Towards Semantic Modeling of Network Physical Devices*

Krzysztof Miksa[1], Marek Kasztelnik[1], Pawel Sabina[1], Tobias Walter[2]

[1] Comarch SA
Al. Jana Pawla II 39A, Krakow, Poland
{krzysztof.miksa, marek.kasztelnik, pawel.sabina}@comarch.com
[2] ISWeb — Information Systems and Semantic Web,
Institute for Computer Science, University of Koblenz-Landau
Universitätsstrasse 1, Koblenz 56070, Germany
walter@uni-koblenz.de

Abstract. One of the challenges faced by network management systems is the increasing need for consistent management of physical network equipment. We propose a solution where equipment is modelled using a dedicated Domain Specific Language (DSL) enriched with the power of logic-based reasoning services. This enables us to define a rich layer of semantics on top of the structural description of the devices. This way, the configuration related constraints are expressed declaratively, in a platform independent manner, and are managed in an integrated way with the structural model. The information kept in the model can then be used on runtime to give guidance to the system user.

1 Introduction

One of the challenges faced by Next Generation Operation Support Systems (OSS) [1] is the increasing need for consistent management of physical network equipment. In large companies the time consumed by maintaining thousands of devices and finding solutions to possible problems is constantly on the rise. State-of-the-art technologies enable vendor independent equipment type identification and access to the attributes of the component types. Furthermore, current solutions often provide the user with convenient graphical modelling of the physical elements structures, but are usually unable to provide consistent support to the user, by answering questions that involve sophisticated configuration related constraints.

In our approach, we propose a solution where equipment is modelled using a dedicated domain specific language enriched with the power of logic-based reasoning services. This enables us to define a rich layer of semantics on top of the structural description of the devices. This way, the configuration related

* This research has been co-funded by the European Commission and by the Swiss Federal Office for Education and Science within the 7th Framework Programme project MOST number 216691

constraints are expressed declaratively, in a platform independent manner, and are managed in an integrated way with the structural model. The information kept in the model can then be used on runtime to give guidance to the system user.

2 Problem description

Myriads of device types and their configurations can make user's everyday work a nightmare. For example, each time a card in some device is broken, the system operator faces questions like, "what are the possible replacements for that card, are some options better then others?" On the other hand, a system analyst planning new services on a particular device wants to know what components he can use with that device, if possible, from those available in the company's warehouse.

Similar questions may also arise while integrating manually maintained repositories of physical network equipment. In such cases, automatic checking of device configuration correctness or even finding the exact device type using only information about its configuration, would surely improve the integrity and correctness of existing data.

Let's take an example of a usual situation in telecommunication companies when one of the physical device cards is broken or not supported any longer and requires replacement. Figure 1 presents an example configuration of the `Cisco 7603 chassis`. It contains two cards. The card in `slot 1` is a `supervisor card`, required by the device to work properly. In `slot 2`, a `backup supervisor card` is inserted.

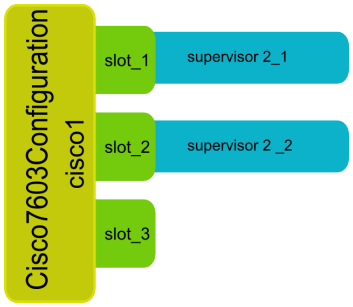

Fig. 1. Example instance of Cisco 7603 configuration in PDIDSL

Let's suppose that the main supervisor card is broken and requires replacement. The person responsible for this device receives a notification about the problem and begins to resolve it. Current solutions of OSS systems require deep knowledge about every sub-component of the physical device (what kind of cards can be used as a replacement for a broken card, what kind of constraints a particular card has, etc.).

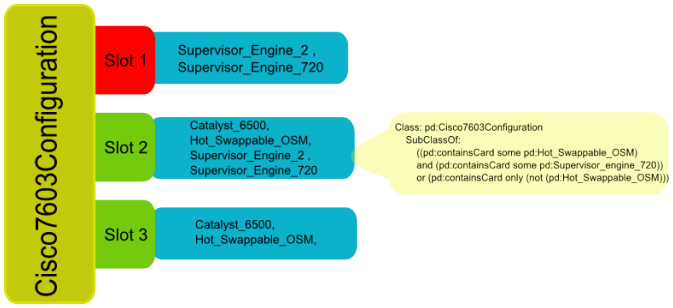

Fig. 2. Cisco 7603 configuration in PDDSL

To limit the effort required to solve such problems, we designed a DSL that describes the structure of the physical device and stores information about a possible connection between physical device elements. In our example of a simplified Cisco 7603 from Figure 2, we specify that it has three slots and that the first slot is required (marked red in the diagram). The possible or required cards are indicated in blue rectangles next to the respective slots. The description can be enriched with the additional ontology axioms. One of the constraints that occur, is that the `Hot_Swappable_OSM` card require a specific supervisor engine, namely `Supervisor_engine_720` in the Cisco 7603 configuration.

As shown, there is clearly a need for tools providing advanced support helping users make correct decisions, tools based on knowledge and semantics, able to reason and bring meaningful answers for user questions. The problems to address are much more broad than simply suggesting the replacements of a broken card:

1. Network planning - what components can I use with a given configuration of a device to build my service?
2. Consistency checks - is my configuration valid?
3. Data quality - what is the exact type of device, given it's configuration?
4. Explanations and guidance: Why the configuration is invalid? How can I fix it?

Such tools, guiding and supporting users through tedious tasks by answering the questions mentioned, would generate substantial profit, and reduce the number of potential errors in device configurations. It would also improve productivity, and mitigate the time consumed studying the technical details of a device's documentation.

3 Integrating models and ontologies

To achieve these goals, we follow an approach proposed by our partner in the MOST project, the University of Koblenz-Landau [3], based on the idea of integration of models and ontologies [2]. Modelling physical devices is a perfect candidate to evaluate this idea, and for a reason. Firstly, a network of physical

[3] http://isweb.uni-koblenz.de

devices can easily be described using a limited number of concepts that makes it a subject of Physical Device Domain Specific Language (cf. section 4). On the other hand, possible device configurations and connections build some kind of knowledge base, which would be hard to express using structural models, but are ideal for representing as an ontology.

Existing works that compare the ontological and metamodelling technology spaces [3] reveal that hybrid approaches present significant advantages over technologies stemming purely from metamodelling space, such as OCL. In the scope of Comarch [4] case study in the MOST project [5], our work goes toward extending the expressiveness of the domain specific language for the description of the physical device structure. This is achieved through integration of Domain Specific Languages with ontologies, and thus opening new spaces for modelling structure and semantics together. The integrated models remain easy to edit, process, and reasoned about using existing tools and approaches, being a structural and semantic model at the same time.

4 Prototype solution

A prototype implementation of a physical devices modelling tool was developed using Eclipse Modelling Framework [6]. The goal of the tool was to enable modelling physical devices in a DSL and, at the same time, take advantage of formal semantics and expressivity of OWL. In this early prototype, the integration of the domains of models and ontologies was achieved by model transformations specified in QVT Operational [4].

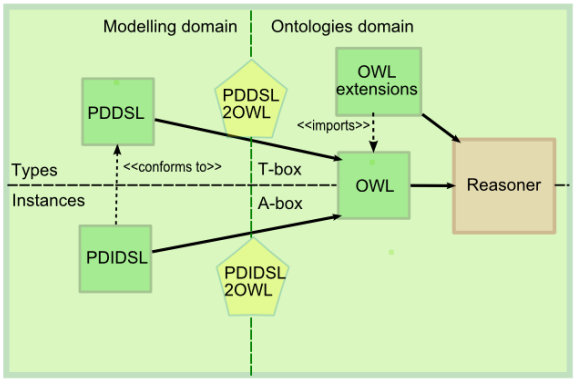

Fig. 3. Conceptual architecture of the prototype

As shown in Figure 3, the prototype consists of the following artefacts:

[4] http://www.comarch.com
[5] http://www.most-project.eu
[6] http://www.eclipse.org/modeling/emf/

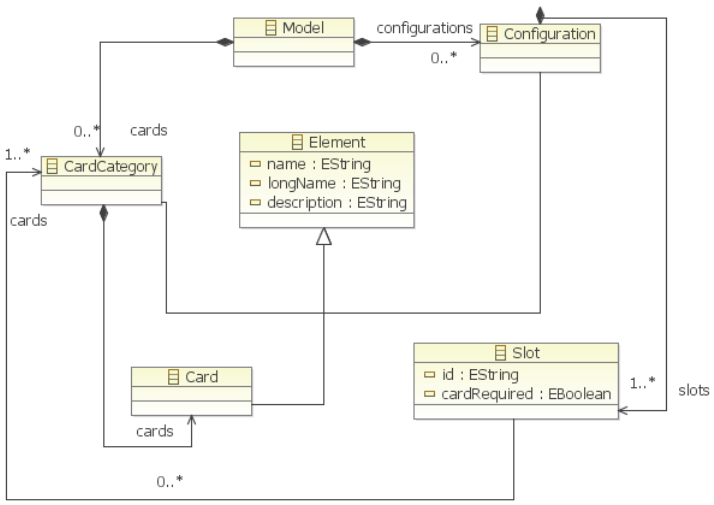

Fig. 4. PDDSL metamodel excerpt

PDDSL The *Physical Devices Domain Specific Language* (PDDSL) enables specification of possible configurations of device models. The language uses concepts related to the structure of devices: *Configuration*, *Slot*, *Card*. Figure 2 gives an example of a model expressed in PDDSL. An excerpt of the metamodel of PDDSL is given in Figure 4. The language has also a concrete textual syntax which, for brevity, is not described in this document. Also, for simplicity, we consider *Configuration* concept as the direct representation of a physical device type, while in real scenario a device could have multiple alternative configurations

PDIDSL The *Physical Devices Instances Domain Specific Language* (PDIDSL) enables definitions of concrete instances of devices that conform to the PDDSL specifications. An example of a PDIDSL model is given in Figure 1. As depicted in Figure 3 PDIDSL models conform to a metamodel which is specified with PDDSL. However, since PDDSL is not a metamodelling language we need an additional step, where some PDDSL model is mapped to an Ecore metamodel. This transformation is not given here for the sake of brievity. An excerpt of the metamodel of PDIDSL is given in Figure 5.

OWL We use *OWL Manchester Syntax* [5] to represent the OWL ontology. Manchester Syntax was chosen because there already exists an Ecore metamodel for it, and more importantly, generated OWL models can be serialized into valid Manchester Syntax files without the need for another transformation (as in case of Ontology Data Model) using EMFText tool [7].

[7] http://www.emftext.org

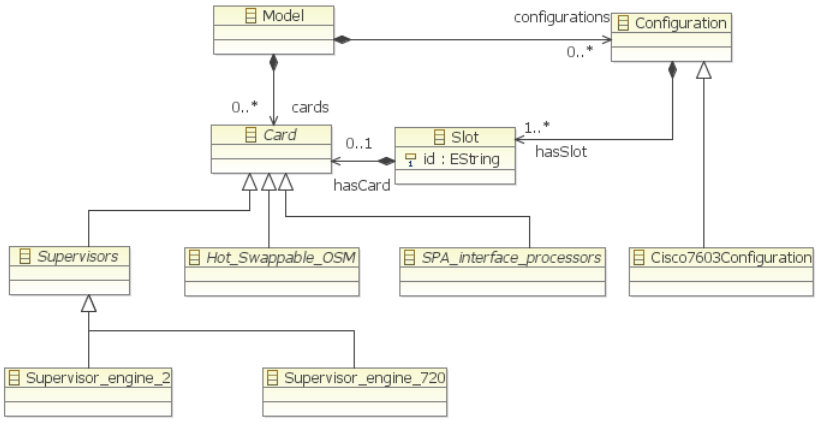

Fig. 5. PDIDSL metamodel excerpt

OWL extensions Additional knowledge related to physical devices is defined in OWL Manchester Syntax. Section 4.4 describes an example of such an ontology.

Reasoner We use Pellet to perform reasoning in the prototype. Examples for reasoning are given in Section 4.5.

PDDSL2OWL A QVT operational transformation transforms PDDSL models into an OWL T-box. It is further described in Secion 4.1.

PDIDSL2OWL A QVT operational transformation transforms PDIDSL models into an OWL A-box. It is further described in Secion 4.2.

In the modeling domain the languages conceptually form the following hierarchy:

M3 level Ecore metametamodel

M2 level PDDSL metamodel defining the language needed to describe possible configurations of devices

M1 level PDDSL models describing the possible configurations of a device. PDIDSL metamodel defining the language needed to describe concrete configurations of devices

M0 level PDIDSL model represents concrete configurations

4.1 Transforming type model

The goal of the PDDSL2OWL transformation is to extract the OWL T-box from the PDDSL models. The transformation maps the concepts of the PDDSL into OWL classes and properties and, equally important, specifies formal semantics of the concepts from PDDSL (e.g. formalization of configuration constraints).

Figure 6 shows an excerpt of the transformation where an object property restriction is generated for the `Configuration` class. Each of the required slots

```
-- configuration is mapped to subclass of Configuration
-- with equivalency axiom
mapping Configuration::toConfiguration() : OWL::Class {
    equivalentClassesDescriptions += object Conjunction {
        primaries += object ClassAtomic{clazz := ConfigurationClass};
        -- condition on required cards
        primaries += self.slots [cardRequired = true]
            -> map toSlotRequiredRestriction();
        -- Another restrictions (not listed here):
        ...
    }
}

-- required cards - configuration has _some_ of the cards
-- specified in required slots
mapping Slot::toSlotRequiredRestriction() : OWL::NestedDescription {
    description := object ObjectPropertySome {
     feature := hasSlotProperty;
        primary := object NestedDescription {
            description := object Conjunction {
                primaries +=  object ObjectPropertySome {
                    feature := hasCardProperty;
                    primary := object ClassAtomic{
                        clazz := self.cards
                            -> first().map toCardCategory()}
                };
                --slot number
                primaries += object ObjectPropertyValue {
                    feature := idProperty;
                    _literal := object IntegerLiteral {
                        value := self.id.toInteger()
                    }
                }
            }
        }
    }
}
```

Fig. 6. Excerpt of T-box generation in PDDSL2OWL transformation

in the `Configuration` is mapped to an `ObjectPropertySome` restriction on the `hasSlot` property. The property is restricted to a conjunction of restrictions on the `hasCard` property. Each of the restrictions in the conjunction specifies the required card and value restriction on the `id` data property indicating the slot number.

4.2 Transforming instance model

The PDIDSL2OWL transformation takes as input a model containing the instances of physical devices expressed in PDIDSL and extracts the respective OWL A-box. The transforation updates the ontology T-box produced by the PDDSL2OWL transformation by adding individuals along with the respective facts assertions. Figure 7 shows an excerpt from the transformation where an instance of `Configuration` is transformed into an OWL individual. The type of the individual is set to the corresponding OWL class, representing the configuration, and the respective slots are related by the `hasSlot` property.

```
mapping Configuration::toConfigurationIndividual(): OWL::Individual {
    iri := self.name;
    types += object ClassAtomic {
        clazz := classes[iri = self.metaClassName()]
            ->  asSequence() -> first();
    };

self.hasSlot -> forEach (i) {
        facts += object ObjectPropertyFact{
            objectProperty := hasSlotProperty;
            individual := i.map toSlotIndividual();
        }
    };
}
```

Fig. 7. Excerpt of A-box generation in PDIDSL2OWL transformation

4.3 Closing the world

The reasoning tasks performed on models, such as consistency checking, often require non-monotonic reasoning [6] (e.g. Close World Assumption). In contrast, OWL adopts Open World Assumption. Therefore it is necessary to be able to close the knowledge explicitly, e.g. by adding the respective axioms to the ontology. which is done in our prototype automatically by PDIDSL2OWL transformation.

One of the means of closing knowledge is to state that a class is equivalent to an enumeration of all of its known individuals (domain closure). An update mapping that closes the world of any class with known individuals, is specified in Figure 8.

```
mapping inout OWL::Class::updateClass() {
    self.equivalentClassesDescriptions += object IndividualsAtomic {
        individuals += PDmodel.objects() [metaClassName() = self.iri]
            .resolve().oclAsType(OWL::Individual);
    }
}
```

Fig. 8. Inout mapping closing an OWL class

4.4 Enriching with additional statements

The prototype allows for enriching the physical devices ontology with additional axioms, not to limit the whole solution to the expressiveness of the DSLs. Currently these axioms are specified in a separate OWL file that references the extracted ontology through a namespace declaration. As shown in Figure 3, to perform the reasoning tasks both OWL files automatically generated from models and manually written extension file are needed by the reasoner. The files are kept separate in order to prevent overwriting manual changes after rerunning the transformations. Figure 9 gives an example of an extension, by specifying that `HotSwappable_OSM` cards in `Cisco7603` are only allowed with `Supervisor_engine_720`.

```
Namespace: pd <http://www.comarch.com/oss/pd.owl#>
Ontology: <http://www.comarch.com/oss/pd-ext.owl>

Class: pd:Cisco7603Configuration
    SubClassOf:
        ((pd:containsCard some pd:Hot_Swappable_OSM)
        and (pd:containsCard some pd:Supervisor_engine_720))
        or (pd:containsCard only (not (pd:Hot_Swappable_OSM)))
```

Fig. 9. Additional axioms

4.5 Reasoning with the resulting ontology

The extracted ontology together with extensions are the artefacts that constitute the input to the reasoner. The prototype allows for various types of reasoning, e.g. classification and consistency checking. We use classification to detect the exact type of a configuration individual, in order to support elaboration of partially incomplete models, which is the often case in physical devices management. Consistency checking was also successfully applied in order to detect and prevent errors in devices configurations. In the remainder of this section we provide an example of classification and consistency check in the physical devices ontology. The examples differ in A-box axioms but use the same T-box. In Figure 10 the basic concepts of the ontologies are defined: `Configuration`, `Slot`, `Card` classes. A `Configuration` may contain multiple Slots (via the `hasSlot` property), while

a `Slot` may contain only one `Card` (via the `hasSlot` property). Additionally, the slots may be identified with an `id` property. `Slot` and `Card` classes are closed by enumerating all of their individuals.

```
Class: Configuration

Class: Slot
    EquivalentTo: {cisco1_slot_3, cisco1_slot_2, cisco1_slot_1}

Class: Card
    EquivalentTo: {supervisor_2_2, HS_OSM_1, supervisor_720_1,
        supervisor_720_3, H_OSM_2, supervisor_2_1,
        supervisor_2_3, spa_1}

ObjectProperty: hasSlot
    Domain:  Configuration
    Range:  Slot
    Characteristics: InverseFunctional

ObjectProperty: hasCard
    Domain:  Slot
    Range:  Card
    Characteristics: InverseFunctional , Functional

DataProperty: id
    Domain:  Slot
    Characteristics: Functional
```

Fig. 10. Basic concepts in the generated ontology

Figure 11 shows another part of the T-box where specific types of `Cards` are defined. Each of the specific `Card` classes is closed by enumerating all of its individuals. Additionally, the respective `Card` subclasses are declared disjoint (not depicted in figure 11).

The class representing the allowed Cisco 7603 configuration is specified in Figure 12. The class is generated as equivalent to an anonymous class that brings together all of the restrictions generated from PDDSL model, i.e.:

1. Cardinality restriction on `hasSlot` property - the configuration has exactly the number of slots specified in PDDSL.
2. Restriction on the required cards - the configuration has to contain cards specified in the slots marked as required in PDDSL.
3. Restriction on the optional cards - the configuration cannot contain other cards than those specified in PDDSL.

The mappings for the first and third constraint are given in Figure 6. E.g. `toSlotRequiredRestriction` mapping generates restriction on the required cards.

```
    Class: Supervisors
        SubClassOf:  Card

    Class: Hot_Swappable_OSM
        SubClassOf:  Card
        EquivalentTo: { HS_OSM_1 , H_OSM_2 }

    Class: SPA_interface_processors
        SubClassOf:  Card
        EquivalentTo: { spa_1 }

    Class: Supervisor_engine_2
        SubClassOf:  Supervisors
        EquivalentTo: {supervisor_2_1, supervisor_2_2, supervisor_2_3}

    Class: Supervisor_engine_720
        SubClassOf:  Supervisors
        EquivalentTo: {supervisor_720_1, supervisor_720_3}
```

Fig. 11. Card types in the generated ontology

```
Class: Cisco7603Configuration
    EquivalentTo:  Configuration and
    # cardinality restriction on slots:
    hasSlot exactly 3  Slot and
    # required cards restriction:
    (hasSlot some (hasCard some Supervisors and id value 1)) and
    #optional card restriction:
    (hasSlot only (((hasCard some Supervisors and id value 1)) or
        ((hasCard some Supervisors and id value 2) or
            (hasCard some Hot_Swappable_OSM and id value 2)) or
        ((hasCard some Hot_Swappable_OSM and id value 3) or
            (hasCard some SPA_interface_processors and id value 3))))
```

Fig. 12. Allowed Cisco 7603 configuration in the generated ontology

Let us now consider the example of a configuration depicted in Figure 13. The model contains a configuration with three slots. Each of the slots contains a card. The model does not specify the specific type of the configuration (`cisco1` is an instance of the generic class `Configuration`). As a consequence, it is not clear what is the specific type of the device.

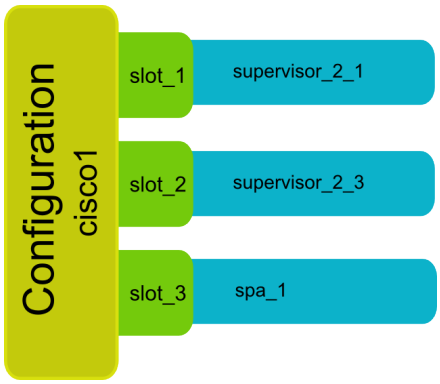

Fig. 13. Example instance of configuration in PDIDSL

In the generated ontology A-box, this model is represented by the set of individuals, specified in Figure 14. The A-box excerpts in this section omit the declarations of each card individuals for the sake of brevity, since all of them are listed in the EquivalentTo axioms of the respective classes in Figure 11.

Using the definitions from the T-box the `cisco1` individual can be classified as `Cisco7603Configuration` (see Figure 12 for the relevant constraints). Then, this inference could be used to provide guidance to the user of PDIDSL, i.e. suggest to change the type of `cisco1` to `Cisco7603Configuration`.

Let us now consider an example where we use consistency checking to prevent illegal configurations. Figure 15 depicts an example of `Cisco7603Configuration`. The configuration is invalid since the required `Supervisors` card is missing in `slot_1`.

The A-box axioms generated from this model are listed in Figure 16. Given the restrictions specified in Figure 12, the reasoner can detect the inconsistency (i.e the required card restriction does not hold). This fact then could be reported to the user by marking the inconsistent elements in the model and providing explanation of the reason of inconsistency. Moreover, employing more sophisticated reasoning services, the user could also get some guidance in form of suggestions what to change in the model in order to fix it.

5 Related Work

In [7], a transformation from UML+OCL to Alloy is proposed. The approach shows how Alloy can be adopted for consistency checking of UML models. F-

```
Individual: cisco1
    Types:  Configuration
    Facts:  hasSlot slot_1, hasSlot cslot_2, hasSlot slot_3

Individual: slot_1
    Types:  Slot
    Facts:  hasCard supervisor_2_1, id 1

Individual: slot_2
    Types:  Slot
    Facts:  hasCard supervisor_2_3, id 2

Individual: slot_3
    Types:  Slot
    Facts:  hasCard spa_1, id 3
```

Fig. 14. Individuals representing an instance of device configuration

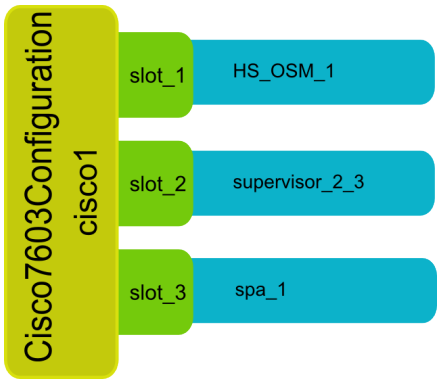

Fig. 15. Example of inconsistent instance of configuration in PDIDSL

Logic is a further prominent rule language that combines logical formulas with object oriented and frame-based description features. Different works (e.g. [8, 9]) have explored the usage of F-Logic to describe configurations of devices or the semantics of MOF models. In general, the above approaches only provide the expressiveness of MOF+OCL because its conforming models are directly transformed into a knowledge representation like Alloy or F-Logic. Our approach provides also a transformation from domain models to OWL ontologies but in addition allows enriching the OWL ontology by additional statements. Thus we can enhance the expressiveness of constraints to be checked.

[10] and [11] present combined approaches. In [11] a methodology to combine DSLs and ontology languages using metamodel integration is presented. Result is an integrated metamodel which allows building domain models and simultaneously annotating model elements with OWL statements, which are directly embedded into the model. In [10] a technical space is presented which

```
Individual: cisco1
    Types:  Cisco7603Configuration
    Facts:  hasSlot slot_1, hasSlot slot_2, hasSlot slot_3

Individual: slot_1
    Types:  Slot
    Facts:  hasCard HS_OSM_1, id 1

Individual: slot_2
    Types:  Slot
    Facts:  hasCard supervisor_2_3, id 2

Individual: slot_3
    Types:  Slot
    Facts:  hasCard spa_1, id 3
```

Fig. 16. Individuals representing an inconsistent instance of device configuration

allows developing EMOF based metamodels with integrated constraints using the ontology language OWL2.

[12] explains characteristics of configurations with description logics (DLs) that make DLs well suited for defining configurations. Our approach and the prototype comply some of the defined requirements of a configuration application. For instance, we provide based on the exported ontology inferencing and knowledge completion, explanations, inconsistency detection, error handling and some more features. Furthermore we support object-oriented modeling and extensible schemas, which is possible by using and extending the PDDSL metamodels. In [12] the representation of rules is desired, which leads us to the idea to improve our approach and prototype (e.g. taking benefits of SWRL).

6 Summary and Outlook

Even though ontologies in computer science have been used for a long time, integration with domain specific languages is a early innovation in the field of data modelling. The problems described and appearing in everyday tasks are of an abstract nature and cannot easily be solved using existing tools and approaches. Introducing integrated models containing structure and semantic information will surely be a great advantage, and will lead to the improvement of existing systems, making them more user-friendly. The presented usage scenario serves as a proof of concept for ontology enriched prmodelling. Initial results have already proved its usefulness.

However, the prototype implementation presented in this paper does not fully take advantage of the integrated approach. The ontology is fully extracted by the means of model transformation, and only then it can be extended with additional constraints. This means that models and additional axioms have to be managed separately. In our future work in the MOST project, we will investigate

how we can use the language integration approach to mitigate this shortcoming. The idea is to take profit from both approaches described in Section 5, and to allow an integrated modelling of additional OWL axioms within PDDSL models.

This would not only solve the problem of effective management of the models, but also improve the understanding of the relationship between the concepts from the two worlds. In general, it is assumed that large majority of models can be described using pure PDDSL constructs, while only limited number of uncommon cases require use of OWL extensions, thus OWL expertise would be required only for some of the users. When this extensions come into play in the integrated modelling could provide such users with understanding about the OWL meaning of PDDSL concepts without knowing the details of PDDSL2OWL transformations.

References

1. Fleck, J.: Overview of the Structure of the NGOSS Architecture . (2003)
2. Silva Parreiras, F., Staab, S., Winter, A.: TwoUse: Integrating UML Models and OWL Ontologies. Technical Report 16/2007, Universitt Koblenz-Landau, Fachbereich Informatik (2007)
3. Parreiras, F.S., Staab, S., Winter, A.: On marrying ontological and metamodeling technical spaces. In: ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, ACM (2007) 439–448
4. OMG: MOF QVT Final Adopted Specification (2005) `http://www.omg.org/docs/ptc/05-11-01.pdf`.
5. Horridge, M., Patel-Schneider, P.F.: OWL 2 Web Ontology Language Manchester Syntax. Technical report (2009) `http://www.w3.org/TR/owl2-manchester-syntax/`.
6. Damsio, C.V., Analyti, A., Antoniou, G., Wagner, G.: Supporting Open and Closed World Reasoning on the Web. In: Proceedings of 4th Workshop on Principles and Practice of Semantic Web Reasoning, Budva, Montenegro. Volume 4187 of LNCS. (2006) 149–163
7. Anastasakis, K., Bordbar, B., Georg, G., Ray, I.: UML2Alloy: A challenging model transformation. In: Proc. of 10th MODELS. Volume 4735 of LNCS., Springer (2007) 436–450
8. Sure, Y., Angele, J., Staab, S.: OntoEdit: Guiding ontology development by methodology and inferencing. Lecture notes in computer science 1205–1222
9. Gerber, A., Lawley, M., Raymond, K., Steel, J., Wood, A.: Transformation: The Missing Link of MDA. In: Proc. of First International Conference Graph Transformation. Volume 2505 of LNCS., Springer (2002) 90–105
10. Walter, T., Silva Parreiras, F., Staab, S.: OntoDSL: An Ontology Based Developement Environment for Domain-Specific Languages. In: Model Driven Engineering Languages and Systems, 12th International Conference, MODELS 2009. Volume 5795 of LNCS., Springer (2009)
11. Walter, T., Ebert, J.: Combining DSLs and Ontologies using Metamodel Integration. In: Proc. of Working Conference of Domain-Specific Languages, Oxford. Volume 5658 of LNCS., Springer (2009) 148–169
12. 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)