Combining DSLs and Ontologies Using Metamodel Integration

Tobias Walter[1,2] and Jürgen Ebert[1]

[1] Institute for Software Technology, University of Koblenz-Landau
Universitätsstrasse 1, Koblenz 56070, Germany
{ebert,walter}@uni-koblenz.de
[2] ISWeb — Information Systems and Semantic Web,
Institute for Computer Science, University of Koblenz-Landau
Universitätsstrasse 1, Koblenz 56070, Germany

Abstract. This paper reports on a case study where the domain specific language BEDSL for the description of network devices for computer networks is combined with the feature description language FODA used for defining the variability structure of product lines. Furthermore, annotations by fragments of the web ontology language OWL can be added.

In essence, the approach is a three-way integration, which regards two documents written in BEDSL and FODA, respectively, and semantic OWL-annotations as three equally important views of the system under discussion. The integration of languages is done on the level of their metamodels. The standard metamodel of OWL 2 is merged with two self-developed metamodels for the respective domain specific languages.

The merge is loss-free, i.e. the resulting merged model still contains all information from its parts. Thus, the BEDSL part can be used to visualize the network model, the FODA part still defines the feature structure of the corresponding product line and the OWL part can be extracted and fed into an OWL tool to assert the semantic conditions.

1 Introduction

Domain specific languages (DSLs) are used to model and develop systems of particular application domains. Such languages are high-level and provide abstractions and notations for better understanding and easier modeling using concepts and notations that are familiar to domain experts. Often a variety of different domain specific languages is used simultaneously to describe the system from several viewpoints.

In the context of *Model Driven Software Engineering* (MDSE) DSL models may be combined with other models in standardized languages to form a complete and consistent overall model of the system under development. Thus, they have to be viewed as one integrated overall model and simultaneously the view-specific (sub-)models shall not be destroyed.

For reasoning about all kinds of models description logics based *ontology languages* get more and more accepted as a means for describing concept structures

W.M. Taha (Ed.): DSL 2009, LNCS 5658, pp. 148–169, 2009.

and constraints. Ontology languages like OWL allow a precise description of the underlying concepts and provide a good basis for reasoning over models in the described application domain.

This paper reports on a case study where a domain specific language for the description of network devices (BEDSL, [1]) for computer networks is combined with a feature description language used for defining the variability structure of product lines (FODA, [2]). Those two languages are integrated into one common description and extended by fragments of the web ontology language OWL 2 [3] to annotate semantics and semantical constraints.

This approach is a three-way integration, which regards two documents written in BEDSL and FODA, respectively, and the semantic OWL-annotations as three equally important views of the system under discussion. The integration of languages is done on the level of their metamodels. The standard metamodel of OWL 2 [3] is merged with two self-developed metamodels for the respective domain specific languages.

The merge is loss-free, i.e. the resulting combined model still contains all information from its parts. Thus, the BEDSL part can be used to visualize the network configuration, the FODA part still defines the feature structure of the corresponding product line and the OWL part can be extracted and fed into an OWL tool to assert the semantic conditions.

The structure of this paper is as follows. Section 2 shortly introduces the case study and gives an example of an integrated BEDSL-FODA-OWL description as it might be produced by the domain engineers. Section 3 discusses the three languages separately, gives a short sketch of their use and presents some relevant parts of their respective metamodels. In section 3 the three single metamodels are integrated into one (using some elementary transformation steps) in such a way that on the one hand no information is lost and on the other hand all intermodel relations are respected and expressed explicitly. Section 5 demonstrates the integration using the overall example and shows e. g. the OWL-part of the model can be extracted in its full form from the integrated model. Section 6 discusses related work, and section 7 summarizes the paper and gives an outlook.

2 Case Study

In this section we will present a case study which is provided by *Comarch*[1], one of the industrial partner in the *MOST project*[2].

Comarch, a Polish software house, is specialized in designing, implementing and integrating IT solutions and services. Specifically, Comarch provides an *Operations Support Systems* (OSS) [4] that generally refers to systems that perform management, inventory, engineering, planning, and repair functions for communications service providers and their networks.

For software development Comarch uses model-driven methods, where different kinds of domain specific languages (DSL) are deployed during the modeling process.

[1] `http://www.comarch.com/`

[2] `http://www.most-project.eu/`

In this case study we want to consider two domain specific languages used by Comarch. On the one hand for representing the business entities from its problem domain, namely network devices, Comarch uses its own self-developed language called BEDSL (Business Entity Domain Specific Language). The language is very simple such that modeling is easy and productivity and quality of the models are obtained. On the other hand the feature description language FODA is considered to define the variability of their product lines. Since Comarch complains that these domain specific languages are not expressive enough, the web ontology language OWL comes into play to define additional semantic conditions for both languages, BEDSL and FODA.

To fullfil this need the idea is to integrate ontologies within the two domain specific languages in the following way. A domain modeler should use BEDSL and FODA as much as he can since he is familiar with these languages. If he realizes that the DSL he is using is not expressive enough or if he wants to define additional semantic conditions he should be able to annotate the model elements with very simple *OWL text*. We use the term OWL text because the domain modeler should not annotate the model elements with complete ontologies but only with relevant parts. On the one side he might not be familiar with the Web Ontology Language (OWL) and developing ontologies, on the other side redundant information that is already defined by the static structure of the domain models should not be stated again in the ontology to avoid redundancy.

Figure 1 depicts two domain models. On the left side BEDSL is used to define the structure of network devices. Here each network device has some ports and each port has a state which either can be free or reserved. On the right side FODA is used to define features of a product line. In the example the diagram states that if the feature *Network Management Operations* is selected exactly one of the subfeatures *Show Ports* and *Allocate Ports* has to be selected, too.

Besides the static structure of the two domain models the Comarch designer wants to define that every `Networkdevice` entity is available for the feature *Show Ports* but not for the feature *Allocate Port*, because only free ports can be allocated. But a network device which has some ports whose state is `FreePort` is available for the *Allocate Port* feature. Furthermore the domain modeler wants to prescribe that each network device has exactly 8 ports and a port has exactly 1 state.

Because the domain specific language is not expressive, much less than for example UML class diagrams (roles, cardinalities etc. are missing), some model elements are annotated with OWL text and global conditions are stated, using the Manchester Syntax style [5] in this case. In the example we first have the constraint `hasPort exactly 8 Port` (1). Thus the designer states that `Networkdevice` using the `hasPort`-association is exactly connected with 8 Ports. Furthermore he defines that each port has exactly one state using the following statements: `hasState exactly 1 State` (2).

The second part of constraints contains the following statements: `AvailableFor value ShowPorts` (3). Here the designer states that the entity `Networkdevice` is available for the *Show Ports* feature.

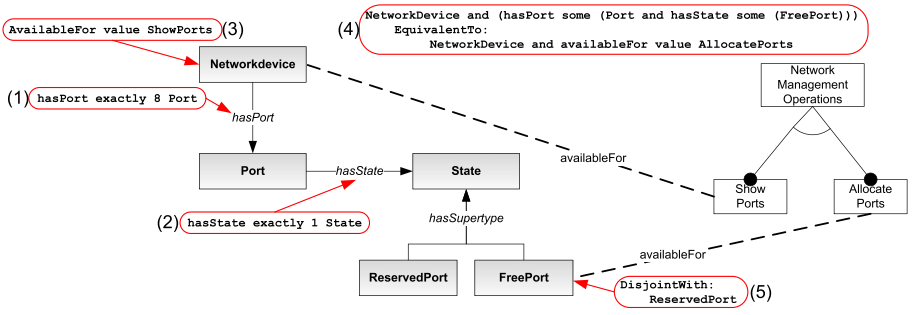


Fig. 1. Domain model with annotated model elements

At last we want to define that a network device which has some port with state `FreePort` is available for the *Allocate Port* feature. Therefore the domain modeler can define the following global constraint: `NetworkDevice and (hasPort some (Port and hasState some (FreePort))) EquivalentTo: NetworkDevice and availableFor value AllocatePorts` (4). Using Manchester syntax style for all annotations he states that an entity of network device is available for the *Allocate Ports* feature, if and only if the network device has a port which has the state `FreePort`.

Using the statements `DisjointWith: ReservedPort` (5) the domain modeler defines that the semantic extension of `ReservedPort` and `FreePort` is disjoint.

Having annotated domain models the domain modeler should be able to check the constraints defined by the OWL text. Here the idea is to project the domain models and its constraints to a complete ontology for querying and reasoning.

3 Domain Specific Modeling and Ontologies

The approach of integrating domain specific languages and ontology languages presented here occurs at the metamodel layer and thus bases on different metamodels for DSLs and ontologies.

This work is based on the *four-layer metamodel hierarchy* as a basic requirement for formally defining domain specific languages. Having the idea of metamodeling we present a simple DSL that is used by our partner Comarch to describe their business entities. The idea of this paper is to integrate ontologies into languages itself. Instead of a precise specification of the metamodels of the web ontology language, which is already done by the OMG [6] or by W3C [3], we want to give the idea of how developing ontologies in a model-driven way.

3.1 Metamodeling

Like other modeling languages DSLs must be defined formally to be supported by tools. Otherwise it would not be possible to generate code from them [7]. Here *metamodeling* is used as well known way to describe languages [8].

An important framework for defining languages is the OMG four-layer modeling architecture. In such a modeling architecture the *M0-layer* represents the real world objects. Models are defined at the *M1-layer*, a simplification and abstraction of the M0-layer. Models at the M1-layer are defined using concepts which are described by metamodels at the *M2-layer*. Each metamodel at the M2-layer determines how expressive its models can be. Analogously metamodels are defined by using concepts described as meta-metamodels at the *M3-layer*. In the OMG model driven architecture approach we have MOF at the M3-layer which is defined by itself [9]. So in general each model is defined by its corresponding metamodel as an instantiation of classes or concepts belonging to the corresponding metamodel.

3.2 Metamodel of the Business Entity Domain Specific Language

As mentioned above Comarch uses a domain specific language to describe business entities and the relation between them. Figure 1 depicts on the left side an example of using the domain specific language BEDSL in concrete syntax. Here the domain model defines the relation between the entities `Networkdevice`, `Port`, `State`, `FreePort` and `ReservedPort`.

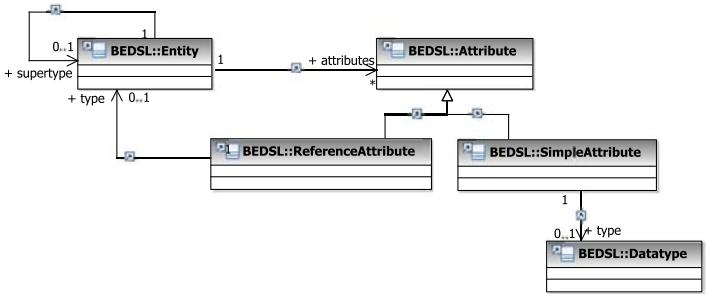


Fig. 2. Metamodel of the Business Entity Domain Specific Language (BEDSL)

Figure 2 depicts the metamodel of the *Business Entity Domain Specific Language* (BEDSL). To describe concepts we first have two classes `Entity` and `Attribute`. Each `Entity` can be specialized and can be assigned with a number of attributes. The metamodel provides two types of attributes for entities: `SimpleAttribute` and `ReferenceAttribute`. A `SimpleAttribute` is used for data values which have to correspond to a data type. Via a `ReferenceAttribute` an entity can point to some other entity.

Figure 3 depicts an instance of the metamodel of the domain specific language at the M1-layer which describes the example from figure 1 in abstract syntax. Here a `NetworkDevice` of type `Entity` has a `ReferenceAttribute` a `Port` of type `Entity`. The `Port` also has a `PortState` which has two specializations `ReservedPort` and `FreePort`.

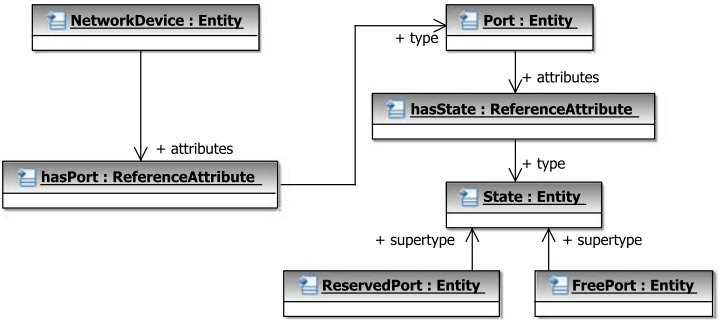


Fig. 3. Example of using the Domain Specific Language

3.3 Metamodel of the Web Ontology Language

In general ontologies are used to define sets of concepts that are used to describe domain knowledge and allow specifying classes by rich, precise logical definitions [10]. Advantages of ontologies, e.g. represented by languages such as OWL or RDFS, are the support of other (domain specific) languages by enabling validation or automated consistency checking. Furthermore ontology languages provide a better support for reasoning than MOF-based languages [11].

Recent works exist that compare the *ontological technology space* (OTS) and the *metamodeling technology space* (MMTS) based on an UML environment [12] [13]. A short summary of the comparison between UML modeling and ontological modeling is provided by figure 4.

To get a feeling for the OWL 2 metamodel and for some requirements for this paper we present two constructs of the metamodel.

The first part of the metamodel depicted in figure 5 presents the *Object Property Axioms* `ObjectPropertyDomain` and `ObjectPropertyRange`. The `ObjectPropertyDomain` axiom is used to connect one OWL class, described by `ClassExpression`, with the domain of one OWL object

OTS	MMTS
ontology	package
class	class, classifier
individual, value	instance
property	association, attribute
subclass, subproperty	subclass, generalization
data types	data types
enumeration	enumeration
domain, range	navigable, non-navigable
cardinality	multiplicity

Fig. 4. Comparison of ontology technology space and metamodeling technology space [13]

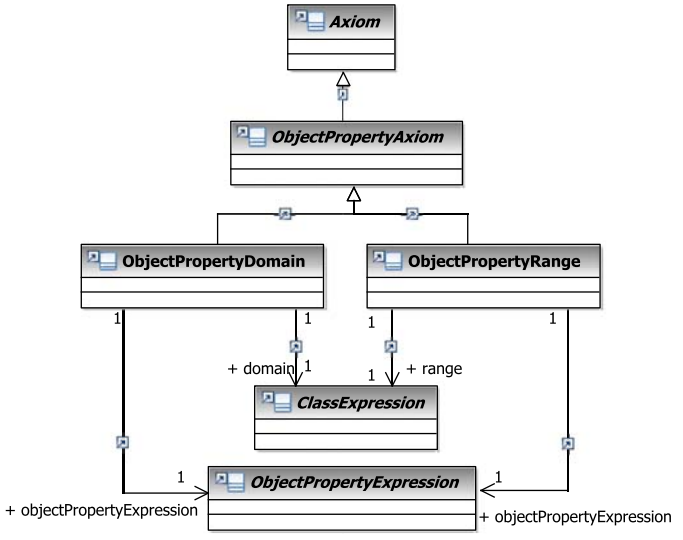


Fig. 5. Object Property Axioms

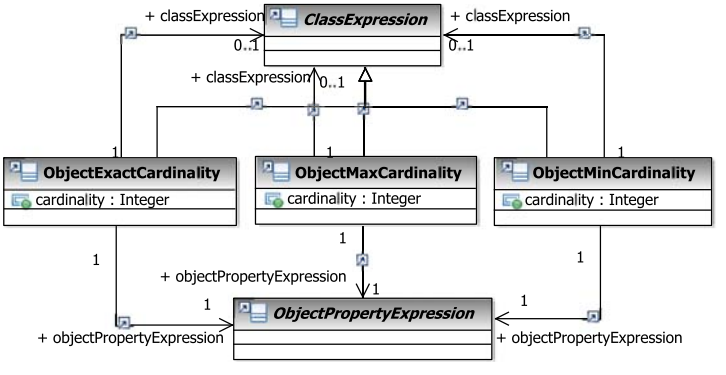


Fig. 6. Object Property Cardinality Restrictions

property, described by the `ObjectPropertyExpression`. Vice versa, the `ObjectPropertyRange` axiom is used to connect one OWL class, described by `ClassExpression`, with the range of one OWL object property, described by the `ObjectPropertyExpression`.

Classes in OWL 2 can be formed by placing restrictions on the cardinality of object property expressions. Figure 6 depicts the *object property cardinality restrictions* provided by OWL 2. The specialization of class expressions `ObjectMinCardinality`, `ObjectMaxCardinality`, and `ObjectExactCardinality` contain those individuals that are connected by an object property expression to at least, at most, or exactly a given number of instances of a specified class expression [3].

A complete and detailed structural specification and the functional-style syntax of the OWL 2 Web Ontology Language is presented in [3].

Figure 7 depicts an example of using the metamodel of OWL 2. Here two instances of `Class` are used to define the OWL classes `Networkdevice` and `Port`. An instance of `ObjectProperty` called `hasPort` in addition with the instances of `ObjectPropertyDomain` and `ObjectPropertyRange` is used to connect the two classes. At last the `ObjectExactCardinality` restriction is adopted on the object property. Figure 8 depicts the ontology in concrete syntax. In fact the ontology defines that a network device has exactly 8 ports.

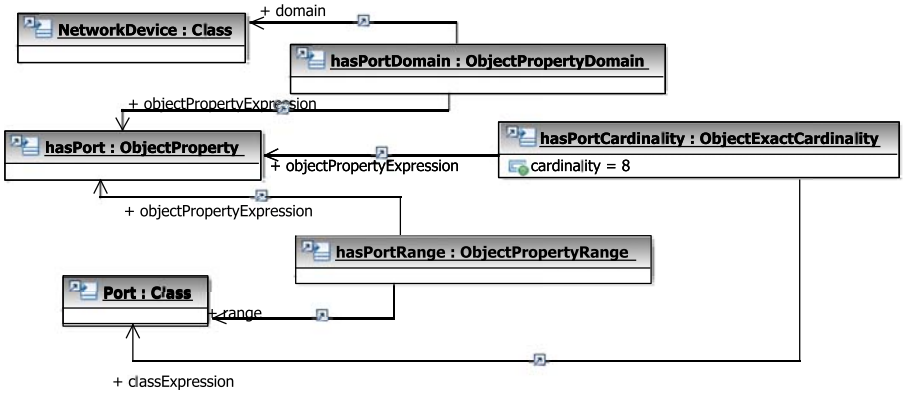


Fig. 7. Example of using the OWL 2 metamodel

```
Class: owl:Thing

Class: NetworkDevice
    SubClassOf:
        owl:Thing

Class: Port
    SubClassOf:
        owl:Thing

ObjectProperty: hasPort
    Domain:
        NetworkDevice
    Range:
        hasPort exactly 8 Port
```

Fig. 8. Ontology in concrete syntax

3.4 Metamodel of a Feature Oriented Domain Analysis

In the first part of this section we will give a short language specification of another domain specific language, called FODA (*Feature Oriented Domain Analysis*) [2]. In the following we depict the metamodel of the language.

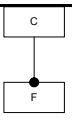
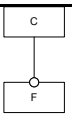
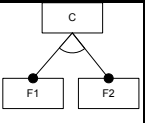
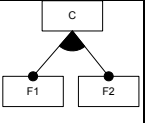
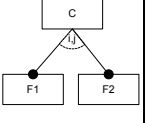
Type	Notation
Mandatory - If feature C is selected, feature F must be selected too	
Optional - If feature C is selected, feature F may or may not be selected	
Alternative - If feature C is selected, exactly one of the child features *F1*, *F2* has to be selected	
Or - If feature C is selected, one or more of the child features *F1*, *F2* can be selected	
Selection - If feature C is selected, any number between i and j of its child features must be selected too	

Fig. 9. Types of feature relationships in a feature diagram

FODA appeals to many product line developers because features are essential abstractions that both customers and developers understand [14].

Thus the main concept in the feature description language FODA is the *feature* itself. Here a feature is a distinguishable characteristic of a concept that is relevant to some stakeholders [15]. To describe features and relationships between them a feature model is used which is represented by a *feature tree*. The nodes of this tree depict the features themselves. Edges of different types are used to specify relationships between these features.

Figure 9 provides an overview of all types of *feature relationships* which connect two different features. Every feature relationship has a unique type and assigns a child feature to its parent feature. For graphical notation we use the one from [2].

Beside feature relationships some features also can depend on some other:

- *Implication* - An implication between two features *F1* and *F2* defines that if feature *F1* is selected *F2* also must be selected.
- *Exclusion* - An exclusion between two features *F1* and *F2* defines that if feature *F1* is selected the feature *F2* must not be selected. Vice versa, if feature *F2* is selected feature *F1* must not be selected.
- *Influence* - An influence between two features *F1* and *F2* defines that if feature *F1* is selected then one property of *F2* is constrained by one value if *F2* is selected too.

Figure 10 depicts a simplified metamodel of FODA with regard to the language description of FODA given here.

Features, represented by the class `Feature` in the metamodel, are on the one hand connected to other features via `FeatureRelationship`-associations, and on the other hand they may be connected via `Dependency`-associations. For every type of feature relationship, there exists a subclass of `FeatureRelationship` in the metamodel. For every kind of dependencies between two features, there also exists one subclass of `Dependency`.

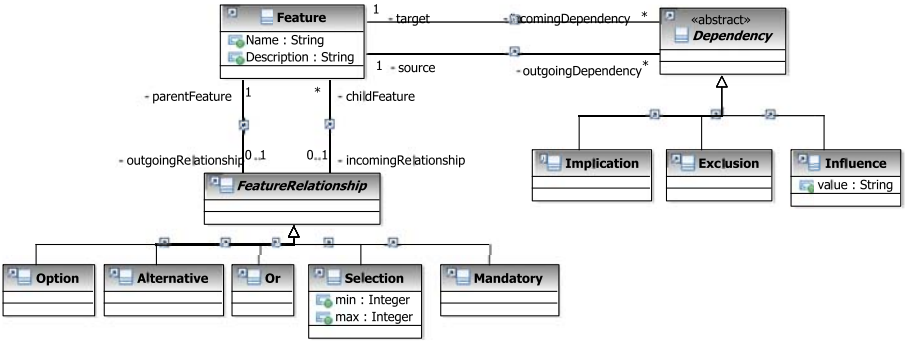


Fig. 10. Metamodel of FODA

4 Integration of Domain Specific Languages and OWL

In this section we describe the integration of the two domain specific languages BEDSL and FODA with the ontology language OWL 2.

The general idea is to have an integrated modeling language that allows us to define domain models and constraints for model elements simultaneously. For instance we want to apply OWL class axioms on entities of the domain model. Furthermore different constraints might be adopted on the relation between entities, features and attributes.

After integrating the three languages BEDSL, OWL and FODA at metamodel-level (M2) we are able to create models that consist on the one side

of domain models but on the other side also of OWL constructs to constrain the domain model. In concrete syntax (cf. figure 1) we would be able to create our domain models and annotate model elements with OWL text in Manchester Syntax style. In our case OWL text means that only necessary parts of the ontology are defined, because redundant information that is already defined by the domain model should not be stated again in OWL to avoid redundancy.

We describe the integration of the domain specific language BEDSL and the ontology language OWL 2 using eight steps. In each step one transformation is presented which describes where the two metamodels are connected or how the metamodels have to be extended for a seamless integration.

After the integration of BEDSL and OWL we present the integration of FODA with the present integrated metamodel which consists of BEDSL and OWL elements.

Finally we depict some parts of the integrated metamodel. Section 5 gives an example of using it.

4.1 Integration of BEDSL and OWL

In this section we want to present eight steps where transformations are applied to integrate the ontology language OWL and the Comarch domain specific language BEDSL. These integration steps are described semi-formally to focus on the application of the approach and not to overload the paper by too much formal text.

Beside of OWL class axioms that should be adopted on entities of BEDSL we want to restrict the relation between entities and attributes. Here we distinguish two cases: constraints and restrictions that are adopted on reference attributes and those that are adopted on simple attributes.

Constraints and restrictions on reference attributes are those that are adopted in OWL on object properties. So we have to materialize an association between `Entity` and `ReferenceAttribute` by a separate class that accepts OWL object property restrictions.

On the other side we have to materialize an association between `Entity` and `SimpleAttribute` by a separate class to adopt data property restriction on this relation.

Step 1: Identify OWL Class and BEDSL Entity. The first step of integration of the metamodels of the BEDSL and OWL occurs using the transformation depicted in figure 11. Here we express the fact that the concepts of an `Entity` class of the BEDSL metamodel and an OWL `Class` of the OWL 2 metamodel are equivalent. Using this transformation the classes are merged to one single class called `IntegratedClass`. Then on the one side we are able to associate attributes of the DSL with the new class `IntegratedClass` and on the other side it is allowed to adopt OWL constructs like class axioms on it.

Step 2: Connect Integrated Class with Reference Attribute. Next after having the new class `IntegratedClass` we connect it with the class

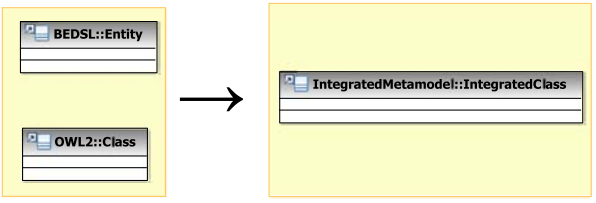


Fig. 11. Step 1: Identify OWL Class and BEDSL Entity

`ReferenceAttribute` creating a new association between them. The reason is to have a relation between the two classes which could be constrained by OWL object property restrictions.

Figure 12 depicts the transformation. Here a *1-to-n* association is constructed between `IntegratedClass` and `ReferenceAttribute`, thus one instance of `IntegratedClass` can be connected with many instances of type `ReferenceAttribute`.

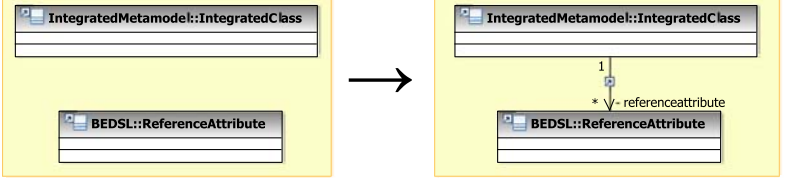


Fig. 12. Step 2: Connect Integrated Class with Reference Attribute

Step 3: Transform Association to Class ReferenceProperty. In OWL restrictions can be applied on object properties. For this purpose the association between class `IntegratedClass` and class `ReferenceAttribute` has to be transformed to a separate class leveraging this association to a class called `ReferenceProperty`.

Figure 13 depicts the transformation that supports the creation of the leveraging class `ReferenceProperty`. The source model of this transformation consists of classes `IntegratedClass` and `ReferenceAttribute` which are connected via an association. In place of the association in the source model there is a new class `ReferenceProperty` which is an element of the integrated metamodel. This class is incident with two associations which link to the classes `IntegratedClass` and `ReferenceAttribute`. Because the role names `domain` and `range` are used, the object property can later be reconstructed to get a consistent ontology for querying and reasoning.

Step 4: Connect OWL Object Property and Reference Property. Step 3 of the integration extended the association between `IntegratedClass` and `ReferenceAttribute` by a new class `ReferenceProperty`. So far the new class was not yet integrated with the OWL `ObjectProperty` class of the OWL 2

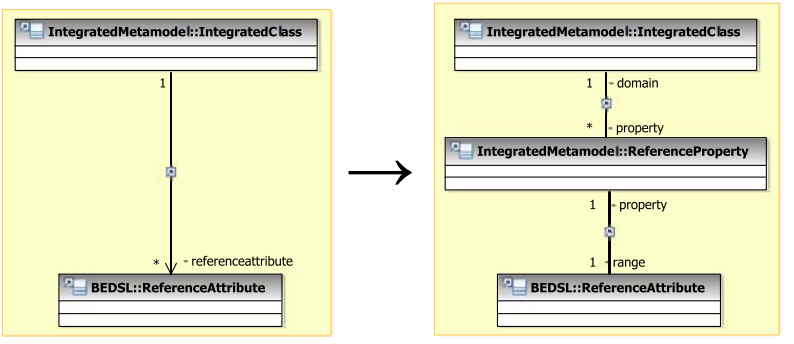


Fig. 13. Step 3: Transform Association to Class ReferenceProperty

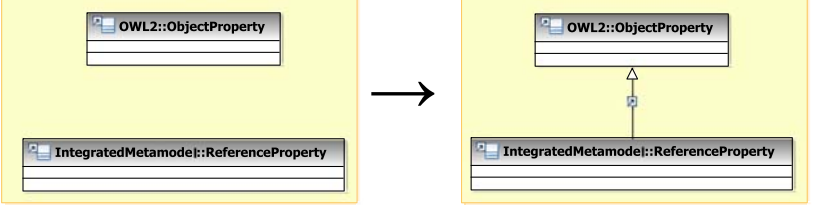


Fig. 14. Step 4: Connect OWL Object Property and Reference Property

metamodel. So in step 4 we create a specialization relationship between the OWL object property class and the newly created `ReferenceProperty` class.

Figure 14 depicts the transformation. As a result now restrictions on object properties can be applied to the relation between `IntegratedClass` and `ReferenceAttribute`.

Step 5: Connect IntegratedClass with Simple Attribute. While in step 2 to 4 the integration of OWL object properties was described, we now consider the integration of OWL data properties with regard to simple attributes of the metamodel of the BEDSL. So analogously to step 2 we connect the `IntegratedClass` with the `SimpleAttribute` class by a new association.

Figure 15 depicts this transformation. After the transformation we have a separate relation between `IntegratedClass` and `SimpleAttribute` which can be constrained by OWL data property restrictions.

Step 6: Transform Association to Class SimpleProperty. To adopt restrictions for OWL data properties on the association between `IntegratedClass` and `SimpleAttribute`, the association is materialized to a separate class called `SimpleProperty`.

Figure 16 depicts the transformation. In the target model of the transformation there now exists the new class `SimpleProperty`, which is connected with `IntegratedClass` and `SimpleAttribute`. Later domain and range of the data property could be reconstructed because of the role names at the association ends.

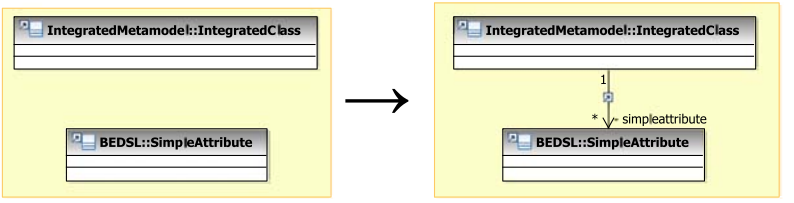


Fig. 15. Step 5: Connect IntegratedClass with Simple Attribute

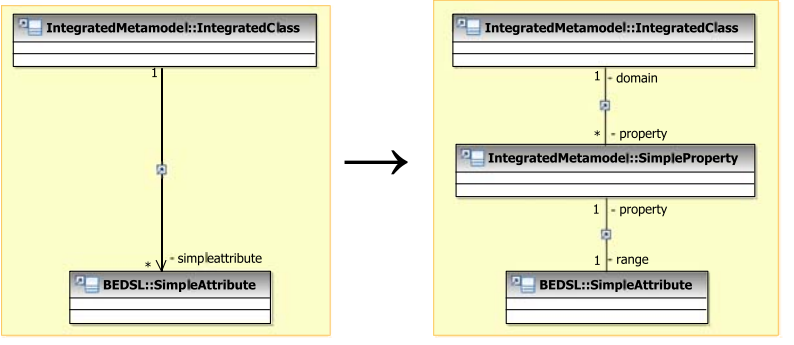


Fig. 16. Step 6: Transform Association to Class SimpleProperty

Step 7: Connect OWL Data Property and Simple Property. Step 6 of the integration extended the association between `IntegratedClass` and `SimpleAttribute` by a new class `SimpleProperty`. So far the new class was not integrated to the OWL `DataProperty` of the OWL 2 metamodel. So in step 7 we create a specialization relationship between the OWL data property class and the newly created `SimpleProperty` class.

Figure 17 depicts the concrete transformation. As a result restrictions on data properties could be applied to the relation between `IntegratedClass` and `SimpleAttribute`.

Step 8: Merge DSL Datatype and OWL Datatype. The last step of integration merges the classes of `Datatype` in the BEDSL metamodel and in the OWL2 metamodel, because both classes represent the same concept of a datatype.

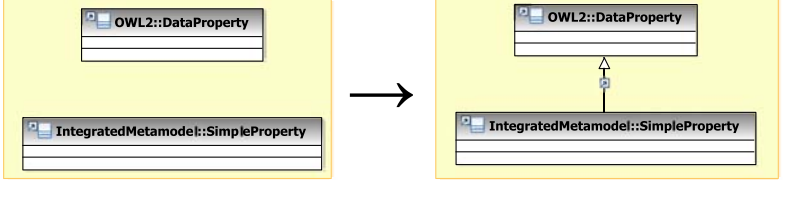


Fig. 17. Step 7: Connect OWL Data Property and Simple Property

Figure 18 depicts the concrete transformation

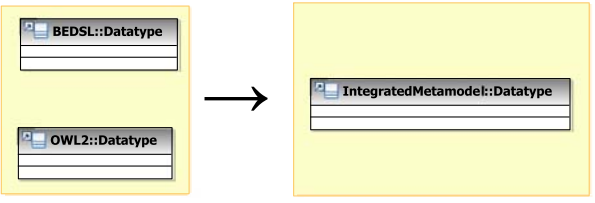


Fig. 18. Step 8: Merge DSL Datatype and OWL Datatype

4.2 Integration of FODA and the Present Integrated Metamodel

After the integration of the ontology language OWL 2 and the domain specific language BEDSL we now want to show how features and a feature model can be integrated with the present enriched domain model.

On the one side we have the domain model and its corresponding metamodel which prescribe the domain specific language enriched by ontologies. On the other side we have the feature model, which is restricted by its corresponding metamodel (cf. figure 10). Now we want to integrate these two languages at the M2-layer.

To connect the FODA metamodel and the present integrated metamodel we need only one step. A feature model describes how feature instances are related to each other. Hence we can assume that a feature is a named individual in the OWL 2 metamodel.

Figure 19 depicts the transformation that creates the inheritance relation between class `Feature` and class `NamedIndividual` and thus is used to connect the two metamodels.

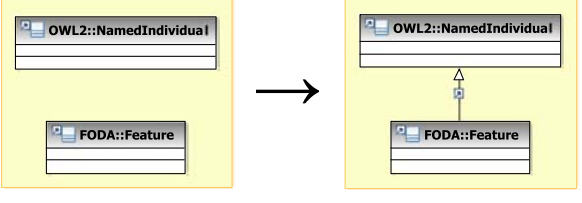


Fig. 19. Connect FODA Feature with OWL Individual

4.3 Integrated Metamodel

After having integrated the three languages BEDSL, OWL 2 and FODA we now present parts of the integrated metamodel as the result of all steps.

On the one side the metamodel provides all classes of the OWL 2 metamodel, on the other side all classes of the BEDSL and FODA metamodel can be used.

The intersection of the BEDSL and the OWL 2 metamodel is built by the classes `IntegratedClass`, `ReferenceProperty`, `SimpleProperty` and `Datatype`.

`IntegratedClass` on the one side can be linked to attributes of the domain specific language but on the other side can also adopt different OWL constructs like restrictions or class axioms which are in general allowed for the OWL class `ClassExpression`.

The class `ReferenceProperty` is a specialization of the OWL Class `ObjectProperty` and is associated with the DSL class `ReferenceAttribute`. Thus OWL object property restrictions can be adopted on the relation between `IntegratedClass` and `ReferenceAttribute`.

Besides OWL object property restrictions we consider the integration of OWL data property restrictions. Here we created the class `SimpleProperty` that represents the relation between `IntegratedClass` and `SimpleAttribute`. Using a specialization relationship between the OWL class `DataProperty` and `SimpleProperty` we can adopt OWL data property restrictions on the relation between `IntegratedClass` and `SimpleAttribute`.

The integration of FODA is realized by creating a specialization relationship between the classes `Feature` and `NamedIndividual`.

Figure 20 depicts in detail the relevant parts of the integrated metamodel that connect the OWL 2 and the BEDSL metamodel.

Considering the case study in section 2, we now are able to describe all relevant parts of Comarchs *Operations Support System* (OSS), just using the integrated

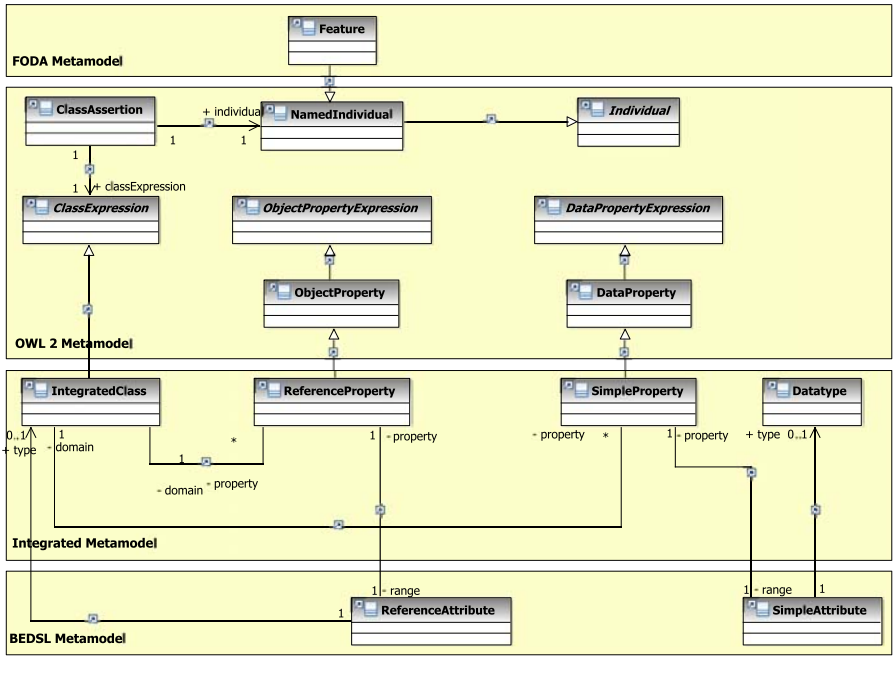


Fig. 20. Relevant parts of the integrated metamodel

metamodel. The BEDSL parts of the OSS are instances of elements of the `BEDSL` package in the integrated metamodel. The product line and configurations of the OSS are described by using the `FODA` package of the integrated metamodel. In addition, we are now able to define any semantics of the OSS and dependencies between the different languages using the `OWL2` package of the integrated metamodel.

5 Example

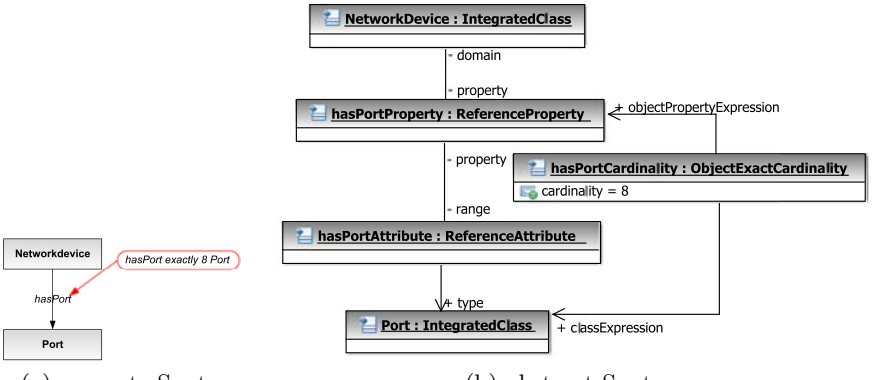
In the following we give an example of using the integrated metamodel and show how we can project the enriched domain models to an ontology for querying and reasoning. We exemplify this using a small part of the model in figure 1.

Figure 21(a) depicts a part of the domain model in concrete syntax. Here we have the two concepts `NetworkDevice` and `Port`, which are connected via an association. This association is annotated with some *OWL text*, in our case we define `hasPort exactly 8 Port`. Thus we want to state that each network device has exactly eight ports.

Figure 21(b) depicts the domain model and its constraint in abstract syntax which is correspondent to the integrated metamodel. Here we have two instances of `IntegratedClass` called `NetworkDevice` and `Port`. These instances are connected via a `ReferenceProperty` and a `ReferenceAttribute`. The `ReferenceProperty` is used to adopt the OWL restriction, the `ReferenceAttribute` is used to point to the `Entity` instance.

Since we have the domain model and its constraints in abstract syntax the next step would be to project the domain model to a complete ontology for reasoning and querying and of course later for constraint checking. Figure 22 depicts which parts of the domain model are considered for constructing the ontology.

The two instances of `IntegratedClass` yield two separate OWL `Classes`. Thus in our case we have the OWL classes `NetworkDevice` and `Port`.



(a) concrete Syntax (b) abstract Syntax

Fig. 21. Domain Model with constraint

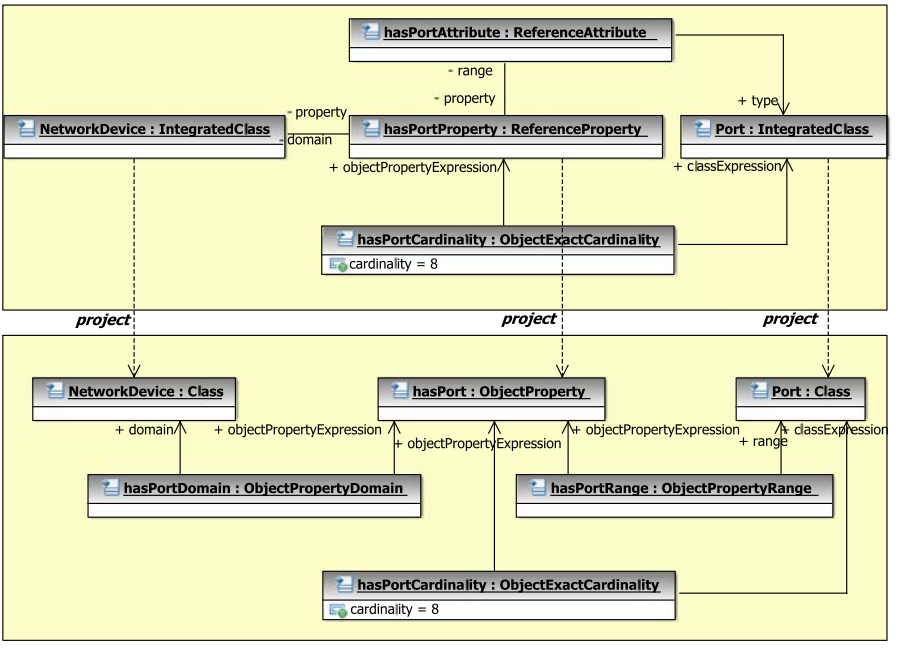


Fig. 22. Projection of the Domain Model to an Ontology

The object property is constructed with regard to the instance of `ReferenceProperty` because in the integrated metamodel `ReferenceProperty` inherits from `ObjectProperty` of the OWL metamodel. The new instances of `ObjectPropertyDomain` and `ObjectPropertyRange` are derived from the unique role names of the links between the instances `NetworkDevice` and `Port`. At last the *exact cardinality restriction* is added to the ontology.

This procedure can be applied to the whole example in figure 5 analogously. Listing 1.1 depicts the complete extracted ontology in concrete syntax that results from the domain models and constraints (cf. figure 1).

In the top part of the listing we first have the definitions of all relevant classes. Important is that the class `NetworkDevice` inherits from the anonymous class `AvailableFor value ShowPorts`. Thus all individuals of `NetworkDevice` are connected via the object property `AvailableFor` with the individual `ShowPorts`. The constraint is extracted from the correspondent annotation of `NetworkDevice` (cf. figure 1).

After the class definition the two feature individuals `ShowPorts` and `AllocatePorts` are declared.

In the following the three object properties `HasState`, `HasPort`, and `AvailableFor` are declared. The constraints `HasPort exactly 8 Port` and `HasState exactly 1 State` result from the corresponding annotations in the domain model and define the cardinality restriction in the range section of each

object property. The domain part of the `AvailableFor` property is empty, because the following global condition defines it exactly.

Listing 1.1. Ontology extracted from the domain models and constraints

```
Class: owl:Thing

Class: NetworkDevice
    SubClassOf:
        AvailableFor value ShowPorts

Class: Port
    SubClassOf:
        owl:Thing

Class: State
    SubClassOf:
        owl:Thing

Class: FreePort
    SubClassOf:
        State
    DisjointWith:
        ReservedPort

Class: ReservedPort
    SubClassOf:
        State

***********************************************************

Individual: ShowPorts
    DifferentFrom:
        AllocatePort

Individual: AllocatePorts
    DifferentFrom:
        ShowPorts

***********************************************************

ObjectProperty: HasState
    Domain:
        Port
    Range:
        HasState exactly 1 State

ObjectProperty: HasPort
    Domain:
        NetworkDevice
    Range:
        HasPort exactly 8 Port

ObjectProperty: AvailableFor

***********************************************************

NetworkDevice and (hasPort some (Port and hasState some (FreePort)))
    EquivalentTo:
    NetworkDevice and AvailableFor value AllocatePorts
```

The last part of the ontology is extracted from the global constraint of the domain model, stating when a network device entity is available for the feature `AllocatePorts`. In the first line we define an anonymous class that describes

a network device which has some port which has the state free. In the third line of this condition we define again an anonymous class which describes all network devices that are available for the *AllocatePorts* feature. These two class definitions are stated as equivalent.

Since all this information is present in the integrated model, its projection (unparsing) to plane OWL text (here in Manchester Syntax style) is easily possible.

6 Related Work

Due to the fact that this paper provides an approach of integrating domain specific languages and ontology languages the following related work concentrates on approaches where ontologies are used with domain specific modeling and different domain specific languages are integrated or connected.

Modeling complex systems usually requires different domain specific languages. Hence the need of integrating them is apparent. Because of semantic overlap of languages, where synergies can be realized by defining bridges, ontologies provide the chance of semantic integration of domain specific languages [16].

A general idea of integrating UML-based models and ontologies is the TwoUse approach (Transforming and Weaving Ontologies and UML in Software Engineering) [17]. TwoUse contains a MOF-based metamodel for UML and OWL modeling and provides a UML profile as a syntactical basis, while our approach mainly deals with domain specific languages and uses their specific syntax for modeling. The querying in TwoUse is based on OCL DL, which extends the OCL basic library with operations for querying ontologies. Our approach is to project the relevant parts of the extended domain models to an ontology and thus using standard reasoners like Pellet or FaCT++ for querying and reasoning.

The core idea in [18] is to establish semantic connectors between different models using an ontology knowledge base. This knowledge base consists of different ontologies. An upper ontology can define concepts that are applicable in most, perhaps all, domains. An application ontology specializes the concepts from the upper ontology with domain-specific variants, where the constructs of this ontology are mapped to the metamodels of different domain specific languages. In fact in this approach ontologies are not integrated within the modeling languages itself. Instead of our approach the languages are not integrated at the metamodel level, they are connected by separate ontologies.

In [19] a domain composition approach is presented which occurs at the metamodel level. Here two types of relationships can be established between concepts of two different metamodels: associations, which are similar to UML associations and correspondences, which relate overlapping concepts. These so called meta-relationships are domain invariants and are shared by all applications. In addition relationships between models at the M1-layer are introduced in to allow designers to connect their models. Our approach tries to provide an integrated modeling at the M1-layer so that the designer can create one model which consists of different languages. The connection of different modeling languages is

forced by the integrated metamodel, thus in our approach the designer cannot connect different modeling languages manually at the M1 level.

An approach of semantic tool integration is presented in [20]. The approach uses an integrated data model and the integration is tool-based. Here the data model of a tool is mapped into other data models. In [20] the authors propose to establish an integrated data model first, and then map the data model of each tool into it. The integrated data model can be defined as a data model that is rich enough to contain data from any of the tools. Our idea is similar in a way that we first want to provide our integrated metamodel that is rich enough to contain all languages separately but also in common.

7 Conclusion

Using a non-trivial example, this paper showed that multi-way merge of domain specific languages on the level of metamodels is feasible. In more detail we showed that ontology languages like OWL can be included in this integration leading to a natural usage of OWL as a part of integrated DSL models.

The work described here was done manually. A prototype implementation as a proof of concept is being developed using the TGraph approach [21] that proved to be a good implementation means for software engineering tools.

The integration steps were described semi-formally to focus on the application of the approach. The rules used in section 4 were not formalized using some model transformation language (like e. g. ATL [22]) in order not to overload the paper by too much formal text.

Further work will focus on the formalization and implementation of the work described in this paper.

Acknowledgement. Our thanks go to Krzysztof Miksa and his colleagues from Comarch for providing us with the example that inspired this paper. Furthermore this work is supported by EU STReP-216691 MOST.

References

1. Comarch: Definition of the case study requirements. MOST Project Deliverable (July 2008), `http://www.most-project.eu`
2. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Software Engineering Institute (1990)
3. Motik, B., Patel-Schneider, P.F., Parsia, B.: OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax (December 2008), `http://www.w3.org/TR/2008/WD-owl2-syntax-20081202/`
4. IEC, International Engineering Consortium: Operations Support Systems (OSS) (2007), `http://www.iec.org/online/tutorials/oss/`
5. Horridge, M., Drummond, N., Goodwin, J., Rector, A., Stevens, R., Wang, H.: The Manchester OWL Syntax. In: OWLED 2006 Second Workshop on OWL Experiences and Directions, Athens, GA, USA (2006)
6. OMG: Ontology Definition Metamodel. Object Modeling Group (November 2007)

7. Kelly, S., Tolvanen, J.: Domain-Specific Modeling. John Wiley & Sons, Chichester (2007)
8. Bézivin, J., Gerbé, O.: Towards a Precise Definition of the OMG/MDA Framework. In: Proceedings of the 16th IEEE international conference on Automated software engineering, Washington, DC, USA. IEEE Computer Society Press, Los Alamitos (2001)
9. OMG: Meta Object Facility (MOF) Core Specification (January 2006), http://www.omg.org/docs/formal/06-01-01.pdf
10. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Patel-Schneider, P.: The description logic handbook: theory, implementation, and applications. Cambridge University Press, New York (2003)
11. Happel, H.J., Seedorf, S.: Applications of Ontologies in Software Engineering. In: International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006), Athens, USA (November 2006)
12. Gasevic, D., Djuric, D., Devedzic, V.: MDA-based automatic OWL ontology development. International journal on software tools for technology transfer (Print) 9(2), 103–117 (2007)
13. Parreiras, F., Staab, S., Winter, A.: On marrying ontological and metamodeling technical spaces. In: Foundations of Software Engineering, pp. 439–448. ACM Press, New York (2007)
14. Kang, K., Lee, J., Donohoe, P.: Feature-oriented product line engineering. IEEE Software 19(4), 58–65 (2002)
15. Sun, J., Zhang, H., Wang, H.: Formal Semantics and Verification for Feature Modeling. In: ICECCS 2005: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems, Washington, DC, USA, pp. 303–312. IEEE Computer Society, Los Alamitos (2005)
16. Gaševic, D., Djuric, D., Devedzic, V., Damjanovic, V.: Approaching OWL and MDA Through Technological Spaces. In: The 3rd Workshop in Software Model Engineering (WiSME 2004), Lisbon, Portugal (2004)
17. Silva Parreiras, F., Staab, S., Winter, A.: TwoUse: Integrating UML Models and OWL Ontologies. Arbeitsberichte aus dem Fachbereich Informatik 16/2007, Universität Koblenz-Landau, Fachbereich Informatik (April 2007)
18. Brauer, M., Lochmann, H.: Towards Semantic Integration of Multiple Domain-Specific Languages Using Ontological Foundations. In: Proceedings of 4th International Workshop on (Software) Language Engineering (ATEM 2007) co-located with MoDELS (2007)
19. Estublier, J., Ionita, A., Vega, G.: A Domain Composition Approach. In: Proc. of the International Workshop on Applications of UML/MDA to Software Systems (UMSS), Las Vegas, USA (June 2005)
20. Karsai, G., Gray, J., Bloor, G., Works, P.: Integration of Design Tools and Semantic Interoperability. In: Engineering and Technical Management Symposium, Dallas (2000)
21. Ebert, J., Riediger, V., Winter, A.: Graph Technology in Reverse Engineering, The TGraph Approach. In: Gimnich, R., Kaiser, U., Quante, J., Winter, A. (eds.) 10th Workshop Software Reengineering (WSR 2008), Bonn, GI. GI Lecture Notes in Informatics, vol. 126, pp. 67–81 (2008)
22. Jouault, F., Kurtev, I.: Transforming Models with ATL. In: Bruel, J.-M. (ed.) MoDELS 2005. LNCS, vol. 3844, pp. 128–138. Springer, Heidelberg (2006)