OntoDSL: An Ontology-Based Framework for Domain-Specific Languages

Tobias Walter[1,2], Fernando Silva Parreiras[1], and Steffen Staab[1]

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

Abstract. Domain-specific languages (DSLs) are high-level and should provide abstractions and notations for better understanding and easier modeling of applications of a special domain. Current shortcomings of DSLs include learning curve and formal semantics. This paper reports on a novel approach that allows the use of ontologies to describe DSLs. The formal semantics of OWL together with reasoning services allows for addressing constraint definition, progressive evaluation, suggestions, and debugging. The approach integrates existing metamodels, concrete syntaxes and a query language. A scenario in which domain models for network devices are created illustrates the development environment.

1 Introduction

Domain-specific languages (DSLs) are used to model and develop systems of 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 fragments of their models are 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].

There is an agreement about the challenges faced by current DSL approaches [2]: *(challenge (1))* tooling (debuggers, testing engines), *(challenge (2))* interoperability with other languages, *(challenge (3))* formal semantics, *(challenge (4))* learning curve and *(challenge (5))* domain analysis.

Addressing these challenges is crucial for the success adoption of DSLs. For example, improving tooling enhances user experience. The interoperability between different languages plays an important role, because more than one language has to be combined in the modeling of systems. Finally, formal semantics is the basis for interoperability and formal domain analysis.

Issues like interoperability and formal semantics motivated the development of ontology languages. Formal semantics precisely describes the meaning of models, such that they do not remain open to different interpretations by different

persons (or machines). For example, Description Logics formalize the W3C standard Web Ontology Language (OWL) [3] and provides a language for ontologies. Indeed, OWL, together with automated reasoning services, provides a powerful solution for formally describing domain concepts in an extensible way, allowing for precise specification of the semantics of domain concepts.

Taking into account that some of the main challenges of DSLs were motivation for developing OWL, the following two questions arise naturally: which characteristics of ontology technologies may help in addressing current DSL challenges? What are the building blocks of a solution for applying ontology technologies in DSLs?

Recent works have explored ontologies to address some DSL challenges. Tairas et al. [4] apply ontologies in the early stages of domain analysis to identify domain concepts (*challenge (5)*). Guizzard et al. [5] propose the usage of an upper ontology to design and evaluate domain concepts (*challenge (3)*) whereas Bräuer and Lochmann[6] propose an upper ontology for describing interoperability among DSLs (*challenge (2)*). Nevertheless, the application of ontology languages and ontology technologies to address the remaining *challenges (1) and (4)* as well as a comprehensive integration is an open issue.

We present *OntoDSL*, an ontology-based framework for DSLs that allows for defining DSLs enriched by formal class descriptions. It allows DSL users to check the consistency of DSLs models and helps to verify and debug DSL models by using reasoning explanation. Moreover, novice DSL users may rely on reasoning services to suggest domain concepts according to the definition of the domain-specific language.

1.1 Advantages

The pragmatic advantages of the ontology-based *OntoDSL* framework are primarily the guidance of DSL designers and DSL users during the modeling process, the support of incomplete knowledge of concepts a DSL provides and the possibility of debugging domain models. Furthermore, *OntoDSL* provides DSL users with suggestions during building domain models and progressive evaluation of domain constraints. To get these and more advantages, *OntoDSL* provides automated reasoning services that can be practically used by DSL designers and DSL users.

The correctness of the domain-specific language in development is important for DSL designers. Thus, they want to *check the consistency* of the developed language, or they might exploit information about *concept satisfiability*, checking if it is possible for a concept in the metamodel to have any instances.

When DSL users want to verify whether all restrictions and constraints imposed by the DSL metamodel hold, they can use a reasoning service to *check the consistency* of domain model. It is important for a domain model, that its elements have the most specific type. Thus, DSL users should be able to select a model element and call simply by clicking a button a reasoning service for *dynamic classification*. Dynamic classification allows for determining the classes which model objects belongs to dynamically, based on object descriptions. Later

this might be useful, for example, to generate the most specific and complete source code from it. Further, it might be interesting for DSL users to retrieve existing model elements of a model repository by describing the concept in different possible ways. Here, *OntoDSL* can support the reuse of elements with retrieval services.

1.2 Methodology

To help DSL designers to define languages compatible with the aforementioned services, we create a new technical space (M3 metametamodel). The framework with its own technical space is arranged according to the OMG's layered architecture depicted in Figure 1 and the roles of DSL user and DSL designer are assigned to the different layers they are responsible for. The metametamodel consists of the KM3 metamodel, the OWL2 metamodel and the OCL metamodel (cf. Section 4.2). KM3 is used to define the general structure of the language, OWL2 is used to define its semantics, OCL is used to define operations for calling the reasoning services.

To support the aforementioned reasoning services, we propose OWL to define constraints and formal semantics of DSLs. OWL is formalized by Description Logics, which provides the reasoning simultaneously on the M1-layer (model) and M2-layer (metamodel). In order that ontology reasoning is possible, the DSL metamodel and domain model are transformed into a Description Logics knowledge base (TBox and ABox).

In order to reduce the learning effort, *OntoDSL* allows for using the familiar Java-like KM3 syntax to a very large extent. If DSL designers recognize that it is not expressive enough they can benefit from an easy to implement OWL natural style syntax to define semantics, constraints and restrictions (cf. Section 4.3).

DSL users should not be confronted with the ontology technology. They only have to call operations that automatically invoke the reasoning services. Only DSL designers have contact with OWL, to define constraints and restrictions on the DSL metamodel.

In the scope of this paper, a DSL framework is an model-driven underlying structure to support the DSL development process and usage. Section 4 gives more details about the *OntoDSL* framework.

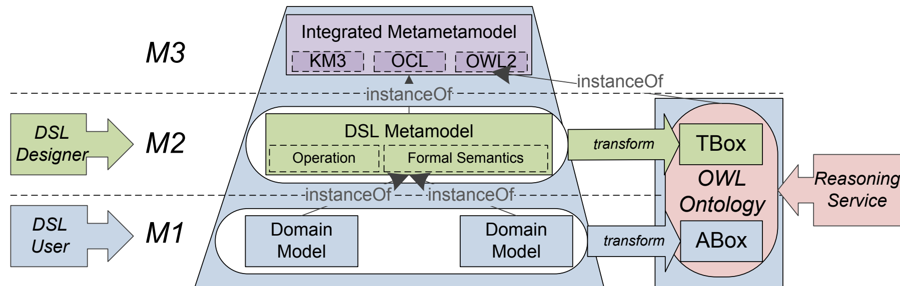

Fig. 1. DSL Designer and DSL User in the OMG' four-layered architecture

We organize the remaining sections as follows: Section 2 describes the running example used through the paper and analyzes the DSL challenges to be addressed with ontology technologies. Section 3 describes the state-of-the-art in domain-specific languages with ontology technologies and discusses the usage of Description Logics to formally describe domain models. The *OntoDSL* framework is described in Section 4 by presenting its metametamodel, its implementation and an example of using it. We revisit the running example in Section 5 and analyze related work in Section 6. Section 7 finishes the paper.

2 Running Example

Comarch[3], one of the industrial partners in the *MOST project*[4] has provide the running example used in this paper. It is a suitable simplification of the user scenario being conducted within the MOST project.

Comarch is specialized in designing, implementing and integrating IT solutions and services. For software development, Comarch uses model-driven methods where different kinds of domain-specific languages are deployed during the modeling process.

Comarch uses a domain-specific language defined using MOF (metametamodel at M3-layer) to model physical network devices. Using such a language, Comarch DSL designers design an DSL to define device structures (e.g. all devices from the Cisco 7600 family) at the M2-layer. Here, the goal of DSL designers is to formally define the logical structures of devices and restrictions over these structures (which leads to the below listed *requirement (1)*).

DSL users use DSL defined by DSL designers to write DSL models that model concrete physical devices (M1-layer). A framework that instructs and guides DSL users during this process is desirable. For example, the consistency of DSL models should be verified, domain concepts should be suggested to DSL users, incomplete parts in the models should be detected and redundancies should be removed [7].

Let us elaborate the following example: The general physical structure of a Device consists of a Bay which has a number of Shelfs. A Shelf contains Slots into which Cards can be plugged. Logically, a Shelf with its possible Slots and Cards is stored as a Configuration.

Figure 2 depicts the development of a DSL model of physical devices by a given DSL user. Firstly (*step 1*), the DSL user starts with an instance of the general concept Device. A device requires at least one configuration. Thus he plugs in a Configuration element into the device.

In *step 2*, the DSL user adds exactly three slots to the device model. At this point, the DSL user wants to verify whether the configuration satisfies the DSL restrictions, which is done, for example, by invoking a query against the current physical device model (*requirement (2)*).

[3] `http://www.comarch.com/`
[4] `http://www.most-project.eu/`

After adding three slots to the model of the physical device, the DSL user plugs in some cards to complete the end product (*step 3*). Knowing which cards and interfaces should be provided by the device, he may insert an SPA Interface Card for 1-Gbps broadband connections, a Supervisor Engine 720 card for different IP and security features and a controller for swapping cards at runtime (Hot Swap Controller).

At this point, the DSL user has reasoning services available by calling operations that are provided by model elements in his DSL model. The calling of operations in our *OntoDSL* is realized by right-clicking on model elements, opening a context menu. This menu provides a list of the reasoning services that can be directly executed in the context of the model element.

The DSL defines the knowledge of which special types of cards are provided by a Configuration. Having further the information that its instance is connected with three slots, the refinement of the Configuration type of the instance by the more specific type Configuration7603 is recommended to the DSL user (*requirement (3)*) as result of activating the reasoning service. Moreover, the DSL user is informed how this suggestion takes place and about restrictions related to such a configuration. Here, debugging support is required (*requirement (4)*).

Since it has been inferred that the device has the Configuration7603, in *step 4*, the available reasoning service for the Device element infers that the device is one of type Cisco7603. The necessary and sufficient condition to be a Cisco7603 are verified and achieved by reasoning services.

Steps *2a and 2b* shows a second path in the scenario of modeling a physical device where debugging comes into play. After creating elements for a device, a configuration and slots, the DSL user plugs into one slot a HotSwappableOSM card and into the remain slots two SPAInterface cards (*step 2a*). Here, the DSL user can invoke the debugging functionality provided by *OntoDSL*. It explains that each configuration must have a slot in which a SuperVisor720 card is plugged in. *OntoDSL* advises the type change of one of the SPAInterface elements to SuperVisor720 (*requirement (4)*). Having a correct configuration, the DSL user can continue with *steps 3 and 4* as described above.

2.1 Requirements

Although the list of requirements for a DSL framework may be extensive, we concentrate on those requirements derived from the running example and from the challenges mentioned in Section 1, mainly on *challenges 1 and 4*. The requirements are classified by two actors: DSL designer and DSL user. At first we present the ones for the DSL designer:

1. *Constraint Definition* (*challenge (3)*). The DSL development environment should allow defining constraints over the DSL metamodel. DSL designers have to define formal semantics of the DSL in development to describe constraints and restrictions the DSL program have to fulfill.

The following requirements we assign to the DSL user:

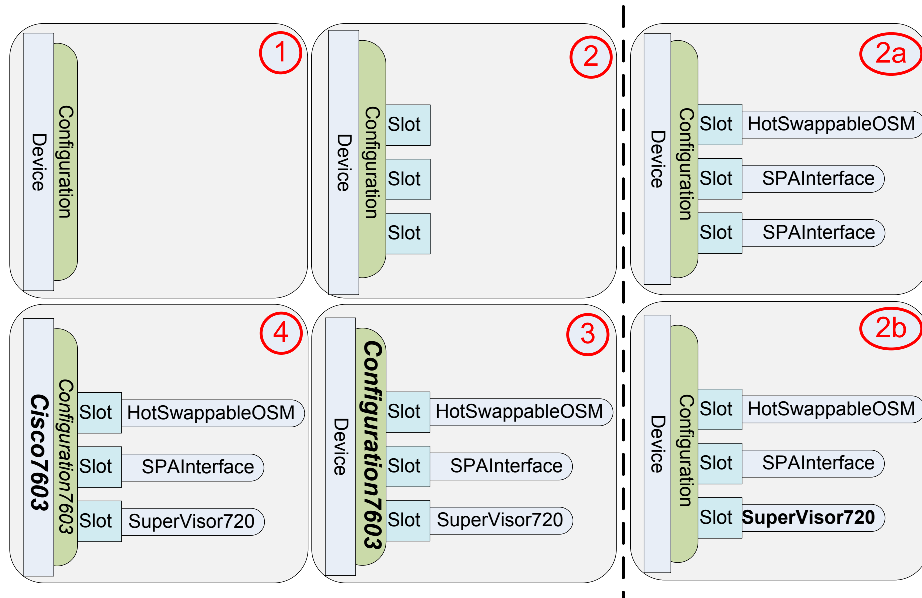

Fig. 2. Modeling a physical device in four steps (M1 layer)

2. *Progressive verification* (*challenges (1), (4)*). Even with an incomplete model, the DSL development environment must provide means by verifying constraints. For example during the modeling phase in *step 2a* the DSL user wants to debug his domain model in the aforementioned scenario, where the inconsistency occurs after adding the two SPAInterface cards.
3. *Suggestions of suitable domain concepts to be used* (*challenge (4)*). The DSL development framework should be able to dynamically classify the elements of the DSL model according to class descriptions in the DSL metamodel. DSL users normally start the modeling with general concepts, e.g. with Device or Configuration in the aforementioned scenario. The framework suggests the refinement of elements to the most suitable ones, e.g. to Configuration7603 or Cisco7603 (*step 3, 4*). Later this might be important, for example, to generate the most specific and suitable source code of the domain model. Further, such classifications together with explanation help novice DSL users to understand how to use the DSL.
4. *Debugging (reasoning explanation)* (*challenges (1), (4)*). Debugging is a requirement for the success of the DSL [8]. DSL users want to debug their domain models to find errors inside them and to get an explanation how to correct the model. They want to have information about consequences of applying given domain constructs. In the scenario, DSL users want to know they have to replace an SPAInterface card with a Supervisor720 card (*step 2b*).

5. *Different ways of describing constructs (syntactic sugar)* (*challenge (4)*). DSL users are not always familiar with all specific concepts a DSL provides. In the aforementioned scenario, for example, DSL users do not have the complete knowledge of the Configuration7603. Thus, they use an alternative way to describe an instance of this concept (*step 2 and 3*). Providing such alternative ways of writing DSL models might improve productivity.

3 Domain-Specific Modeling and Ontologies

The DSL development process may be divided into the following phases [9]: decision, analysis, design and implementation. Usually, ontologies can be employed as a design time enhancement in the analysis and design phases and as runtime enhancement. This section illustrates the suitability of ontologies for each of these phases.

In the analysis phase, the problem domain is examined. Beside different domain analysis approaches (e.g. Feature Oriented Domain Analysis (FODA) or others [9]) ontologies can be used in the early phases of the development during the domain analysis. The process of ontology-based domain engineering [10] can be used to develop a vocabulary for the specification of a domain that can be translated into different formats for forthcoming phases. Existing ontologies may also provide a starting point for domain analysis without the need to start from scratch.

In the design phase, the domain model produced in the analysis phase is used to define the metamodel of the DSL. MOF-like metametamodels usually describe the metamodels. The semantics of MOF-based metamodels is limited compared to the ones of ontologies, i.e., ontology languages are more expressive than MOF-like technical spaces and provide a better support for reasoning than MOF-based languages [11].

In the implementation phase, interpretation and compilation of the DSL are addressed. Here, DSL compilers or interpreters may implement calls to reasoner APIs to enable services like reasoning explanation, instance checking, consistency checking and query answering.

To sum up, ontologies may be applied during different phases of the DSL development process. Ontology-based approaches lead to formal domain-specific models that may be exploited for a variety of services, from consistency checking [12] to semi-automatic engineering and to explanations [13].

3.1 Ontologies as conceptual models

Among ontology languages, we highlight the W3C standard OWL. OWL actually stands for a family of languages with increasing expressiveness. OWL2, the emerging new version of OWL, is more expressive and still allows for sound and complete calculi that are decidable as well as pragmatically efficient.

The capability to describe classes in many different ways and to handle incomplete knowledge distinguishes OWL from class-based modeling languages like UML class diagrams, MOF and Ecore. These OWL features increase the

expressiveness of the metamodeling language, making OWL a suitable language to formally define DSL metamodels.

We use the running example to illustrate these features. In the following list, we describe the knowledge base of the running example. The class Supervisor720 is a subclass of SupervisorCard (eq. 1). The class Card is a complete generalization of HotSwappableOSM, SPAInterfaceProcessor and SupervisorCard (eq. 2).

$$Supervisor720 \sqsubseteq SupervisorCard \quad (1)$$

$$Card \equiv HotSwappableOSM \sqcup SPAInterfaceProcessor \sqcup SupervisorCard \quad (2)$$

$$Configuration \sqsubseteq \exists \geq 1 hasSlot.Slot \sqcap \exists hasSlot.(\exists hasCard.SupervisorCard) \quad (3)$$

$$Configuration7603 \equiv \exists = 3 hasSlot.Slot \sqcap \exists hasSlot.(\exists hasCard.(HotSwappableOSM \sqcup SPAInterfaceProcessor)) \quad (4)$$

$$Cisco7603 \equiv \exists hasConfiguration.Configuration7603 \quad (5)$$

$$Configuration \sqcap Slot \sqsubseteq \bot \quad (6)$$

Addressing Constraint Definition. OWL allows describing logical restrictions over classes. For example, the class Configuration requires at least one Slot and a Slot in which a SupervisorCard is inserted (eq. 3). Moreover, class descriptions may be declared as equivalent (eq. 2, 4, 5), e.g., the class Configuration7603 is equivalent to an anonymous class that has exactly 3 Slots, one of them with either a HotSwappableOSM or SPAInterfaceProcessor inserted (eq. 4). It means that individuals of the class Configuration7603 belong to the anonymous class and *vice versa*.

Addressing Progressive Verification. These logical restrictions can be verified progressively. For example, supposing that a DSL user adds a instance of the class Configuration7603. The reasoner infers that this instance has exactly 3 slots. If there are less than 3 slots in the model, the reasoner will trow an inconsistency. As soon as the DSL user adds 3 slots, the model becomes consistent again. Although the DSL user did not associate the 3 slots with the Configuration7603, the reasoner infers there is a relation. When the DSL user associates the 3 slots with another instance of the class Configuration, the reasoner points the inconsistency again.

As illustrated, OWL provides various means for expressing classes: enumeration of individuals, property restrictions (eq. 3), intersections of classes (eq. 6), unions of class descriptions (eq. 2), complements of a class description or combinations of any of those means.

Addressing Debugging. Reasoning services may be combined with non-standard reasoning services to provide reasoning explanation. The goal is to identify minimal and sufficient sets of axioms that explain relationships between domain elements, i.e., to identify justifications [13].

Addressing Syntactic Sugar. By declaring two classes as equivalent, DSL designers give DSL users the possibility of modeling a concept in two different

ways. In our example, DSL users have different ways of creating instances of the concept Cisco7603 (eq. 5): by declaring it directly or by creating an Device with one Configuration7603.

Open vs. Closed World Assumption While the underlying semantics of UML-based class modeling adopts the closed world assumption, OWL adopts open world assumption by default. However, research in the field of combining description logics and logic programming [14] provides solutions to support OWL reasoning with closed world assumption. Different strategies have been explored like adopting an epistemic operator [15], already supported by the tableau-based OWL reasoner Pellet [16, 17]. Thus, it allows us to avoid the semantic clash in merging the two languages.

In this section, we have seen that Description Logics can attend as conceptual models for describing domain-specific languages. However, our intention is not to demand DSL designers to develop DSLs directly and completely as ontologies. Instead, we have an ontology-based domain-specific language framework which provides a seamless and integrated development of formal semantics within the language definition itself using some natural to use and simple to learn ontology languages, which can be used in combination with other, more familiar concrete syntaxes.

4 An Ontology-Based Framework for DSLs

In this section, we introduce our ontology-based domain-specific language framework – *OntoDSL*. After presenting the general idea of our approach in Section 4.1, we concentrate on the implementation of the new technical space in Section 4.2. We integrate the KM3 metametamodel, a simplified subset of MOF, with the OWL2 metamodel and additionally with the OCL metamodel at the M3 layer. Thus, we can provide a new technical space which allows implementing DSL metamodels with formal semantics, conditions and queries. Finally, we give an example of defining a new metamodel in *OntoDSL* concrete syntax in Section 4.3.

4.1 Overall Approach

As already mentioned in Section 1.2, Figure 1 depicts the layered architecture and provides an overview of all roles and models considered in the running example. Now we consider more technical details of the framework.

At the M3 layer, we first have a seamless integration of the MOF based metametamodel KM3, the OWL metamodel and the OCL metamodel. Having the integrated metametamodel at the M3-layer (the new technical space), DSL designers can define domain-specific languages using KM3, OWL and OCL constructs in seamless manner. They describe the static structure of DSL metamodels (e.g. using KM3 constructs), formal semantics (e.g. using OWL constructs) or operations (e.g. using OCL constructs).

DSL users may then use the developed DSL with additional benefits. Results are domain models (M1-layer). Having formal semantics of the DSL, consistency

checking is available. Furthermore, the execution of operations in the context of a model element is available. These operations call reasoning services which work on ontologies constituted from the DSL metamodel and the domain models DSL users have created.

4.2 Implementation

In this subsection we consider the implementation of *OntoDSL*. We mainly focus on how we build the abstract syntax of the new technical space and how we developed the concrete syntax. Both, the abstract syntax, represented as metamodel and the textual concrete syntax can be downloaded from our project website `http://ontodsl.semanticsoftware.eu`.

Abstract Syntax The core of our technical space (our M3 metametamodel) consists of KM3 [18], OCL [19] and OWL2 [20]. We use our metamodel integration approach presented in [21] and [22] to combine the different metamodels. The result is the integrated metametamodel at the M3 layer which describes the abstract syntax of our technical space.

The *OntoDSL* metamodel provides all classes of the KM3 metametamodel, the OCL and OWL metamodel. It contains different adapter classes to integrate, for example, *OWL class* with *KM3 class*, *OWL Object Property* with *KM3 reference attribute* or *OWL Data Property* with *KM3 simple attribute*. Thus, we build a bridge between the different languages. Furthermore, we define that classes can contain operations. A new operation class in the integrated metametamodel is associated to classes for OCL operation definitions of the OCL metamodel.

Overall, with the new abstract syntax we describe all aspects DSL designers can use to define metamodels. To provide reasoning based on domain metamodels and models we implemented several transformations, e.g. using ATL [23], that translate the metamodels and models to pure OWL ontologies. We use the Pellet reasoner [16] to provide OWL reasoning services.

Concrete Syntax Listing 4.1 in the section below gives an example of using the concrete syntax to define a metamodel.

The concrete syntax of *OntoDSL* is based on KM3. The motivation is that DSL designers should use the Java-like KM3 syntax as much as they can. To take benefit from OWL or OCL, they should be able to annotate elements of their DSL metamodel in a textual manner. Hence, we extend the grammar of the KM3 concrete syntax by new non-terminals which are defined in grammars of a textual OWL2 concrete syntax or of a textual OCL concrete syntax.

For example, we are able to annotate KM3 classes with OWL class axioms, KM3 reference attributes with OWL Object Property Axioms or implement OCL operations within KM3 classes. We have developed our own *OWL 2 natural style syntax* which is an adaptation of the OWL Manchester Syntax [24] to get a natural controlled language for coding OWL2 ontologies. As OCL concrete syntax, we take the one from [19].

Overall, we have a grammar which consists of rules to produce KM3 statements in combination with textual OWL annotations and embedded OCL operation definitions.

We solve the mapping between abstract and concrete syntax by using the Eclipse component TCS [25] for textual concrete syntax specification of DSLs.

4.3 Example in Concrete Syntax

In Listing 4.1, we see an excerpt of an M2 metamodel that is created by a DSL designer using the new integrated metametamodel. Using the KM3 syntax, he defines that a Device has Configurations, a Cisco7603 is a specialization of Device, each Configuration has Slots and in each Slot one to many cards can be plugged in.

Furthermore, the DSL designer defines some formal semantics using the OWL natural style concrete syntax, which is integrated with the existing KM3 syntax. In Listing 4.1, he states that every Cisco7603 device has at least one Configuration7603. A Configuration is a Configuration7603 if and only if it has exactly three slots in which either a HotSwappableOSM card or a SPAinterfaceProcessors card is plugged in.

Addressing Suggestions. At the end of Listing 4.1, we see the definition of the class Thing, which is in OWL the superclass of all classes. Here, using OCL syntax, we define a new operation getSpecificSubClasses(). Because of inheritance from the superclass Thing, this operation can be executed in the context of all classes. Having an operation called getSpecificSubClasses(), the DSL designer can support the DSL user with suggestions of suitable domain concepts. In this case, the DSL user has to call the operation in the context of a model element and gets as feedback the suggestion to refine the type of the current model element (*requirement (3)*).

Listing 4.1. Example of defining an M2 metamodel

```
1  class Device {
2    reference hasConfiguration [1-*]: Configuration;
3  }
4
5  class Cisco7603 extends Device, equivalentWith restrictionOn
       hasConfiguration
6  with min 1 Configuration7603 { }
7
8  class Configuration extends IntersectionOf(restrictionOn hasSlot with
       min 1
9  Slot, restrictionOn hasSlot with some restrictionOn hasCard with some
10 SuperVisor720){
11   reference hasSlot : Slot;
12 }
13
14 class Configuration7603 extends Configuration, equivalentWith
15 IntersectionOf(restrictionOn hasSlot with exactly 3 Slot, restrictionOn
       hasSlot
16 with some restrictionOn hasCard with some UnionOf(HotSwappableOSM,
17 SPAinterfaceProcessors) { }
18
19 class Slot {
20   reference hasCard [1-*]: Card;
21 }
22
```

```
23  class Thing {
24    query getSpecificSubClasses (): Set(Thing)
25    = self.owlAllTypes()
26  }
```

5 Analysis of the Approach

In this section, we establish the viability of our approach by a proof of concept evaluation. We analyze the approach with respect to the requirements of Section 2.1.

To address formal semantics and constraints (*requirement (1)*), we integrated the EMOF based metametamodel KM3 and concrete syntax with OWL, allowing for a formal and logical representation of the solution domain. Thus, DSL designers count on an expressive language that allows for modeling logical constraints over DSL metamodels (*requirement (1)*). Reasoners check the consistency of metamodels and constraints and debugging services clarify the inferences (*requirement (4)*).

Formal semantics enable the usage of reasoning services to help DSL users to find appropriate constructs based on DSL models (*requirement (3)*). For example, DSL users may get suggestions of devices to be used in their DSL models based on the configuration of the device.

The expressiveness of OWL enables DSL designers to define classes and properties as equivalent. DSL designers my use this functionality to provide DSL users with different means for declaring objects (*requirement (5)*). For example, a DSL user may describe a Cisco 7603 device in two different ways: by creating an instance of class Device with a configuration with three slots and a supervisor card in one slot; or by creating an instance of class Cisco7603.

The nature of the logical restrictions allowed by OWL enables progressive evaluation of DSL model consistency (*requirement (2)*). For example, a DSL user may drag a new configuration into a DSL model with already two supervisor cards. A configuration requires at least one supervisor card. Even though it is not asserted that any of the supervisor cards are part of the new configuration, the reasoner assumes that at least one of the cards is related with this configuration.

DSL users call OCL-like queries defined by DSL designers with the DSL metamodel to query objects in DSL models (*requirement (3)*). These queries are the interface between DSL users and reasoning services. For example, a DSL user may use a reasoning service which is implemented as query defined in the DSL metamodel and queries all classes that describe an object in the DSL model.

While solutions provided by DSL development environments for teaching DSL users are usually limited to help files and creation of the example models, we have an interactive assisted solution by suggesting concepts and explaining inferences (*requirement (3)*). Nevertheless, addressing the aforementioned requirements lead us to new challenges as well as it demands to consider tradeoffs between expressiveness and completeness/soundness, expressiveness and user complexity.

OWL is a logical language and it requires logical expertise of DSL designers. On the other side, the usage of OWL is encapsulated from DSL users. They only use operations to invoke different reasoning services which work on ontologies. Our approach extends the KM3 vocabulary with a controlled natural language as OWL textual concrete syntax to smooth the usage of logical assertions. Nevertheless, the usability of our approach for DSL designers and DSL users still need further evaluations.

6 Related Work

In the following, we group related approaches into two categories: approaches with formal semantics and approaches for model-based domain-specific language development.

Among approaches with formal semantics, one can use languages like F-Logic or Alloy to formally describe models. In [26], a transformation of UML+OCL to Alloy is proposed to exploit analysis capabilities of the Alloy Analyzer [27]. In [28], a reasoning environment for OWL is presented, where the OWL ontology is transformed to Alloy. Both approaches show how Alloy can be adopted for consistency checking of UML models or OWL ontologies. F-Logic is a further prominent rule language that combines logical formulas with object oriented and frame-based description features. Different works (e.g. [29, 30]) have explored the usage of F-Logic to describe configurations of devices or the semantics of MOF models.

The integration in the cases cited above is achieved by transforming MOF models into a knowledge representation language (Alloy or F-logic). Thus, the expressiveness available for DSL designers is limited to MOF/OCL. Our approach extends these approaches by enabling DSL designers to specify class descriptions *à la* OWL together with MOF/OCL, increasing expressiveness.

Examples of model-based DSL development environments are MetaEdit+ [31], XMF (eXecutable modelling framework) [32], Generic Modeling Environment (GME) [33] and ATLAS Model Management Architecture (AMMA) [34]. These approaches are aligned with the OMG four-layer metamodel architecture and provide support to OCL-like languages (like in XMF GME and AMMA) for specifying queries and constraints. Our approach adds value on them by providing a logic-based approach to define formal semantics of DSLs. The logic-based approach allows us to provide functionalities based on Description Logics constructs like equivalence, class descriptions to DSL users. Concretely, it allows us to support guidance and suggestions to DSL users.

7 Conclusion

In this paper, we presented an approach how to address major challenges in the field of domain-specific languages with OWL ontologies and automated reasoning. The new technical space integrates EMOF and OWL at the M3-layer and

enables applications of reasoning to help DSL designers and DSL users through the development and usage of DSLs. DSL designers profit by formal representations, an expressive language and constraint analysis. DSL users profit by progressive verification, debugging support and assisted programming. The approach has been used and tested in the telecommunication domain under EU STReP MOST. Future work into this direction would investigate ways of extending concrete syntaxes to support the flexibility of OWL.

Acknowledgement. We like to thank Krzysztof Miksa from Comarch for providing the use cases. Further, we like to thank Prof. Dr. Juergen Ebert for comments and improvement remarks. This work is supported by CAPES Brazil and EU STReP-216691 MOST.

References

1. Kelly, S., Tolvanen, J.: Domain-Specific Modeling. John Wiley & Sons (2007)
2. Gray, J., Fisher, K., Consel, C., Karsai, G., Mernik, M., Tolvanen, J.P.: Panel - DSLs: the good, the bad, and the ugly. In: OOPSLA Companion '08, ACM (2008)
3. McGuinness, D.L., van Harmelen, F.: OWL Web Ontology Language overview (February 2004) Available at `http://www.w3.org/TR/2004/REC-owl-features-20040210/`.
4. Tairas, R., Mernik, M., Gray, J.: Using ontologies in the domain analysis of domain-specific languages. In: Proceedings of the 1st International Workshop on Transforming and Weaving Ontologies in Model Driven Engineering 2008. Volume 395 of CEUR Workshop Proceedings., CEUR-WS.org (2008)
5. Guizzardi, G., Pires, L.F., van Sinderen, M.: Ontology-based evaluation and design of domain-specific visual modeling languages. In: Proceedings of the 14th International Conference on Information Systems Development, Springer (2005)
6. Brauer, M., Lochmann, H.: An Ontology for Software Models and Its Practical Implications for Semantic Web Reasoning. LNCS **5021** (2008) 34–48
7. Miksa, K., Kasztelnik, M.: Definition of the case study requirements. Deliverable ICT216691/CMR/WP5-D1/D/PU/b1, Comarch (2008) MOST Project.
8. Gilmore, D.J.: Expert programming knowledge : a strategic approach. In: Psychology of Programming. Academic Press
9. Mernik, M., Sloane, A.: When and how to develop domain-specific languages. ACM Computing Surveys (CSUR) **37**(4) (2005) 316–344
10. de Almeida Falbo, R., Guizzardi, G., Duarte, K.: An ontological approach to domain engineering. In: In Proc. of SEKE 2002, ACM Press New York (2002) 351–358
11. Happel, H.J., Seedorf, S.: Applications of ontologies in software engineering. In: Proc. 2nd International Workshop on Semantic Web Enabled Software Engineering (SWESE 2006), November 6th, 2006, Athens, USA
12. Van Der Straeten, R., Mens, T., Simmonds, J., Jonckers, V.: Using Description Logic to Maintain Consistency between UML Models. In: In Proc. of UML 2003 - The Unified Modeling Language. Volume 2863 of LNCS. 326–340
13. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: In IJCAI International Joint Conference on Artificial Intelligence, Morgan Kaufmann (2003) 355–362

14. Motik, B., Horrocks, I., Rosati, R., Sattler, U.: Can owl and logic programming live together happily ever after? In: In Proc. ISWC-2006. Volume 4273 of LNCS., Springer (2006) 501–514
15. Donini, F.M., Nardi, D., Rosati, R.: Description logics of minimal knowledge and negation as failure. ACM Trans. Comput. Logic **3**(2) (2002) 177–225
16. Parsia, B., Sirin, E.: Pellet: An OWL DL Reasoner. In: Proc. of the 2004 International Workshop on Description Logics (DL2004). Volume 104 of CEUR Workshop Proceedings. (2004)
17. Katz, Y., Parsia, B.: Towards a Nonmonotonic Extension to OWL. In: Proceedings of the OWLED 2005, Galway, Ireland, November 11-12, 2005. Volume 188 of CEUR Workshop Proceedings. (2005)
18. Jouault, F., Bezivin, J.: KM3: A DSL for Metamodel Specification. In: In Proc. of 8th IFIP WG 6.1 International Conference, FMOODS 2006
19. OMG: Object Constraint Language Specification, version 2.0. Object Modeling Group. (June 2005)
20. Motik, B., Patel-Schneider, P.F., Horrocks, I.: OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. `http://www.w3.org/TR/2009/WD-owl2-syntax-20090421/` (April 2009)
21. Silva Parreiras, F., Staab, S., Winter, A.: TwoUse: Integrating UML models and OWL ontologies. Technical Report 16/2007, University of Koblenz-Landau Available at `http://isweb.uni-koblenz.de/Projects/twouse/tr162007.pdf`.
22. Parreiras, F.S., Walter, T.: Report on the combined metamodel. Deliverable ICT216691/UoKL/WP1-D1.1/D/PU/a1, University of Koblenz-Landau (2008) MOST Project.
23. Jouault, F., Kurtev, I.: Transforming Models with ATL. In: MoDELS Satellite Events. Volume 3844 of LNCS., Springer (2005) 128–138
24. Horridge, M., Patel-Schneider, P.: Manchester syntax for OWL 1.1. In: International Workshop OWL: Experiences and Directions (OWLED08). (2008)
25. Jouault, F., Bézivin, J., Kurtev, I.: TCS:: a DSL for the specification of textual concrete syntaxes in model engineering. In: Proceedings of the GPCE 2006, ACM Press (2006) 249–254
26. Anastasakis, K., Bordbar, B., Georg, G., Ray, I.: UML2Alloy: A challenging model transformation. Lecture Notes in Computer Science **4735** (2007) 436
27. Jackson, D.: Software Abstractions: logic, language, and analysis. The MIT Press (2006)
28. Wang, H., Dong, J., Sun, J., Sun, J.: Reasoning support for Semantic Web ontology family languages using Alloy. Multiagent and Grid Systems **2**(4) (2006) 455–471
29. Sure, Y., Angele, J., Staab, S.: OntoEdit: Guiding ontology development by methodology and inferencing. Lecture notes in computer science 1205–1222
30. Gerber, A., Lawley, M., Raymond, K., Steel, J., Wood, A.: Transformation: The missing link of MDA. Lecture Notes in Computer Science (2002) 90–105
31. Kelly, S., Lyytinen, K., Rossi, M.: MetaEdit+: A Fully Configurable Multi-User and Multi-Tool CASE and CAME Environment, Springer (1996) 1–21
32. Clark, T., Sammut, P., Willans, J.: Applied Metamodelling: a Foundation for Language Driven Development (2nd Edition). Ceteva (2008)
33. Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason, C., Nordstrom, G., Sprinkle, J., Volgyesi, P.: The Generic Modeling Environment. In: Proceedings of the IEEE Workshop on Intelligent Signal Processing (WISP'2001), IEEE (2001)
34. Bézivin, J., Jouault, F., Kurtev, I., Valduriez, P.: Model-Based DSL Frameworks. In: OOPSLA, ACM (2006) 22–26