Combining ontology-enriched Domain-Specific Languages*

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. Domain-specific languages (DSLs) are high-level and should provide abstractions and notations for a better understanding and easier modeling of applications in a special domain. It is essential to combine different DSLs while modeling a complete system. Hence the question arises: How can we combine DSLs and in addition integrate their semantics and constraints? Based on a case study from an industrial partner, in this paper we first show how DSLs can be enriched by the ontology language OWL to define additional constraints and semantics within the DSL. Then we give an answer of how to combine these ontology-enriched DSLs and present a solution of integrating constraints and semantics of several languages.

1 Introduction

Today *domain-specific languages* (DSLs) are used to model and develop systems of different application domains. Such languages are high-level and should provide abstractions and notations for better understanding and easier modeling of applications of a special domain. A variety of different domain-specific languages and fragmentations of their models may be used to develop one large software system. Each domain-specific language focuses on different problem domains and as far as possible on automatic code generation [1].

Besides different domain-specific modeling concepts there are different rules, constraints and semantics. These rules and semantics have to be defined, too, to provide different benefits like error prevention, guidance and consistency [1]. Here ontology languages come into play, because they support the definition of constraints, rules and semantics by describing them by logic based concepts.

Description logics is a family of logics for concept definitions and are used to describe domain knowledge and allow specifying concepts by rich, precise logical definitions [2]. OWL2, the Web Ontology Language, is a W3C recommendation with a very comprehensive set of constructs for concept definitions [3]. Advantages of OWL2 are the support of other (domain-specific) languages by enabling

* This work is supported by EU STReP-216691 MOST.

validation or automated consistency checking. Furthermore ontology languages provide a better support for reasoning than MOF-based languages [4], like OCL.

In this paper we use structural domain-specific languages, which are widely used in domain modeling and are instantiable. It might be useful to combine these structural domain-specific languages and ontology languages at the metamodel layer [5] to allow usage of ontology languages together with DSLs. Having this integration it is possible for the domain modeler to define different semantics and rules at the model layer which have to be fulfilled at the instance layer.

It is essential to combine different DSLs while modeling a complete system, for example to allow multi-viewpoint modeling or to check if the models defined by using overlaping DSLs still are consistent. In this paper we want to show how the semantics and constraints of several languages can be combined.

In this paper first we show how DSLs can be integrated with ontology languages to enrich their exprissiveness. The main contribution of this paper is to consider already ontology-enriched DSLs with integrated constraints and semantics and to combine these languages. In [5] we have proposed different metamodel transformations that can be used for language integration. Now we reconsider these transformations with strong regard to the constraints and semantics of the languages that have to be handled by the transformation. The task is to define the OWL-constraint part of the transformation in such a way that the resulting constraints are compatible with OWL semantics.

Generally, our approach should be useable for integrating several domain-specific languages and their models and in addition the combination of constraints and semantics the domain models contain, while the constraints could be expressed by e.g. OCL or other constraint languages.

In section 2 of this paper we start with two different scenarios. The first scenario considers the integration of DSLs and the ontology language OWL2 at the metamodel level. The second scenario considers the integration of two ontology-enriched DSLs. Section 3 presents all languages used in this paper and simultaneously provides a solution for the first scenario. Afterwards, section 4 considers the second scenario and presents a solution of integrating DSLs at the metamodel layer and its semantics at the model layer. Finally, we present some related work and give a conclusion.

2 Scenario

Comarch[3], a polish IT company, specialized in software for telecommunication providers, uses different model-driven methods for software development where different kinds of domain-specific languages (DSL) are deployed during the modeling process. One of the languages Comarch uses is the *Business Entities Domain-Specific Language* (BEDSL). BEDSL is platform independent and is used to model different business entities in the scope of the operation support system that Comarch provides. A second language Comarch uses is the *Physical Device Domain-Specific Language* (PDDSL). The language provides concepts

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

of the domain of network devices, e.g. *Chassis*, *Slot*, *Card* or *Port*. In general PDDSL defines the structure of physical network devices. Both languages, their abstract syntax and examples in concrete syntax, are presented in section 3.2.

In the following we consider two different scenarios, schematically depicted in figure 1. The first scenario deals with integrating ontology languages and one of the two Comarch DSLs resulting in a new more expressive language. Using this we are able to create domain models and to define several constraints within the domain models. An approach for handling this scenario was presented in [5], its solution for BEDSL and PDDSL is given in section 3.2. The second scenario is about the integration of two different DSLs which are already enriched by an ontology language. Here we consider the enriched *BEDSL+OWL* and *PDDSL+OWL* and already existing domain models which conform to these languages. An approach solving this scenario is presented in section 4.

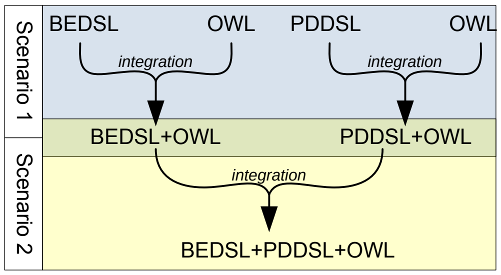


Fig. 1. Overview and dependencies of scenarios considered in this paper

2.1 Integration of DSLs and Ontology Languages.

Studying the Comarch BEDSL and PDDSL presented in section 3.2 we ascertain that the languages are very simple to use but therefore do not provide much expressiveness. For instance, BEDSL is only focusing on *business objects* represented by *entities* and their *attributes*. Thus it makes sense to allow annotations of model elements which are defined by simple OWL statements. For example domain modelers may want to annotate model elements with very simple OWL text to define additional constraints and semantic annotations, if they feel that the language is not expressive enough. Equally, PDDSL is not able to define restrictions or special configurations of physical devices. For example it is not possible to describe with PDDSL that a device is a *Cisco7603 Router* if and only if it has a configuration with exactly three slots in which specific cards are plugged in. Here again ontologies can be used and integrated within PDDSL. Thus more complex concepts like the one of the *Cisco7603* router can be described.

In general it makes sense to integrate the DSLs and an ontology language to provide advantages like constraint definition, formal semantics and reasoning. In this scenario the question arises how a metamodel of a DSL and the one of an ontology language can be integrated. We gave a first answer in [5]. There we provide different transformations for integrating metamodels, which mainly consider the relation between two classes/concepts of the two different metamodels that are to be combined. The combination for example could be the merge of two metaclasses, the creation of an inheritance relationship between them or only the creation of an metaassociation. Thus we are able to create domain models with model elements that can be annotated with simple OWL text. The adoption of such an integration is presented in section 3.2.

Thus we realize our intention that a designer should use the language he is familiar with as much as he can. If he recognizes that it is not expressive enough he is able to use additional annotations. Furthermore a change of the existing infrastructure is not necessary.

2.2 Merge of ontology-enriched DSLs.

An advantage of BEDSL is that there exists support for storing BEDSL models in *relational databases*. Furthermore *Comarch* aims also to store models of physical network devices in relational databases. Thus, it makes sense to integrate both languages, the BEDSL with its capabilities for databases and the PDDSL with its features for describing network devices.

During the integration existing semantics and constraints of each separate domain model are also to be integrated. In most cases they overlap or depend on each other. Here inconsistencies can occur, and existing semantics can be changed and thus can lead to incorrect domain models.

Thus, in this scenario the question arises how domain models with additional, embedded formal constraints and semantics can be merged. In section 4 we present a first *combination approach* where the integration of models plus its semantics and constraints is considered in the context of the new integrated metamodel which contains a BEDSL-, a PDDSL- and an OWL-part.

3 Integration of DSLs and Ontology Languages

In this section we want to present three different languages. In section 3.1 we present an example of using the axiom-based ontology language OWL2. A detailed specification of OWL2 can be found in [3]. In section 3.2 we present the integrated metamodel of the *Business Entity Domain-Specific Language* (BEDSL) extended by OWL2. Furthermore we show in section how the *Physical Device Domain-Specific Language* (PDDSL) is extended by the OWL2 metamodel. For all metamodels we also give an example in concrete syntax.

3.1 OWL2

In general description logics based ontology languages are used to define sets of concepts that describe domain knowledge and allow specifying classes by rich, precise logical definitions. Among different ontology languages, we use the W3C standard OWL2 in this paper. OWL actually stands for a family of languages with increasing expressiveness. OWL2, the current draft, is more expressive and still allows for sound and complete reasoning that is decidable as well as pragmatically efficient.

The difference between OWL and class-based modeling languages like UML class diagrams is its capability to describe classes in many different ways and to handle incomplete knowledge. These OWL features increase the expressiveness of the metamodeling language, making OWL2 a suitable language to formally define DSL models. Therefore, our intention is to integrate OWL2 with the DSL such that domain models with additional, embedded constraints can be defined.

OWL2 is axiom-based and thus provides different constructs to restrict classes or properties. The three important kinds of axioms are class axioms, object property axioms and data property axioms. A detailed specification of the ontology language OWL2 and its axioms is presented in [3].

Using a concrete syntax in Manchester Style [6], we only give an example of using the OWL2 language in figure 2. Here we first define the classes `Device`, `Cisco7603` a subclass of `Device`, `Configuration` and `Configuration7603`, a subclass of `Configuration`. Further, we define the object property `hasConfiguration` with its domain `Device` and its range `Configuration`. At last, using the `SubClassOf` axiom, we state, that a `Cisco7603` has exactly one `Configuration7603`.

```
Class: Device

Class: Cisco7603
    SubClassOf:
        Device,
        hasConfiguration exactly 1 Configuration7603

Class: Configuration

Class: Configuration7603
    SubClassOf:
        Configuration

ObjectProperty: hasConfiguration
    Domain:
        Device
    Range:
        Configuration
```

Fig. 2. Example of using the OWL2 metamodel

3.2 DSL Metamodels integrated with OWL2

In [5] we presented a way of how to extend domain-specific languages by ontologies. The general idea was to integrate the metamodels of a DSL and the metamodel of an ontology language like OWL2. Hence we were able to adopt OWL class axioms on classes and different OWL property restrictions on associations at the model layer (M1 layer).

In this section we want to describe two integrated metamodels. The metamodels come from our project partner Comarch and are precisely presented in [7]. In the following both metamodels are separately integrated with the metamodel of the ontology language OWL2. Thus using them we are able to define DSL models with embedded constraints and formal semantics defined using OWL2 statements.

Business Entities Domain-Specific Language. The *Business Entities Domain-Specific Language* (BEDSL) is a domain-specific language intended to model business entities. BEDSL is platform independent, abstracting from any specific technology, focusing on representing business objects, like entities, their attributes and relations. Thus it provides a good basis to save such domain models in relational databases.

Figure 3 depicts an example in concrete syntax of using the enriched BEDSL. In such a diagram all business entities of Cisco devices and in our case its specialization Cisco7603 can be defined. A Cisco device can contain `CiscoSlots` and `CiscoCards`. In the example we have three specializations of Cisco cards, namely `HotSwappable` card, `SPAInterface` card and `Supervisor` card.

The technical specification of a Cisco device requires that a Cisco has at least a `Supervisor` card and has exactly 3 slots. These requirements are defined in the annotation (1). Furthermore we define that if and only if a Cisco device contains any `Card` then of course it contains a `CiscoSlot` (2).

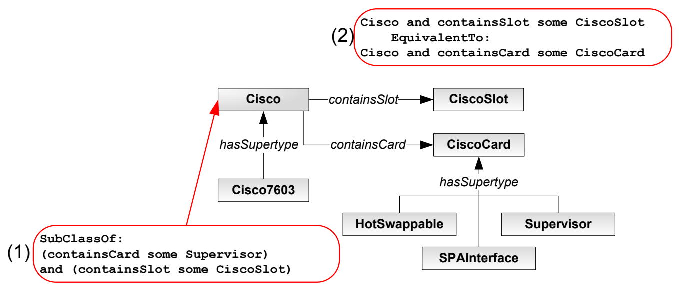


Fig. 3. Example of using the enriched BEDSL

Figure 4 presents the metamodel of BEDSL. Here we first have a `Model` which is a container for entities. The purpose of an `Entity` is to collect a set of attributes. `Attributes` can be either a `SimpleAttribute` or a `ReferenceAttribute`. Simple attributes have a `Datatype` which defines the type of values of the attribute. An `Enumeration` is a specific data type which defines all values it can contain. Reference attributes are responsible for the connection of two entities.

Furthermore the BEDSL language is integrated with the ontology language OWL2 using the approach defined in [5]. `Entity` is a specialization of OWL `Class`. Thus it is possible to adopt several OWL2 class axioms on entities. The relations between entities and its attributes are extended by separate classes. The class `ReferenceProperty` connects `Entity` with `ReferenceAttribute` and simultaneously inherits from OWL `ObjectProperty`. Analogously we extend the relation between `Entity` and `SimpleAttribute` by the class `SimpleProperty` which inherits from OWL `DataProperty`. Thus we are able to adopt different OWL object property and data property axioms on relations between entities and its attributes. In the following we call the enriched language *BEDSL+OWL*.

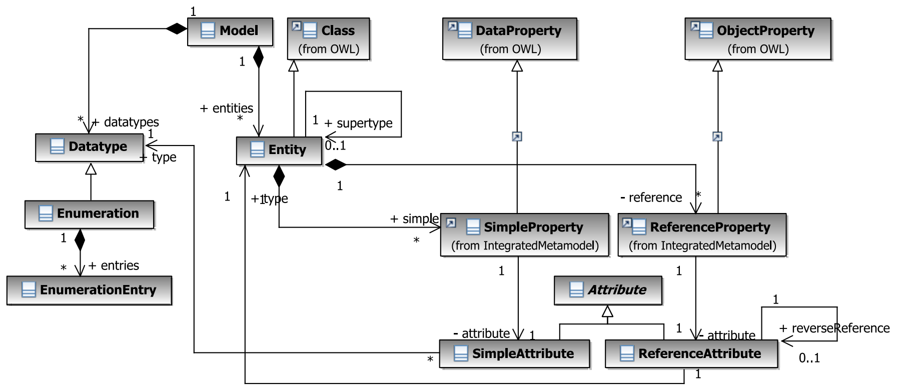


Fig. 4. Integrated Metamodel of the Business Entities Domain-Specific Language (BEDSL+OWL)

Physical Device Domain-Specific Language. To describe possible connections between physical device elements the *Physical Device Domain-Specific Language* (PDDSL) is used.

Figure 5 depicts an example in concrete syntax of using the enriched PDDSL. Here in general a *Cisco7603* with its possible configurations is modeled. Using simple OWL annotations we define that `Cisco7603` is a subclass of `Cisco` (1), `Configuration7603` is a subclass of `CiscoConfiguration` (2) and `HotSwappable`, `SPAInterface` and `Supervisor` are subclasses of `CiscoCard` (3). Furthermore we define that each Cisco device has some `CiscoConfiguration` (4) and each `Cisco7603` only has exactly one `Configuration7603` (5). Furthermore by defining a global constraint, we state that if and only if a `Configuration7603` contains a `HotSwappable` card it also has a `SPAInterface` card (6).

The metamodel of this language is presented in figure 6. PDDSL describes physical devices and thus provides concepts like `Shelfs`, `Chassis`, `Slots` and `Cards`. `Shelfs` and `Chassis` can be connected with other `Cards` through `Slots`. Concrete configurations (a valid set of `Slots` with compatible `Cards`) are encapsulated in a `Configuration`. Every `Shelf` or `Chassis` contains many possible configurations.

Analogously to the integration of BEDSL and OWL2 we have integrated an ontology language with PDDSL. Here, `Element` inherits from OWL Class. Thus it is possible to adopt several OWL class axioms on elements of a physical device.

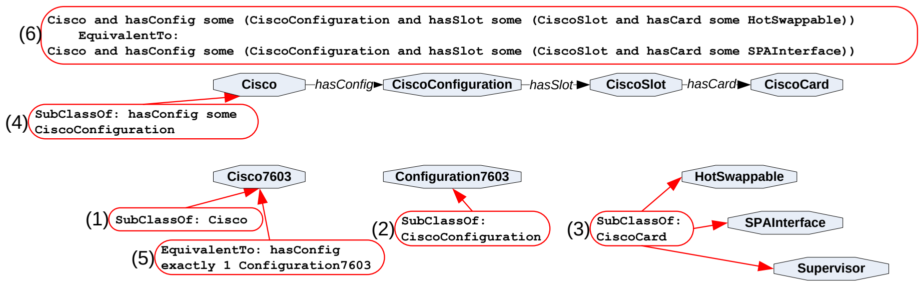

(6)
Cisco and hasConfig some (CiscoConfiguration and hasSlot some (CiscoSlot and hasCard some HotSwappable))
EquivalentTo:
Cisco and hasConfig some (CiscoConfiguration and hasSlot some (CiscoSlot and hasCard some SPAInterface))
Cisco
hasConfig
CiscoConfiguration
hasSlot
CiscoSlot
hasCard
CiscoCard
(4)
SubClassOf: hasConfig some CiscoConfiguration
Cisco7603
Configuration7603
HotSwappable
(1)
SubClassOf: Cisco
(2)
SubClassOf: CiscoConfiguration
(3)
SubClassOf: CiscoCard
SPAInterface
(5)
EquivalentTo: hasConfig exactly 1 Configuration7603
Supervisor

Fig. 5. Cisco Device Domain Model using the enriched PDDSL

Furthermore we materialize the relation between a `SlotContainer` and its configurations by the separate class `ConfigurationProperty`, the relation between `Configuration` and `Slot` by the separate class `SlotProperty` and the relation between `Slot` and `Card` by the separate class `CardProperty`. All property classes inherit from OWL object property. Hence it is allowed to adopt different OWL object property axioms on these relations. In the following we call the enriched language *PDDSL+OWL*.

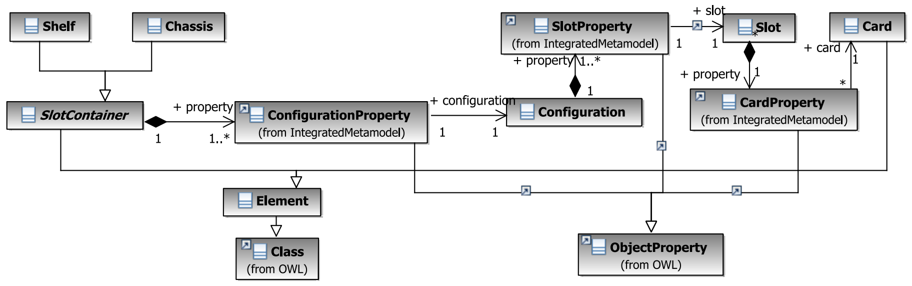


Fig. 6. Integrated Metamodel of the Physical Device Domain-Specific Language (PDDSL+OWL)

4 Merge of ontology-enriched DSLs

In this section we want to consider how the metamodels of *BEDSL+OWL* and *PDDSL+OWL* can be integrated and how the constraints behave after the integration. At first we present the general approach of combining models and additional constraints. Afterwards we give an example and concentrate on two integration transformations, namely the *Merge Transformation* and the *Specialization Transformation*, presented in [5] and precisely described in section 4.4. Here we first look at the definition of the transformation at the M2 layer, and later we will examine how domain models and their constraints at the M1 layer, which are part of the domain models, change after applying the transformation on them.

4.1 General Approach

If two metamodels are combined, the integration transformations have to consider the different semantics of the two languages. Model elements of two different languages can only be combined, if the result conforms to the already existing semantics of the two different languages.

To present an approach for solving the problem of merging semantic annotations of model elements and constraints, we have to consider the integration transformations on two different layers. First the transformation is defined at the M2-layer where it defines which different classes of the different metamodels are to be connected (e.g. merged).

Then we have to define at the domain model layer (M1-layer) which model elements (instances of concepts in the metamodels) are considered by the transformation (e.g. some of them can be merged). Here the domain knowledge of a DSL expert is important, which leads to intensional semantics of concepts at the M1-layer. These semantics later are represented by description logics axioms and are used to decide which concepts are transformed. The result of the execution of the transformation at the M2-layer is an integrated metamodel and a corresponding integration of the models.

The result of the execution of the transformation at the M1-layer is a modified abstract syntax graph of the domain model. This new abstract syntax graph describes how the considered instances are connected corresponding to the applied transformation. It further must define how the abstract syntax parts of the annotations and constraints are combined. This combination depends mainly on the transformation and the kind of annotation (e.g. on the type of axiom used in an annotation of a model element).

Considering the scenario of integrating BEDSL+OWL and PDDSL+OWL, two model elements, both annotated with a constraint, that are selected by a domain expert to be merged to one single element have to fulfill both constraints. In fact, the merged element has to keep the constraints given for the origin BEDSL+OWL model and it has to keep the constraints for the origin PDDSL+OWL model.

4.2 Integration Transformations

In the following we present two transformations that are used to combine different metamodels. The transformations themselves are defined at the M2 layer. Furthermore we also depict how the instances at the M1 layer are changed and define which instances at the M1 layer are considered by the transformation by stating a pre-condition defined by OWL axioms.

Merge Transformation. Figure 7 depicts the merge transformation on the M2 and M1 layer. Generally the two meta-concepts `MA` and `MB` are identified to be merged and are replaced by the meta-concept `MC` at the M2 layer. In addition the model elements at the M1 layer, instances of the corresponding meta-concepts at the M2 layer and visualized by a domain-specific concrete syntax, can be merged, too. Furthermore, the concrete syntax model at the M1 layer acts as new structural language which is able to be instantiated. Its instances lie at the M0 layer (not depicted in figure 7).

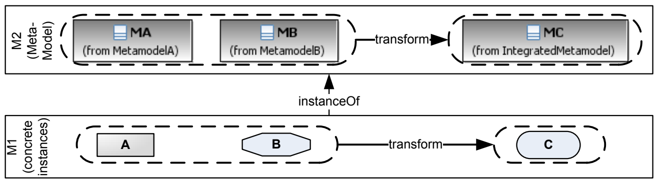


Fig. 7. Merge Transformation

Based on intensional knowledge of domain experts we can assume that instances `A` and `B` of `MA` and `MB` are identically if they have equal names (e.g. the name attribute of the instance is equal). Furthermore if `A` and `B` are identically we can assume that they have the same potential instances at the M0 layer.

We can describe this intensional knowledge by the following semantic description, which further acts as a pre-condition of the merge transformation:

```
Class: A EquivalentWith: B
```

The merge transformation can be applied on the M2 layer and in addition on all identical instances, if the pre-condition holds for at least two instances `A` and `B`. Constraints, which are annotated at the instances `A` and `B` are combined and transformed to one single constraint of the target instance `C`. The new constraint depends on the kind of axioms annotated at `A` and `B` and the pre-condition (cf. section 4.3). Instances of type `MA` and `MB` which cannot be matched to identical ones are transferred together with their constraints without any change. Their new type is `MC`.

Specialization Transformation. Figure 8 depicts the specialization transformation on the M2 and M1 layer. Generally the two meta-concepts `MA` and `MB` are identified to be connected by a specialization relationship. In addition the model elements `A` and `B` at the M1 layer, instances of the corresponding meta-concepts at the M2 layer can be transformed too.

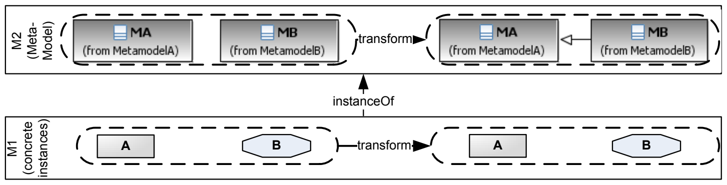


Fig. 8. Specialization Transformation

Based on intensional knowledge of domain experts we assume that an instance `A` of `MA` is a super-concept of an instance `B` of `MB`, if the name of `A` is prefix of the name of `B` (like *Cisco* of *Cisco7603*). Furthermore if `A` is a super-concept of `B` we assume that the set of instances of `B` is a subset of instances of `A` at the M0 layer.

As a consequence we can describe this intensional domain knowledge by the following semantic description, which further acts as a pre-condition of the specialization transformation:

```
Class: B SubClassOf: A
```

The specialization transformation can be applied on the M2 layer and in addition on all instances `A` and `B`, which fulfill the pre-condition. Constraints, which are annotated at the instance `A` are combined by the transformation with the ones of `B`. Instances of `MA` and `MB` which do not match the sub-concept condition are again together with its constraints transferred without any changes.

4.3 OWL Class Axioms

In the following we want to consider the two above presented integration transformations and consider the constraints and additional semantics, which are defined by annotations. In fact using OWL as a constraint language these constraints are OWL class axioms. Thus in the following we will consider the combination of class axioms and transformations.

– The *SubClassOf* axiom allows to state that each instance of a subclass is also an instance of the corresponding superclass. Table 1 depicts the resulting constraints after one transformation is applied on two M1 concepts `A` and `B` and its corresponding *SubClassOf* axiom. Here the pre-condition of the transformation is taken into account.

Before any Transformation	*After Merge Transformation*	*After Specialization Transformation*

`Class: A SubClassOf: ClassExpressionA`
 	
`Class: B SubClassOf: ClassExpressionB`
	`Class: C SubClassOf: (ClassExpressionA and ClassExpressionB)`	`Class: A SubClassOf: ClassExpressionA`
		
		`Class: B SubClassOf: A and (ClassExpressionA and ClassExpressionB)`

Table 1. Combination of SubClassOf Axioms

– The *EquivalentClasses* axiom allows to state that several class expressions are equivalent to each other. Table 2 depicts the resulting constraints after one transformation is applied on two M1 concepts `A` and `B` and its corresponding *EquivalentClasses* axiom. Here the pre-condition of the transformation is taken into account.

– The *DisjointClasses* axiom allows one to state that several class expressions are pairwise disjoint. Table 3 depicts the resulting constraints after one transformation is applied on two M1 concepts `A` and `B` and its corresponding

Before any Transformation	*After Merge Transformation*	*After Specialization Transformation*
`Class: A EquivalentTo: ClassExpressionA` `Class: B EquivalentTo: ClassExpressionB`	`Class: C EquivalentTo: ClassExpressionA`	`Class: A EquivalentTo: ClassExpressionA` `Class: B EquivalentTo: ClassExpressionB SubClassOf: A`

Table 2. Combination of EquivalentClasses Axioms

DisjointClasses axiom. Here the pre-condition of the transformation is taken into account.

Before any Transformation	*Merge Transformation*	*Specialization Transformation*
`Class: A DisjointWith: ClassExpressionA` `Class: B DisjointWith: ClassExpressionB`	`Class: C DisjointWith: ClassExpressionA, ClassExpressionB`	`Class: A DisjointWith: ClassExpressionA` `Class: B DisjointWith: ClassExpressionB SubClassOf: A`

Table 3. Combination of DisjointClasses Axioms

– The *DisjointUnion* axiom allows to define a class as a disjoint union of several class expressions and thus to express covering constraints. Table 4 depicts the resulting constraints after one transformation is applied on two M1 concepts `A` and `B` and its corresponding *DisjointUnion* axiom. Here the pre-condition of the transformation is taken into account.

Before any Transformation	*After Merge Transformation*	*After Specialization Transformation*
`Class: A DisjointUnionOf: ClassExpressionA` `Class: B DisjointUnionOf: ClassExpressionB`	`Class: C DisjointUnion: ClassExpressionA DisjointUnion: ClassExpressionB`	`Class: A DisjointUnionOf: ClassExpressionA` `Class: B DisjointUnionOf: ClassExpressionB SubClassOf: A`

Table 4. Combination of DisjointUnion Axioms

If after a transformation two axioms of different types are connected to one merged or specialized model element, they are not combined in our approach. For example in table 5, after applying the merge transformation the *EquivalentWith*- and the *SubClassOf* axiom are not combined. Although the transformed model element `C` (resp. `A` and `B`) at the M1-layer fulfills both axioms/constraints because the axiom-based OWL2 ontology, which later could be extracted from the merged domain model, provides a conjunction of all axioms.

Before any Transformation	*After Merge Transformation*	*After Specialization Transformation*

`Class: A SubClassOf: ClassExpressionA`
 	
`Class: B EquivalentTo: ClassExpressionB`
	`Class: C SubClassOf: ClassExpressionA EquivalentTo: ClassExpressionB`	`Class: A SubClassOf: ClassExpressionA`
		
		`Class: B EquivalentTo: ClassExpressionB`

Table 5. Combination of different Axioms

4.4 Example: Merge Transformation

In the following we consider the merge transformation. At first we present its definition at the M2 layer. Afterwards we give an idea of how to use it by considering the domain models presented in section 3.2.

Transformation at the M2 layer. As presented in section 3.2, *BEDSL+OWL* provides the concept `Entity` and *PDDSL+OWL* provides the concept `Element`. Both concepts can be identified to be merged, because to apply the merge transformation we require that some instance of `Entity` is identical with some instance of `Element`. As later shown, this will be the `Cisco` element at the M1 layer. Figure 9 depicts an instance of the *Merge Transformation* on the M2 layer. Result of the transformation is the new class `EntityElement`, which merges the single concepts of `Element` and `Entity`. All OWL constructs which are connected with `Element` and `Entity` at the M2 layer are combined by the transformation as well.

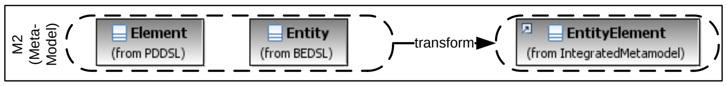


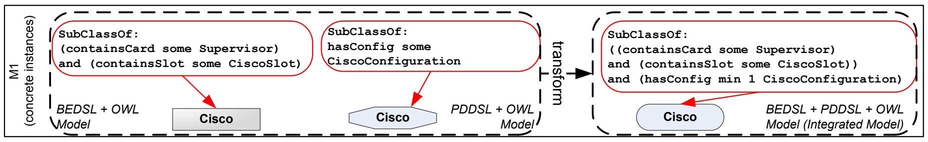
Fig. 9. Merge Transformation at the M2-layer

Transformation at the M1 layer. After presenting the merge transformation at the M2 layer we show in the following how it is adopted on the M1 layer. In the left side of figure 10 we have excerpts of two domain models conforming to the *BEDSL+OWL* and *PDDSL+OWL* metamodel. Both models contain the concept `Cisco` and we assume that these concepts are equivalent. Thus the precondition axiom for the merge transformation is fulfilled and we can merge both to one single concept `Cisco`, which now is instance of `EntityElement`.

Because both `Cisco` concepts in the domain models on the left are restricted by constraints, these constraints have to be considered by the transformation, too. In the example the transformation at the M1 layer has to identify if the two concepts to be merged both are annotated by a `SubClassOf` axiom. If so, the transformation has to combine the two annotations automatically. In the example in figure 10 the class expression `(containsCard some Supervisor) and (containsSlot some CiscoSlot)` and the class expression `hasConfig some`

CiscoConfiguration are combined by using an and-operator. After the transformation a valid Cisco still has to contain a Supervisor card and it still has to be connected to at least one CiscoConfiguration.

Besides the constraints annotated directly at the Cisco model elements there exists one global constraint in each of the domain models (not depicted in figure 10). These constraints are not merged because in our approach only the ones with equal class expressions are merged. In fact, this is not the case here and thus the global constraints are only transferred independently and without any change to the integrated model.



```
SubClassOf:
(containsCard some Supervisor)
and (containsSlot some CiscoSlot)
```

```
SubClassOf:
hasConfig some
CiscoConfiguration
```

```
SubClassOf:
((containsCard some Supervisor)
and (containsSlot some CiscoSlot))
and (hasConfig min 1 CiscoConfiguration)
```

Fig. 10. Merge of the M1-concept Cisco presented in conrete syntax

5 Related Work

The approach presented in [8] defines references between elements of different domain-specific models. References are modeled explicitly and connect elements with same names. In our example we also have combined model elements by name (the two Cisco elements were merged), but we consider the semantics and constraints of both elements. Only if the pre-condition of the integration transformation is fulfilled we can apply the merge.

In [9] an approach is presented where the semantics of domain models are considered during the integration. The general idea is to establish semantic connectors between different models using an ontology knowledge base. This knowledge base consists of different ontologies. An upper ontology defines concepts that are applicable in most, perhaps all, domains. An application ontology specializes the concepts from the upper ontology with domain-specific variants and maybe adds additional constraints and semantics. 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 like in our approach.

[10] present some issues on ontology integration. It considers problems of how to integrate several existing ontologies within a new one that is being built. One solution is the specification of a set of integration operations that tell how knowledge in the ontology is going to be included and combined with the knowledge in the resulting ontology. Instead of pure ontologies our approach considers domain models with integrated ontologies. Although we also provide a set of transformations/operations which define, how the ontology parts are combined.

6 Conclusion

In this paper we have presented an approach of combining constrained domain models. In section 3 we presented two DSLs, namely BEDSL and PDDSL and showed how they are integrated with the ontology language *OWL2*. Thus it is possible to define several class axioms adopted on concepts of domain models or property restrictions adopted on attributes of concepts. Because there is the need to integrate the enriched BEDSL and PDDSL we presented the idea of integrating the domain models and simultaneously combining the constraints of each domain model in section 4. Therefore we extended our integration approach presented in [5]. Each integration transformation has a pre-condition and can be applied if it is fullfilled.

The transformation considers the additional constraints of enriched domain models. If constraints exist the right combination of them has to be applied. We have seen that these combinations are language dependent and thus have to be defined by a DSL designer before integrating the languages.

Based on the user-defined pre-condition and existing constraints in the model the transformation is able to compute the new constraints of the transformed model elements. We have shown two transformations how the combination can be derived from the pre-conditions of the transformations. Generally, our presented approach would be useable for integrating other languages and their constrained models (e.g using OCL instead of OWL2).

References

1. Kelly, S., Tolvanen, J.: Domain-Specific Modeling. John Wiley & Sons (2007)
2. 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)
3. Motik, B., Patel-Schneider, P.F., Horrocks, I.: OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. `http://www.w3.org/TR/2008/WD-owl2-syntax-20081202/` (December 2008)
4. Happel, H.J., Seedorf, S.: Applications of ontologies in software engineering. In: International Workshop on Semantic Web Enabled Software Engineering (SWESE'06), Athens, USA (November 2006)
5. Walter, T., Ebert, J.: Combining DSLs and Ontologies using Metamodel Integration. In: Domain-Specific Languages. Volume LNCS., Springer (2009) 148–169
6. Horridge, M., Drummond, N., Goodwin, J., Rector, A., Stevens, R., Wang, H.: The Manchester OWL Syntax. In: OWLED2006 Second Workshop on OWL Experiences and Directions, Athens, GA, USA (2006)
7. Comarch: Case Study Design. MOST Project Deliverable (January 2009) `www.most-project.eu`(restricted to specified group).
8. Warmer, J., Kleppe, A.: Building a Flexible Software Factory Using Partial Domain Specific Models. In: 6th OOPSLA Workshop on Domain-Specific Modeling (DSM06). 15
9. 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)
10. Pinto, H., Gómez-Pérez, A., Martins, J.: Some issues on ontology integration. In: IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5), Citeseer (1999)