UNIVERSITÄT KOBLENZ · LANDAU

Institut WeST

FB 4 Informatik

Model-driven Generation of APIs for OWL-based Ontologies

Stefan Scheglmann
Ansgar Scherp
Steffen Staab

Nr. 7/2010

Arbeitsberichte aus dem Fachbereich Informatik

Arbeitsberichte des Fachbereichs Informatik

ISSN (Print): 1864-0346
ISSN (Online): 1864-0850

Herausgeber / Edited by:

Der Dekan:
Prof. Dr. Zöbel

Kontaktdaten der Verfasser

Stefan Scheglmann, Steffen Staab, Ansgar Scherp
Institut WeST
Fachbereich Informatik
Universität Koblenz-Landau
Universitätsstraße 1
D-56070 Koblenz
EMail: schegi@uni-koblenz.de, Scherp@uni-koblenz.de, staab@uni-koblenz.de

Model-driven Generation of APIs for OWL-based Ontologies

Stefan Scheglmann and Ansgar Scherp and Steffen Staab
{schegi, scherp, staab}@uni-koblenz.de

WeST, Institute for Web Science and Technologies, Universität Koblenz-Landau,
Universitätstr. 1, 56070 Koblenz, Germany

Abstract. Existing tools for generating application programming interfaces (APIs) for ontologies lack sophisticated support for mapping the logics-based concepts of the ontology to an appropriate object-oriented implementation of the API. Such a mapping has to overcome the fundamental differences between the semantics described in the ontology and the pragmatics, i.e., structure, functionalities, and behavior implemented in the API. Typically, concepts from the ontology are mapped one-to-one to classes in the targeted programming language. Such a mapping only produces concept representations but not an API at the desired level of granularity expected by an application developer. We present a Model-Driven Engineering (MDE) process to generate customized APIs for ontologies. This API generation is based on the semantics defined in the ontology but also leverages additional information the ontology provides. This can be the inheritance structure of the ontology concepts, the scope of relevance of an ontology concept, or design patterns defined in the ontology.

1 Introduction

Ontologies are a powerful means to model and represent semantic, structured data on the web. In order to use this knowledge in concrete applications, appropriate application programming interfaces (APIs) are required. Recently, tools to generate APIs for ontologies such as OWL2Java [5], AliBaba[1], and ActiveRDF [8] have gained widespread popularity. However, the existing tools generate ontology APIs in a naive way.

The main challenge when generating APIs for ontologies is to bridge the gap between the semantic description in the logics-based ontology and the pragmatic handles needed by an object-oriented API. To this end, we have to abstract from the "what" that is defined in the ontology and the "how" to use this ontology through an API. In the naive approach, each ontology concept is mapped to one API class. This only produces concept representations but does not provide information about how to use these representations, i.e., which classes have to be instantiated. Thus, an additional layer is required to sufficiently abstract from the knowledge structure defined by the ontology. This layer defines which objects

[1] `http://www.openrdf.org/doc/alibaba/2.0-alpha4/` last visit June 10, 2010

and properties have to be used together to provide appropriate and easy to use API functionalities. This kind of usable abstraction is not provided by existing tool support for generating ontology APIs. Our experience in developing such usable APIs for our ontologies like the Event-Model-F [12] and X-COSIMO [2] has shown that it is a cumbersome and error-prone process.

To alleviate this situation, we present a multi-stage Model-Driven Engineering (MDE) process to generate customized APIs for ontologies. In order to handle the API generation process, we introduce two models as intermediate steps: The Model for Ontologies (MoOn) bases on the Ontology Definition Metamodel (ODM) [7] and uses its OWL profile to represent ontologies in UML without loosing the semantics. The ODM provides various models for representing different Description Logic (DL) operators. The models for some of these operators like IntersectionOf and Union do not follow the UML specification. Thus, we have developed modified models in order to use them directly in the API generation process. The representation of the ontology in MoOn is mapped to the Ontology API model (OAM). The OAM is an abstract syntax to model the final, object-oriented representation of the API. It can be mapped directly to a concrete programming language such as Java.

Using the two models MoOn and OAM, application developers can customize the API they like to generate. While there exists a default mapping from an OWL based ontology to an object-oriented API via MoOn and ODM, the application developer may want to add information about the API structure and about the way how the ontology should be used. In addition, he can add additional information about how existing target APIs should be aligned with the ontology. This is useful, e.g., to integrate an existing storage infrastructure API for binary large objects with the ontology API. Thus, the application developer can customize the mapping from the ontology to the API according to his needs.

To illustrate the challenge of generating usable APIs from ontology definitions, we first introduce an example ontology for representing multimedia metadata and its corresponding API and refer to it throughout the paper. Subsequently, we discuss in Section 3 what kind of information is in principle required for generating such ontology APIs and conduct a thorough analysis of the problem. In Section 4, we present possible sources for obtaining this information and how it is used to model the API. We present our model-driven process for ontology APIs in Section 5 and its implementation in Section 6. The current state of the art is reviewed in Section 7, before we conclude the paper.

2 Running Example: Ontology-based Modeling of Multimedia Metadata

We first present an example of an ontology-based modeling of the annotation and decomposition of a multimedia presentation. Secondly, we introduce the structure and functionality of an ideal API for this ontology.

2.1 Decomposition and Annotation of a Multimedia Presentation

The concrete example in Figure 1 shows an excerpt of an ontology modeling multimedia metadata. The multimedia presentation Presentation-1 is decomposed into two images, Image-1 and Image-2. Image-1 is annotated with an EXIF geo-coordinate Geo-Point-1 by the user Paul-1. Figure 1 shows also further individuals such as CompositeConcept-1 classifying the Presentation-1 as composite in the decomposition and ComponentConcept-1 and ComponentConcept-2 that denote Image-1 and Image-2 are components. In addition, there are also some additional ontology classes shown such as Image and Region and their inheritance structure.

The example is based on our Multimedia Metadata Ontology (M3O) [11] for representing annotation, decomposition and provenance information of multimedia data. Although the example seems to be quite small, the ontology presented allows for sophisticated features in terms of annotating and decomposing multimedia content. Not only the Presentation-1 can be decomposed into images, but also the images such as Image-2 can be decomposed into segments. In addition, besides the annotation of a single Image-1, also the image segments as well as the Presentation-1 itself can be annotated. Finally, by attaching information about the creator of an annotation such as Paul-1 for Image-1, the ontology also provides for modeling provenance information.

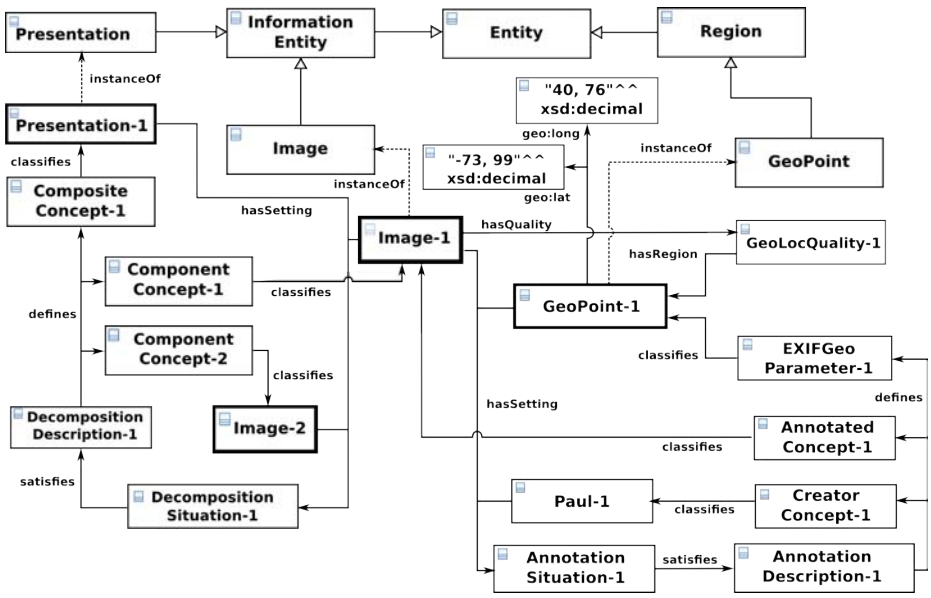

Fig. 1. Instance of an Ontology-based Annotation and Decomposition of a Multimedia Presentation and some Additional Ontology Classes

2.2 Ideal API for the Running Example

An ideal ontology API defines class handles for the central concepts of the ontology and provides functionality for using these classes. For the ontology example above, Figure 2 shows an excerpt of the ideal ontology API. It provides a distinct class for each central concept of the ontology such as Presentation and Image for multimedia objects and GeoPoint for metadata objects. As functionalities, we provide decomposition and annotation of multimedia objects. They are represented by the classes Decomposition and AnnotationSet in the API. The interface InformationEntity is implemented by Presentation and Image. This is required to provide annotation and decomposition of arbitrary multimedia objects as the methods addAnnotation(...) and addDecomposition(...) in the classes AnnotationSet and Decomposition show. The class AnnotationSet allows to attach multiple AnnotationEntities to an InformationEntity. The API interface AnnotationEntity refers to the ontological concept Entity from the example above. The AnnotationEntity interface is implemented in classes such as GeoPoint. As the term *entity* is not very intuitive for application developers using the API, the more conclusive term *annotation entity* is used instead. Finally, when creating an object of the AnnotationSet class, besides the InformationEntity to be annotated also a User parameter is provided. This allows to represent annotations coming from different users, i.e., distinguishing the provenance of annotation information.

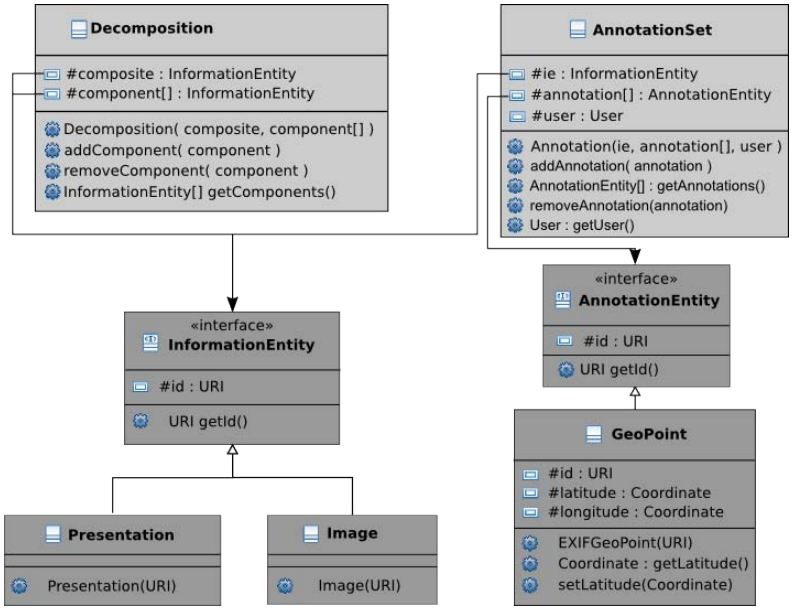

Fig. 2. Model of an Ideal API for the Running Example

3 Information Required for Generating Ontology APIs

The API generation process starts from the ontology definition and ends up in the generation of the API code. To this end, the process has to bridge the fundamental difference between the semantics defined in the ontology (the "what") and the pragmatic handles provided by the API (the "how to use"). The ontology provides a structured representation of the knowledge the application has to deal with. For example, the ontology presented in Section 2.1 can be used by a multimedia management system to represent the annotation and decomposition of multimedia presentations. However, the ontology per se does not provide all information needed to generate an ideal API as depicted in Figure 2. With other words, not all design decisions of the API can be directly motivated from the ontology. In this section, we analyze what kind of information is required and missing in order to generate the ideal API.

(A) Concept Representations An important questions is how to represent the concepts of the ontology in the API. The ideal API in Section 2.2 proposes class representations only for selected concepts from the ontology. This is motivated from the application developer's point of view, who needs classes for the concepts Presentation, Image and EXIFGeoPoint. These concepts and their respective API classes consitute the "content" the multimedia management applications deals with. However, in order to support the full ontological knowledge representation also the other concepts need to be supported in the API. For example, for the decomposition of a Presentation-1 in a concrete application context also the concepts CompositeConcept-1, ComponentConcept-1, ComponentConcept-2, DecompositionDescription-1, and DecompositionSituation-1 are needed. This divides the concepts in an ontology into two disjoint sets of concepts. We call the set of concepts providing content to the API *content concepts* and the set that only serves structural purposes *structure concepts*. The information required to split the concepts into these two sets is not available in ontology definitions. For reason of completeness, the individuals of content concepts and structure concepts are called *content individuals* and *structure individuals*. Table 1 defines the existential conditions for both content individuals and structure individuals. It shows the scope of the individuals and provides examples.

(B) Pragmatic Units An API provides functionalities to the application developer in order to access and use the structured knowledge defined in the ontology. Each functionality involves different concepts of the ontology and the relationships between them. These concepts are either classified as content concepts or structure concepts. Based on this, we can now precisely define a *pragmatic unit* as triple $PU = (CO, SO, R)$ consisting of a set of content concepts CO, a set of structure concepts SO, and a set of relations R between the content concepts and structure concepts. For our running example, we find two pragmatic units for decomposition and annotation as depicted in Figure 3. From the ontology

definition, we need information how many pragmatic units it contains and how they are defined.

Table 1.: **Ontology Individuals Classification**

	Content Individuals	Structure Individuals
Existence	Stand-alone	Only in the context of a pragmatic unit
Scope	Used in multiple pragmatic unit instances	Unique to one pragmatic unit instance
Example	Presentation-1, Image-1, Image-2, and the GeoPoint-1 in the running example	DecompositionSituation-1, GeoLocQuality-1, ComponentConcept-2 and the other concepts in the running example

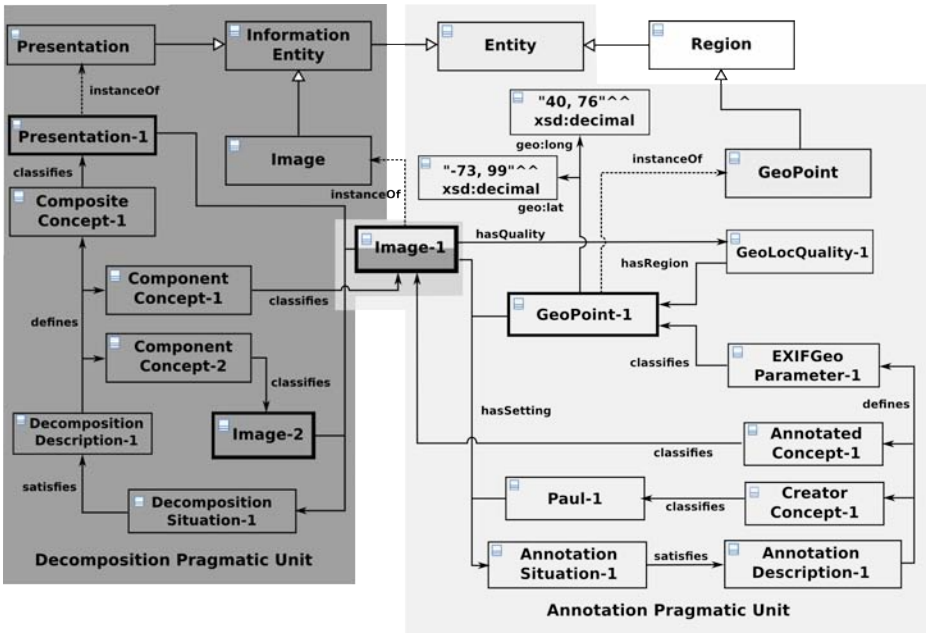

Fig. 3. Pragmatic Units in the Running Example

(C) Method Contracts In our example, the API provides the functionalities for annotating and decomposing multimedia presentations. In a specific programming environment, the application developer may want to ensure that each decomposition has at least two known components. Thus, when calling the methods addComponent(...) and removeComponent(...) of the Decomposition class it needs to ensure that there are at least two components. In our example ontology in Figure 2, there have to be at least two components like Image-1 and

Image-2 of the presentation decomposition. If this constraint is not fulfilled, an ideal API should raise an exception and notify the application about the inconsistency. This behavior of the API is specified in the so-called method contracts. However, the logic world of ontologies with its open world assumption does not require the existence of at least one component individual. A reasoner would not fail because of a missing individual and just assumes that it does not know it.

Thus, in this case the behavior of the API is more restrictive in terms of ensuring consistency of the represented information than the ontology. It is not possible to create a decomposition with less than the minimum number of components using the API whereas this does not cause any inconsistencies for the ontological knowledge representation.

However, it is also possible that the ontology is more restrictive than its API. This is the case when the API cannot ensure consistency and needs to fall back to a reasoner running in the background to decide whether a specific operation is allowed or not. For example, there might be an API for modeling the configuration of a car's equipment. Depending on the different extras such as air conditioning and radio, the car has to be equipped with a different generator. Here, it must be possible to create different configurations using the API, i.e., a specific selection of extras and generator. Only then, the reasoner is applied to determine whether the given configuration is a valid one. Thus, in this case it is possible to create invalid car configurations using the API and determining this invalidity only by using the reasoner.

In a third case, the API might call a reasoner in the background for each operation it performs. Thus, whenever the application using the API calls one of its functionality, the reasoner is used to check the validity of the operation. Based on the result the reasoner computes, the API adapts its behavior and either raises an exception or not. This behavior should also be configurable by the application developer.

These differences in the behavior of the ontology and the ideal API need to be considered and information is required from the ontology to specify the API method contracts. In the remainder of this paper, we consider the first case where the API is more restrictive than the ontology.

(D) Inheritance Structure The concepts defined in an ontology have different relationships. Of particular interest is the inheritance relationship that classifies ontology concepts into sub-concepts and super-concepts. The challenge when generating an API from the ontology definition is that not all of these inheritance relationships and the corresponding concepts need to be mapped to the API.

Looking at the running example of our multimedia metadata ontology in Figure 1, we see that both concepts Image and Presentation inherit from the super-concept InformationEntity. In the ideal API depicted in Figure 2, this is reflected by the classes Image and Presentation implementing the interface InformationEntity. This interface is needed in order to apply the API's annotation functionality and decomposition functionality on any multimedia content that is of type InformationEntity.

However, the running example depicted in Figure 1 also shows the concept Region for which we do not find a counterpart in the API. Thus, for generating an ontology-based API we need information how to abstract a lean and useful inheritance structure from the often complicated inheritance structure of an ontology. We need to know which classes in the inheritance structure are to be represented by the API and which not.

4 Modeling Ontology-based APIs

The information needed to map the ontology definition to the target API must be in principle provided by the API developer. For example, the API developer identifies the pragmatic units of the ontology and defines API functionality in terms of classes and methods for it. He also decides which concepts in the ontology inheritance structure are mapped to abstract classes or interfaces in the API. Our past experience in developing such ideal APIs for ontologies such as the Event-Model-F [12] and X-COSIMO [2] has shown that such a manual creation is very cumbersome and errorprone. Thus, one should obtain as much as possible information automatically from the ontology definition. In this section, we describe the modeling of ideal ontology-based APIs and present possible sources how the required information can be obtained from ontology definition.

4.1 Modeling the Class Representations for the API

According to our problem analysis in Sections 3 (A) and 3 (B), we need to model representations for the ontology concepts and pragmatic units in our API. In contrast to the simple concept-to-class mapping of existing approaches, we model only selected concepts through dedicated API classes as described in Section 3 (A). In addition to this, we model and implement class representations for the pragmatic units provided by the API.

In the past, we have developed ontologies such as the Event-Model-F [12] and X-COSIMO [2]. For these ontologies, we have followed a pattern-based ontology design approach [1]. Ontology design patterns are generic solutions to recurring modeling problems. From the experiences developing such pattern-based ontologies, we have realized that the patterns in the ontologies correlate to the pragmatic units discussed in Section 3 (B). Also our ontology exampled based on the M3O in Section 2.1 follows this pattern-based approach. It provides among others the annotation pattern and the decomposition pattern. Thus, patterns help us in identifying the pragmatic units, i.e., to raise the level of abstraction at which the ontology is formulated [4]. In addition, the patterns help in classifying the ontology concepts as introduced in Section 3 (A) into content concepts and structure concepts. When a concept is referenced in multiple patterns, it is a strong sign that this concept has a content nature, i.e., is central to the domain under consideration.

Corresponding to the decomposition pattern, the ideal API as depicted in Figure 2 provides the class Decomposition. It is associated through the InformationEntity interface to the specializations Presentation and Image. These subclasses

are the content concepts of the pragmatic unit of the decomposition. For the annotation pattern, the ideal API provides the class AnnotationSet, associated through the interface InformationEntity and AnnotationEntity to the specializations Presentation, Image, and GeoPoint. These specializations are the content concepts of the pragmatic units of the annotation. All structure concepts of the pragmatic units for decomposition and annotation are implemented as pairs of attributes in the API classes Decomposition and AnnotationSet (not shown in Figure 2). The pair of class attributes for each structure concepts consists of one URI for the individual and one for the concept.

Based on this, we are able to define pragmatic mapping units for ontology APIs. A pragmatic mapping unit is a quadruple $PMU = (MC, PU, CC, MT)$. It consists of a single mapping class MC that is created for each pragmatic unit PU, a set of content classes CC representing the content concepts of the pragmatic unit, and a set of mapping tuples $MT = \{ < ID_1, Type_1 > , ..., < ID_n, Type_n > \}$ for the structure concepts of the pragmatic unit where $n \geq 0$. A tuple $< ID_i, Type_i > \in MT, 0 \leq i \leq n$ represents the individual URI and the concept URI of the structure concept. Instances of content classes are called content objects. As structure concepts are mapped as pairs of attributes in the mapping class MC, there is not structure concept or structure object on the API side. Table 2 shows the different concepts we have introduced for the ontology side and their counterparts in the API.

Table 2.: **Overview of Mappings between Ontology and API**

Ontology	**API**
Content concepts	Content classes
Content individuals	Content objects
Structure concepts	Class attributes
Structure individual	Individual URI and concept URI
Pragmatic unit	Pragmatic mapping unit

4.2 Modeling the Inheritance Structure of the API

Modeling the concrete class inheritance structure of the API as discussed in Section 3 (D) is dependent on the requirements of the concrete application. In our running example, the classes Image and Presentation implement the interface InformationEntity and the class GeoPoint implements the interface AnnotationEntity. These interfaces refer to the concepts InformationEntity and Entity of the ontology example depicted in Figure 2.1. Please note that the concept Entity has been renamed and implemented in the API by the interface AnnotationEntity to provide the application developer a more intuitive use of the API.

As we can see from the example, the inheritance structure of the ontology is more complex. The M3O bases on DOLCE+DnS Ultralight [6] with several layers of inheritance and further inter-concept relationships. For example, there is an inheritance relation from Entity to InformationEntity. There is also another

concept Region in the inheritance chain from Entity to GeoPoint. However, there are no API representations for these additional inheritance relations and the Region concepts.

Deciding which concepts in the inheritance hierarchy are mapped to interfaces or abstract classes in the API is not trivial. One condition for deciding which concepts in the ontology inheritance hierarchy are mapped to the API is looking at common super-classes of content concepts and the axiomatizations of the ontology to these super-classes. For example, the concept InformationEntity is a common super-concept of the two concepts Image and Presentation. In addition, the M3O provides an axiom saying that the AnnotatedConcept classifies only individuals of concept InformationEntity (not shown in Figure 1). This, together with the fact that the AnnotatedConcept is part of the pragmatic unit for annotations resulted in introducing the interface InformationEntity in the API. Whether to use interfaces or abstract classes is a design decision of the application developer and depends on the concrete usage context of the API.

4.3 Modeling the Class-Methods of the API

The methods provided by an API to work on the ontological knowledge representation can be distinguished into create, read, update, and delete (CRUD) methods . In the following, we investigate how the definition of the different CRUD methods interrelate with the representation structures of the API, i.e., the content classes, structure concept URIs, and pragmatic mapping units.

4.3.1 Create The initial method is the creation of content objects, structure concept representations, and pragmatic mapping units. We consider the following three cases:

C1: Create Content Objects From an application point of view it makes sense to create content objects such as Presentation-1 and Images-1. Create methods for content objects are unproblematic, as they have no side effects on the pragmatic mapping units the content object may be involved in.

C2: Create Structure Concept URIs Taking the distinction of Section 3 (A) into account, only the create method for content objects and pragmatic mapping units make sense. Structure concept URIs are only of relevance in the context of a specific pragmatic mapping unit. Thus they should be created with the corresponding pragmatic mapping unit itself.

C3: Create Pragmatic Mapping Units When creating a pragmatic mapping unit, all mandatory relationships defined between the concepts of the associated pragmatic unit have to be fulfilled. For example, a create method for the decomposition should instantiate all content classes CC representing the content concepts of the pragmatic unit and all mapping tuples MT representing the structure concepts. Thus, the mapping class MC of the pragmatic mapping unit has to instantiate all individual URIs and concept URIs for the structure concepts and

refer to at least three content objects, namely one composite and at least two components. In the case of references to dependent content object, we have to check whether they already exist in the knowledge base, i.e., have been created previously. In general, the create method for pragmatic mapping units has to fulfill all dependencies declared mandatory in the method contract as discussed in Section 3 (C).

4.3.2 Read With read methods, single content objects or pragmatic mapping units are read from the knowledge base. Thus, read methods are quite similar to create methods. The only difference is that the knowledge is obtained from the knowledge base. Like with all API methods, it is necessary to maintain a valid object structure. Especially in the Semantic Web environment, where we often have to deal with incomplete data in the knowledge base, a read method does not necessarily result in a valid API representation. Thus, we have to provide functionalities for recognizing incomplete data when conducting read methods.

R1: Read Content Objects In order to read complete content objects from the knowledge base, the information obtained from a read method has to cover all inner-concept dependencies of this particular content concept. For example, when reading GeoPoint-1 in our running example it has to have a property geo:lat as well as the property geo:long in order to create a complete content object.

R2: Read Structure Concept URIs As with the create method, a particular read method for structure objects is not meaningful. Structure concepts are unique to pragmatic units. Thus, the corresponding individual URI and concept URI are only of relevance in the context of a specific pragmatic mapping unit.

R3: Read Pragmatic Mapping Units For reading pragmatic mapping units, we have to ensure the existence of all mandatory content objects and structure concept URIs. To this end, the knowledge base must contain all information mandatory for instantiation of the pragmatic mapping unit. For example, to be able to create an Annotation object a read method has to provide information about all involved content objects and URIs for representing the structure concepts. Thus, the individual URI and concept URI for the structure individuals AnnotationDescription-1, AnnotationSituation-1, AnnotatedConcept-1, and AnnotationConcept-1 need to be present. For the content objects, we have to check if there already exist objects on API side, otherwise we have to perform a read method for Image-1 and GeoPoint-1 as well.

R4: Searching for Content Objects and Pragmatic Mapping Units Reading from a knowledge base always means querying for a specific content object, set of content objects, or pragmatic mapping units. Thus, an ontology API should provide functionalities for targeted search. For specific content objects, we use the individual URI. To retrieve a set of content objects we can use the class type URI of the objects. Due to the fact that a individual URI of a structure concept is unique to a pragmatic unit, using this URI allows us to query a single

pragmatic mapping unit instance. Another search feature provided might be retrieving all pragmatic mapping units that refer to a specific content object by using the content object's individual URI. Further, possibly application-specific search functionality can be provided by an ontology API.

4.3.3 Delete The delete method can be a major source of problems regarding the consistency of pragmatic mapping units. This is due to the fact that a delete method performed on a single content object can affect multiple other content objects as well as mapping units. As with the create method, we distinguish between delete methods performed on content objects, structure concepts URIs, and pragmatic mapping units.

D1: Delete Content Objects Deleting a content object can affect different pragmatic mapping units and also single concept objects. For example, if the GeoPoint-1 object from the running example was deleted, the pragmatic mapping unit of its annotation would become meaningless. Or if we deleted one of the image objects from our example, the pragmatic mapping unit of the presentation decomposition become invalid as it requires two components.

As mentioned above these delete methods does not violate the consistency of the ontological knowledge due to the open world assumption. However, the API shall provide only meaningful pragmatic mapping units. A delete method on content objects has to recognize the use of the object in pragmatic mapping units and behave accordingly. One option would be to subsequently delete all pragmatic mapping units affected by the deletion of the particular content object, i.e., removing all corresponding pragmatic units from the triple-store. However in other cases, we might want to work with incomplete pragmatic mapping units when deleting a content objects. This is the case, when subsequently further methods on the incomplete pragmatic mapping unit are applied such as creation of two other content objects.

D2: Delete Structure Concept URIs Like the create method, also the delete method makes only sense for content objects and pragmatic mapping units. Thus they should be deleted when the corresponding pragmatic mapping unit is deleted.

D3: Delete Pragmatic Mapping Units When deleting whole pragmatic mapping units, we have to decide how to handle the content objects referred to by this unit. If a concrete content object is still referenced by another pragmatic mapping unit, we leave the content object untouched. But if the deleted pragmatic mapping unit is the last one referencing the content object, we have to decide how to handle the content object. To be able to detect such a case, it is important to count the references from pragmatic mapping units to the content objects. Based on this counter, we can decide whether a content object is deleted along with the pragmatic mapping unit or not. For example, one could decide to never delete a content object, even if the last referring pragmatic mapping unit is deleted. This decision is strongly influenced by the needs of the concrete application and should be configurable by the application developer.

4.3.4 Update Update methods are combined delete methods and create methods. Thus, we delete the old individuals in the knowledge base and replace them by adding new individuals with the current values. We need to provide support for updating content objects and pragmatic mapping units.

5 Model-Driven Generation Process of Ontology APIs

In this section, we describe the concrete Model Driven Engineering (MDE) process of generating APIs from the ontology definition. Figure 4 depicts the multi-stage API generation process and the two intermediate representation models involved. We assume that the input to the process is an ontology defined in OWL. The ontology might be axiomatized using Description Logics (DL). The OWL-based definition of the ontology is represented using the Model for Ontologies (MoOn) as a metamodel. MoOn bases on the Ontology Definition Metamodel (ODM) [7] and extends it by models for some DL constructors. In this model, the ontology concepts are classified into content objects and structure objects as introduced in Section 3 (A). In addition, one can define which parts of the ontology inheritance structure are mapped. In the following transformation this ontology representation is transformed into the Ontology API Model (OAM). The OAM is an object-oriented representation of the resulting API. All special characteristics and properties of the intended API are defined in this model. The OAM comprises all necessary information to generate API code. We are able to use various tools to generate code from the OAM to an arbitrary programming language.

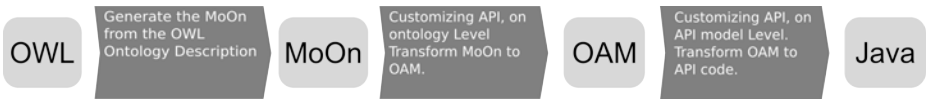

Fig. 4. The API Generation Process

5.1 Model for Ontologies

We use the stereotypes defined in the OWL profile of the ODM to represent OWL constructs in the MoOn. The ODM is a set of lightweight extensions to the UML2 Class Diagram, defined by the Object Management Group (OMG). To separate the pragmatic units from each other, a MoOn is structured as follows

- An *UML:Model* stereotyped with ≪ontology≫ for each ontology.
- An *UML:Package* stereotyped with ≪pragmaticUnit≫ for each pragmatic unit in the ontology.

Some of the models defined in the ODM to represent DL constructors are not UML-conform such as IntersectionOf and Union. Thus, we have modified these models in order to follow the UML-standard and to be able to process them with the tools available today. We will make these models available together with the documentation of our software, when the paper is published.

We have introduced an additional profile with stereotypes to assign the ontology concepts to the sets of content concepts or structure concepts. Only the concepts that have been assigned to one of these sets are mapped to the OAM, i.e., content classes for content concepts and individual URI and concept URI for structure concepts. This gives the application developer a mechanism at hand to control which concepts of the ontology are represented in the API.

5.2 Ontology API Model

The Ontology API Model (OAM) describes the object-oriented structure of the API to generate. With OAM, it is possible to integrate classes from arbitrary existing frameworks or APIs into the ontology API and to add semantic information to these classes. For example, to use the java.awt.Image class of the Java AWT-Framework[2], we integrate our API Image class from our running example by inheriting from java.awt.Image and modifying the classes' constructors.

6 Implementation

We have presented the different stages and models of our model driven engineering process for pattern-based ontology APIs. To demonstrate the feasibility of our process, we have implemented it for generating Java-based APIs. The implementation of our model-driven API generation process uses a set of plug-ins for the Eclipse software development toolkit[3]. For the generation of the API, the first plug-in loads the OWL-based ontology and represents it using MoOn. The loading of the ontology is conducted using the OWL API[4]. The MoOn representation leverages the Model Development Tools (MDT) for UML2[5]. The MDT-UML2 package provides a graphical model representation and manipulation framework for EMF UML2 based models.

Based on the MoOn representation, a second plug-in is applied to transform this representation to OAM. For the OAM, another MDT-UML2-based object structure is created. Subsequently, SPARQL query snippets are attached in the OAM for all concepts in the MoOn that are classified as content objects.

Finally, the last plug-in is responsible to generate Java code out of the OAM representation of the API. For the code generation, we use Java Emitter Templates (JET)[6]. JET is a generic template engine to generate various different textual output such as Java code. The implementation has been applied to generate an API for our running example introduced in Section 2. The API is currently integrated and tested within our SemanticMM4U framework for the generation of multimedia presentations.

[2] `http://java.sun.com/javase/6/docs/technotes/guides/awt/reference.html` last visit June 26, 2010

[3] `http://www.eclipse.org/` last visit June 18, 2010

[4] `http://owlapi.sourceforge.net/` last visit June 10, 2010

[5] `http://www.eclipse.org/modeling/mdt/?project=uml2` last visit June 18, 2010

[6] `http://www.eclipse.org/modeling/m2t/?project=jet` last visit June 10, 2010

7 Related Work

The idea of generating code from an ontology definition is not new. Multiple frameworks like ActiveRDF [8], AliBaba[7], OWL2Java [5], Jena/Jastor[8], OntoJava[9], and others were developed in the past. An overview can be found at Tripresso[10], a project web site on mapping RDF to the object-oriented world.

Most of these frameworks have in common that they use a simple approach transforming each concept of the ontology into a class in a specific programming language, like Java or Ruby. Properties of the ontology concepts are transformed into fields of the declared class. But even the translation of concepts of the ontology into object-oriented software representations is not trivial, as you can see in the discussion made in [5, 3]. None of these frameworks generate ontology APIs on the level of granularity desired by an application developer.

Only Àgogo [9] goes a different way. Àgogo is a programming language independent model driven approach for automatically generating ontology APIs. It introduces an intermediate step based on a Domain Specific Language (DSL). This DSL captures domain concepts necessary to map ontologies to an object-oriented representations. To achieve full support for all these features the single intermediate step of Àgogo for ontology API generation using a DSL is not sufficient.

The approach by Puleston et al. [10] combines so-called direct mappings and indirect mappings of concepts to APIs. In the direct mapping, the ontological entities are statically mapped to a corresponding Java API. In the indirect mapping, Java classes access dynamically the external knowledge representation encoded in OWL. Thus, one part of the knowledge exists statically in the API structure whereas the other part of the model exists and is developed directly in OWL. The latter allows to use background reasoning to make specific concepts of the ontology visible to the API depending on the context of use. This work provides a flexible access to the ontological knowledge representation. However, it does not provide any criteria or support to decide which concepts of the ontology are how to be mapped to the API. Thus, unlike our approach it does not provide criteria based on the structure and semantics of the ontology definition how the API structure should look like and how it should behave.

Our approach bases on the experiences of the API generation frameworks mentioned above. We use many of the mappings proposed in the literature such as [7, 5] and apply a model-driven approach like Àgogo. But to be able to generate specific APIs for multiple application scenarios, the application developer must have full control and customization facilities on all levels of the generation process. To achieve this, our approach suggest to introduce two different intermediate step one for control and customization on ontology level and one for the same tasks on API level.

[7] `http://www.openrdf.org/doc/alibaba/2.0-alpha4/` last visit June 10, 2010
[8] `http://jastor.sourceforge.net/` last visit June 10, 2010
[9] `http://www.aifb.uni-karlsruhe.de/WBS/aeb/ontojava/` last visit June 10, 2010
[10] `http://semanticweb.org/wiki/Tripresso` last visit June 12, 2010

8 Conclusion

We have presented a multi-stage model-driven approach to generate application programming interfaces (APIs) out of logics-based ontologies. In contrast to existing tool support for generating ontology APIs, we leverage the semantics specified in the input ontology such as cardinality, provide explicit support for ontology design patterns, and allow for integrate existing APIs. Our generation process makes use of the structural information in pattern-based ontologies. It generates APIs allowing the application developers to easily integrate such ontologies in the systems and alleviates them from the time-consuming API development task. In our future work, we plan to provide support for integrating and manipulating T-Box knowledge via the generated API.

Acknowledgements This research has been co-funded by the EU in FP7 in the WeKnowIt project (215453).

References

1. Valentina Presutti Aldo Gangemi. *Ontology Design Patterns*. Springer, 2009.
2. Thomas Franz, Steffen Staab, and Richard Arndt. The X-COSIM integration framework for a seamless semantic desktop. In *Knowledge capture*. ACM, 2007.
3. L. Hart and P. Emery. OWL Full and UML 2.0 Compared. `http://uk.builder.com/whitepapers/0and39026692and60093347p-39001028qand00.htm`, 2004.
4. Luigi Iannone, Alan L. Rector, and Robert Stevens. Embedding knowledge patterns into OWL. In *ESWC*, 2009.
5. Aditya Kalyanpur, Daniel Jiménez Pastor, Steve Battle, and Julian A. Padget. Automatic Mapping of OWL Ontologies into Java. In *SEKE*, 2004.
6. C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltramari, and L. Schneider. WonderWeb deliverable D17. the WonderWeb library of foundational ontologies and the DOLCE ontology. Technical report, ISTC-CNR, 2002.
7. OMG. *Ontology Definition Metamodel*. Object Modeling Group, May 2009. `http://www.omg.org/spec/ODM/1.0/PDF`.
8. Eyal Oren, Renaud Delbru, Sebastian Gerke, Armin Haller, and Stefan Decker. Activerdf: object-oriented semantic web programming. In *WWW*. ACM, 2007.
9. Fernando Silva Parreiras, Carsten Saathoff, Tobias Walter, Thomas Franz, and Steffen Staab. 'a gogo: Automatic Generation of Ontology APIs. In *IEEE Int. Conference on Semantic Computing*. IEEE Press, 2009.
10. Colin Puleston, Bijan Parsia, James Cunningham, and Alan L. Rector. Integrating object-oriented and ontological representations: A case study in Java and OWL. In *International Semantic Web Conference*, 2008.
11. Carsten Saathoff and Ansgar Scherp. Unlocking the Semantics of Multimedia Presentations in the Web with the Multimedia Metadata Ontology. In *WWW*. ACM, 2010.
12. Ansgar Scherp, Thomas Franz, Carsten Saathoff, and Steffen Staab. F–a model of events based on the foundational ontology DOLCE+DnS Ultralight. In *K-CAP '09*, New York, NY, USA, 2009. ACM.

This article was processed using the LaTeX macro package with LLNCS style

Bisher erschienen

Arbeitsberichte aus dem Fachbereich Informatik

(http://www.uni-koblenz-landau.de/koblenz/fb4/publications/Reports/arbeitsberichte)

Stefan Scheglmann, Ansgar Scherp, Steffen Staab, Model-driven Generation of APIs for OWL-based Ontologies, Arbeitsberichte aus dem Fachbereich Informatik 7/2010

Daniel Schmeiß, Ansgar Scherp, Steffen Staab, Integrated Mobile Visualization and Interaction of Events and POIs, Arbeitsberichte aus dem Fachbereich Informatik 6/2010

Rüdiger Grimm, Daniel Pähler, E-Mail-Forensik – IP-Adressen und ihre Zuordnung zu Internet-Teilnehmern und ihren Standorten, Arbeitsberichte aus dem Fachbereich Informatik 5/2010

Christoph Ringelstein, Steffen Staab, PAPEL: Syntax and Semantics for Provenance-Aware Policy Definition, Arbeitsberichte aus dem Fachbereich Informatik 4/2010

Nadine Lindermann, Sylvia Valcárcel, Harald F.O. von Kortzfleisch, Ein Stufenmodell für kollaborative offene Innovationsprozesse in Netzwerken kleiner und mittlerer Unternehmen mit Web 2.0, Arbeitsberichte aus dem Fachbereich Informatik 3/2010

Maria Wimmer, Dagmar Lück-Schneider, Uwe Brinkhoff, Erich Schweighofer, Siegfried Kaiser, Andreas Wieber, Fachtagung Verwaltungsinformatik FTVI Fachtagung Rechtsinformatik FTRI 2010, Arbeitsberichte aus dem Fachbereich Informatik 2/2010

Max Braun, Ansgar Scherp, Steffen Staab, Collaborative Creation of Semantic Points of Interest as Linked Data on the Mobile Phone, Arbeitsberichte aus dem Fachbereich Informatik 1/2010

Marc Santos, Einsatz von „Shared In-situ Problem Solving“ Annotationen in kollaborativen Lern- und Arbeitsszenarien, Arbeitsberichte aus dem Fachbereich Informatik 20/2009

Carsten Saathoff, Ansgar Scherp, Unlocking the Semantics of Multimedia Presentations in the Web with the Multimedia Metadata Ontology, Arbeitsberichte aus dem Fachbereich Informatik 19/2009

Christoph Kahle, Mario Schaarschmidt, Harald F.O. von Kortzfleisch, Open Innovation: Kundenintegration am Beispiel von IPTV, Arbeitsberichte aus dem Fachbereich Informatik 18/2009

Dietrich Paulus, Lutz Priese, Peter Decker, Frank Schmitt, Pose-Tracking Forschungsbericht, Arbeitsberichte aus dem Fachbereich Informatik 17/2009

Andreas Fuhr, Tassilo Horn, Andreas Winter, Model-Driven Software Migration Extending SOMA, Arbeitsberichte aus dem Fachbereich Informatik 16/2009

Eckhard Großmann, Sascha Strauß, Tassilo Horn, Volker Riediger, Abbildung von grUML nach XSD soamig, Arbeitsberichte aus dem Fachbereich Informatik 15/2009

Kerstin Falkowski, Jürgen Ebert, The STOR Component System Interim Report, Arbeitsberichte aus dem Fachbereicht Informatik 14/2009

Sebastian Magnus, Markus Maron, An Empirical Study to Evaluate the Location of Advertisement Panels by Using a Mobile Marketing Tool, Arbeitsberichte aus dem Fachbereich Informatik 13/2009

Sebastian Magnus, Markus Maron, Konzept einer Public Key Infrastruktur in iCity, Arbeitsberichte aus dem Fachbereich Informatik 12/2009

Sebastian Magnus, Markus Maron, A Public Key Infrastructure in Ambient Information and Transaction Systems, Arbeitsberichte aus dem Fachbereich Informatik 11/2009

Ammar Mohammed, Ulrich Furbach, Multi-agent systems: Modeling and Virification using Hybrid Automata, Arbeitsberichte aus dem Fachbereich Informatik 10/2009

Andreas Sprotte, Performance Measurement auf der Basis von Kennzahlen aus betrieblichen Anwendungssystemen: Entwurf eines kennzahlengestützten Informationssystems für einen Logistikdienstleister, Arbeitsberichte aus dem Fachbereich Informatik 9/2009

Gwendolin Garbe, Tobias Hausen, Process Commodities: Entwicklung eines Reifegradmodells als Basis für Outsourcingentscheidungen, Arbeitsberichte aus dem Fachbereich Informatik 8/2009

Petra Schubert et. al., Open-Source-Software für das Enterprise Resource Planning, Arbeitsberichte aus dem Fachbereich Informatik 7/2009

Ammar Mohammed, Frieder Stolzenburg, Using Constraint Logic Programming for Modeling and Verifying Hierarchical Hybrid Automata, Arbeitsberichte aus dem Fachbereich Informatik 6/2009

Tobias Kippert, Anastasia Meletiadou, Rüdiger Grimm, Entwurf eines Common Criteria-Schutzprofils für Router zur Abwehr von Online-Überwachung, Arbeitsberichte aus dem Fachbereich Informatik 5/2009

Hannes Schwarz, Jürgen Ebert, Andreas Winter, Graph-based Traceability – A Comprehensive Approach. Arbeitsberichte aus dem Fachbereich Informatik 4/2009

Anastasia Meletiadou, Simone Müller, Rüdiger Grimm, Anforderungsanalyse für Risk-Management-Informationssysteme (RMIS), Arbeitsberichte aus dem Fachbereich Informatik 3/2009

Ansgar Scherp, Thomas Franz, Carsten Saathoff, Steffen Staab, A Model of Events based on a Foundational Ontology, Arbeitsberichte aus dem Fachbereich Informatik 2/2009

Frank Bohdanovicz, Harald Dickel, Christoph Steigner, Avoidance of Routing Loops, Arbeitsberichte aus dem Fachbereich Informatik 1/2009

Stefan Ameling, Stephan Wirth, Dietrich Paulus, Methods for Polyp Detection in Colonoscopy Videos: A Review, Arbeitsberichte aus dem Fachbereich Informatik 14/2008

Tassilo Horn, Jürgen Ebert, Ein Referenzschema für die Sprachen der IEC 61131-3, Arbeitsberichte aus dem Fachbereich Informatik 13/2008

Thomas Franz, Ansgar Scherp, Steffen Staab, Does a Semantic Web Facilitate Your Daily Tasks?, Arbeitsberichte aus dem Fachbereich Informatik 12/2008

Norbert Frick, Künftige Anfordeungen an ERP-Systeme: Deutsche Anbieter im Fokus, Arbeitsberichte aus dem Fachbereicht Informatik 11/2008

Jürgen Ebert, Rüdiger Grimm, Alexander Hug, Lehramtsbezogene Bachelor- und Masterstudiengänge im Fach Informatik an der Universität Koblenz-Landau, Campus Koblenz, Arbeitsberichte aus dem Fachbereich Informatik 10/2008

Mario Schaarschmidt, Harald von Kortzfleisch, Social Networking Platforms as Creativity Fostering Systems: Research Model and Exploratory Study, Arbeitsberichte aus dem Fachbereich Informatik 9/2008

Bernhard Schueler, Sergej Sizov, Steffen Staab, Querying for Meta Knowledge, Arbeitsberichte aus dem Fachbereich Informatik 8/2008

Stefan Stein, Entwicklung einer Architektur für komplexe kontextbezogene Dienste im mobilen Umfeld, Arbeitsberichte aus dem Fachbereich Informatik 7/2008

Matthias Bohnen, Lina Brühl, Sebastian Bzdak, RoboCup 2008 Mixed Reality League Team Description, Arbeitsberichte aus dem Fachbereich Informatik 6/2008

Bernhard Beckert, Reiner Hähnle, Tests and Proofs: Papers Presented at the Second International Conference, TAP 2008, Prato, Italy, April 2008, Arbeitsberichte aus dem Fachbereich Informatik 5/2008

Klaas Dellschaft, Steffen Staab, Unterstützung und Dokumentation kollaborativer Entwurfs- und Entscheidungsprozesse, Arbeitsberichte aus dem Fachbereich Informatik 4/2008

Rüdiger Grimm: IT-Sicherheitsmodelle, Arbeitsberichte aus dem Fachbereich Informatik 3/2008

Rüdiger Grimm, Helge Hundacker, Anastasia Meletiadou: Anwendungsbeispiele für Kryptographie, Arbeitsberichte aus dem Fachbereich Informatik 2/2008

Markus Maron, Kevin Read, Michael Schulze: CAMPUS NEWS – Artificial Intelligence Methods Combined for an Intelligent Information Network, Arbeitsberichte aus dem Fachbereich Informatik 1/2008

Lutz Priese,Frank Schmitt, Patrick Sturm, Haojun Wang: BMBF-Verbundprojekt 3D-RETISEG Abschlussbericht des Labors Bilderkennen der Universität Koblenz-Landau, Arbeitsberichte aus dem Fachbereich Informatik 26/2007

Stephan Philippi, Alexander Pinl: Proceedings 14. Workshop 20.-21. September 2007 Algorithmen und Werkzeuge für Petrinetze, Arbeitsberichte aus dem Fachbereich Informatik 25/2007

Ulrich Furbach, Markus Maron, Kevin Read: CAMPUS NEWS – an Intelligent Bluetooth-based Mobile Information Network, Arbeitsberichte aus dem Fachbereich Informatik 24/2007

Ulrich Furbach, Markus Maron, Kevin Read: CAMPUS NEWS - an Information Network for Pervasive Universities, Arbeitsberichte aus dem Fachbereich Informatik 23/2007

Lutz Priese: Finite Automata on Unranked and Unordered DAGs Extented Version, Arbeitsberichte aus dem Fachbereich Informatik 22/2007

Mario Schaarschmidt, Harald F.O. von Kortzfleisch: Modularität als alternative Technologie- und Innovationsstrategie, Arbeitsberichte aus dem Fachbereich Informatik 21/2007

Kurt Lautenbach, Alexander Pinl: Probability Propagation Nets, Arbeitsberichte aus dem Fachbereich Informatik 20/2007

Rüdiger Grimm, Farid Mehr, Anastasia Meletiadou, Daniel Pähler, Ilka Uerz: SOA-Security, Arbeitsberichte aus dem Fachbereich Informatik 19/2007

Christoph Wernhard: Tableaux Between Proving, Projection and Compilation, Arbeitsberichte aus dem Fachbereich Informatik 18/2007

Ulrich Furbach, Claudia Obermaier: Knowledge Compilation for Description Logics, Arbeitsberichte aus dem Fachbereich Informatik 17/2007

Fernando Silva Parreiras, Steffen Staab, Andreas Winter: TwoUse: Integrating UML Models and OWL Ontologies, Arbeitsberichte aus dem Fachbereich Informatik 16/2007

Rüdiger Grimm, Anastasia Meletiadou: Rollenbasierte Zugriffskontrolle (RBAC) im Gesundheitswesen, Arbeitsberichte aud dem Fachbereich Informatik 15/2007

Ulrich Furbach, Jan Murray, Falk Schmidsberger, Frieder Stolzenburg: Hybrid Multiagent Systems with Timed Synchronization-Specification and Model Checking, Arbeitsberichte aus dem Fachbereich Informatik 14/2007

Björn Pelzer, Christoph Wernhard: System Description:“E-KRHyper“, Arbeitsberichte aus dem Fachbereich Informatik, 13/2007

Ulrich Furbach, Peter Baumgartner, Björn Pelzer: Hyper Tableaux with Equality, Arbeitsberichte aus dem Fachbereich Informatik, 12/2007

Ulrich Furbach, Markus Maron, Kevin Read: Location based Informationsystems, Arbeitsberichte aus dem Fachbereich Informatik, 11/2007

Philipp Schaer, Marco Thum: State-of-the-Art: Interaktion in erweiterten Realitäten, Arbeitsberichte aus dem Fachbereich Informatik, 10/2007

Ulrich Furbach, Claudia Obermaier: Applications of Automated Reasoning, Arbeitsberichte aus dem Fachbereich Informatik, 9/2007

Jürgen Ebert, Kerstin Falkowski: A First Proposal for an Overall Structure of an Enhanced Reality Framework, Arbeitsberichte aus dem Fachbereich Informatik, 8/2007

Lutz Priese, Frank Schmitt, Paul Lemke: Automatische See-Through Kalibrierung, Arbeitsberichte aus dem Fachbereich Informatik, 7/2007

Rüdiger Grimm, Robert Krimmer, Nils Meißner, Kai Reinhard, Melanie Volkamer, Marcel Weinand, Jörg Helbach: Security Requirements for Non-political Internet Voting, Arbeitsberichte aus dem Fachbereich Informatik, 6/2007

Daniel Bildhauer, Volker Riediger, Hannes Schwarz, Sascha Strauß, „grUML – Eine UML-basierte Modellierungssprache für T-Graphen“, Arbeitsberichte aus dem Fachbereich Informatik, 5/2007

Richard Arndt, Steffen Staab, Raphaël Troncy, Lynda Hardman: Adding Formal Semantics to MPEG-7: Designing a Well Founded Multimedia Ontology for the Web, Arbeitsberichte aus dem Fachbereich Informatik, 4/2007

Simon Schenk, Steffen Staab: Networked RDF Graphs, Arbeitsberichte aus dem Fachbereich Informatik, 3/2007

Rüdiger Grimm, Helge Hundacker, Anastasia Meletiadou: Anwendungsbeispiele für Kryptographie, Arbeitsberichte aus dem Fachbereich Informatik, 2/2007

Anastasia Meletiadou, J. Felix Hampe: Begriffsbestimmung und erwartete Trends im IT-Risk-Management, Arbeitsberichte aus dem Fachbereich Informatik, 1/2007

„Gelbe Reihe“
(http://www.uni-koblenz.de/fb4/publikationen/gelbereihe)

Lutz Priese: Some Examples of Semi-rational and Non-semi-rational DAG Languages. Extended Version, Fachberichte Informatik 3-2006

Kurt Lautenbach, Stephan Philippi, and Alexander Pinl: Bayesian Networks and Petri Nets, Fachberichte Informatik 2-2006

Rainer Gimnich and Andreas Winter: Workshop Software-Reengineering und Services, Fachberichte Informatik 1-2006

Kurt Lautenbach and Alexander Pinl: Probability Propagation in Petri Nets, Fachberichte Informatik 16-2005

Rainer Gimnich, Uwe Kaiser, and Andreas Winter: 2. Workshop "Reengineering Prozesse" – Software Migration, Fachberichte Informatik 15-2005

Jan Murray, Frieder Stolzenburg, and Toshiaki Arai: Hybrid State Machines with Timed Synchronization for Multi-Robot System Specification, Fachberichte Informatik 14-2005

Reinhold Letz: FTP 2005 – Fifth International Workshop on First-Order Theorem Proving, Fachberichte Informatik 13-2005

Bernhard Beckert: TABLEAUX 2005 – Position Papers and Tutorial Descriptions, Fachberichte Informatik 12-2005

Dietrich Paulus and Detlev Droege: Mixed-reality as a challenge to image understanding and artificial intelligence, Fachberichte Informatik 11-2005

Jürgen Sauer: 19. Workshop Planen, Scheduling und Konfigurieren / Entwerfen, Fachberichte Informatik 10-2005

Pascal Hitzler, Carsten Lutz, and Gerd Stumme: Foundational Aspects of Ontologies, Fachberichte Informatik 9-2005

Joachim Baumeister and Dietmar Seipel: Knowledge Engineering and Software Engineering, Fachberichte Informatik 8-2005

Benno Stein and Sven Meier zu Eißen: Proceedings of the Second International Workshop on Text-Based Information Retrieval, Fachberichte Informatik 7-2005

Andreas Winter and Jürgen Ebert: Metamodel-driven Service Interoperability, Fachberichte Informatik 6-2005

Joschka Boedecker, Norbert Michael Mayer, Masaki Ogino, Rodrigo da Silva Guerra, Masaaki Kikuchi, and Minoru Asada: Getting closer: How Simulation and Humanoid League can benefit from each other, Fachberichte Informatik 5-2005

Torsten Gipp and Jürgen Ebert: Web Engineering does profit from a Functional Approach, Fachberichte Informatik 4-2005

Oliver Obst, Anita Maas, and Joschka Boedecker: HTN Planning for Flexible Coordination Of Multiagent Team Behavior, Fachberichte Informatik 3-2005

Andreas von Hessling, Thomas Kleemann, and Alex Sinner: Semantic User Profiles and their Applications in a Mobile Environment, Fachberichte Informatik 2-2005

Heni Ben Amor and Achim Rettinger: Intelligent Exploration for Genetic Algorithms – Using Self-Organizing Maps in Evolutionary Computation, Fachberichte Informatik 1-2005