on description logic. This results in the OperA ontology, which formalises OperA conceptual framework concepts and their relationships, as well as domain independent OperA semantics constraints. Meanwhile, the integration of the metamodelling and ontologies supports domain specific languages and also offers the opportunity to query the models. Indeed, as the OperA metamodel is associated to the OperA ontology, a MAS model is associated to a MAS ontology instantiation. Consequently, the semantics of our models are stored in the MAS ontology instantiation, which is automatically produced using

[3] http://www.eclipse.org/modeling/mdt/?project=eodm

the MAS model (following a straightforward transformation). Hence, it allows for the designer, the verification and validation of models using ontologies; namely, the application of description logics for reasoning about the OperA language. In the following, some examples of semantics constraints, that we are interested in verifying, have been proposed.

- Checking if the objective of a dependency is an objective or a sub-objective of the dependency's initiator. For instance, in the running scenario, this would mean verifying that the role fire-station possesses an objective or sub-objective extinguish fire as it appears to be dependent on Firefighter_Team for it.
- Checking if each dependency is realised by at least one scene.
- Checking if roles are involved in a dependency, do indeed cooperate in at least one realisation scene of that dependency. Again, referring to the scenario, there must be at least one realisation scene of objective deal_with_big_fires dependency in which Firefighter_Team and Firefighter_Truck must cooperate.
- Checking that each role posses at least one dependency link.

The approach described above has been partly implemented in and illustrated by the OperettA prototype [5]. The results obtained with that prototype have encouraged us to move towards a more standardised and accessible version of the tool. The actual version is currently under development using Eclipse.

3.3 Conforming design models to domain ontology

The second aspect of our approach permits the use of ontology within our models as the source of domain specific related knowledge, necessary for the description and support of roles interaction and communication in OperA organisation and at a lesser extend, domain specific semantics constraints enrichment. That is, domain ontologies are integrated at the model level. They are defined, used and imported within the OperA modelling language as well as exported from it. For the latter two, a relation is defined at the meta-level, between the Eclipse Ontology Definition Metamodel and the OperA metamodel enabling the interaction with standard ontology representation languages like OWL. Meanwhile, in the ontological world the integration is at the same level. That is, the OperA ontology and domain ontology are at the same level within logic hierarchy; paving the way for the analysis of the overall structural aspect of the organisation, consisting in querying one knowledge base being the combination of the OperA ontology, the Domain Ontology and the derived MAS ontology instantiation.

Firstly, a vocabulary is available for describing interactions and supporting communications related to a domain. Referring back to our Fire_Station organisation, given the equivalent formal notation for the objective extinguish –e.g., *Extinguish-Fire(L : location)*– of the Firefigther_Team role, the concept of *Location* must be defined in the Domain ontology, otherwise the parameter type will not be available when defining the objective. Furthermore, at run-time, this ontology will be used by members of the organisation in order to communicate.

Secondly, the OperA Ontology can be enlarged by further modelling domain-specific constraints enriching the generic semantic constraints. It provides the opportunity to define modelling rules within the domain ontology as (re)configuration rules e.g., a specific domain could forbid more than 2 dependencies between two specific roles.

This second aspect provides a separation of concerns for knowledge taking into account its domain specific part and enabling at the same time the tool to tailor himself based on it.

4 Conclusions

This paper describes the features of an implemented design tool, OperettA, based on the OperA methodology. This enables the construction of a development framework to support software and services engineering. Besides, this contributes to the achievements of a more general research objective established within the ALIVE project. Specifically, the ALIVE project combines cutting edge Coordination technology and Organisation theory mechanisms to provide flexible, high-level means to model the structure of inter-actions between services in the environment.

The proposed approach (partly) implemented in OperettA deals with techniques to integrate features of the model driven architecture (MDA) with features of the ontology languages (OWL). Main benefits of our approach come from the fact that it provides a model driven approach for the specification of a MAS dealing with its semantics and its provision of domain specific knowledge. Hence, our approach allows designers for powerful reasoning mechanisms to be employed as well as a smart integration of domain specific knowledge that comes first to refine and enrich (along with further constraints) the specification of the design model.

References

1. V. Dignum. *A Model for Organizational Interaction: based on Agents, founded in Logic*. PhD thesis, Universiteit Utrecht, 2004.
2. Brian Henderson-Sellers and Paolo Giorgini, editors. *Agent-Oriented Methodologies*. Idea Group Inc., 2005.
3. A. Fuxman, P. Giorgini, M. Kolp, and J. Mylopoulos. Information systems as social structures. In *FOIS '01: Proceedings of the international conference on Formal Ontology in Information Systems*, pages 10–21, New York, NY, USA, 2001. ACM.
4. J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: representing knowledge about information systems. *ACM Trans. Inf. Syst.*, 8(4):325–362, 1990.
5. D. M. Okouya and V. Dignum. A prototype tool for the design, analysis and development of multi-agent organizations. In *DEMO Session: Proc. of the 7th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS-08)*. ACM, 2008.
6. J. Bézivin, N. Farcet, J.-M. Jézéquel, B. Langlois, and D. Pollet. Reflective model driven engineering. In *UML*, pages 175–189, 2003.