Ontology Driven Design of EMF Metamodels and Well-formedness Constraints [*]

Benedek Izsó
izso@mit.bme.hu

Zoltán Szatmári
szatmari@mit.bme.hu

Gábor Bergmann
bergmann@mit.bme.hu

Ákos Horváth
ahorvath@mit.bme.hu

István Ráth
rath@mit.bme.hu

Dániel Varró
varro@mit.bme.hu

Budapest University of Technology and Economics
Department of Measurement and Information Systems
H-1117 Magyar tudósok krt. 2., Budapest, Hungary

ABSTRACT

Ontologies provide high-level means for capturing requirements of systems with precise semantics and automated meta-level reasoning techniques to identify specification flaws early in the design even if certain parts of the system is underspecified. Domain-specific modeling environments effectively support domain engineers for designing the system by providing efficient means for instance-level validation of well-formedness constraints. In the current paper, we aim at a combined use of ontologies and DSM techniques where domain requirements captured in textual ontology languages like OWL2/SWRL will drive the development of a DSM environment. More specifically, we provide (i) an automated mapping from OWL2 ontologies to metamodels defined by the industry-standard EMF platform, which is extended by a (ii) mapping of requirements and constraints (captured in OWL2 and SWRL) into a textual graph pattern language efficiently evaluated by the EMF-INCQUERY incremental model query technology.

Categories and Subject Descriptors

D.2.13 [**Reusable Software**]: Domain engineering

General Terms

ontology, requirement specification

1. INTRODUCTION

Discovering and exploiting the synergies between ontologies of semantic web engineering and metamodels of domain-specific language engineering has become a hot research topic in recent years [9, 13, 20]. *Ontologies* using popular languages like OWL2 [11] or SWRL [8] are able to capture domains and its requirements in a very early phase of the design in a precise yet natural way. Ontology reasoners are optimized for concept-level (i.e., meta-level) validation (like Pellet [1] or RacerPro [12]), which can detect inconsistent specifications and classifications in an early phase of design based upon precise semantic foundations of ontologies. *Domain-specific language engineering* frameworks and tools, on the other hand, are tailored to the needs of domain engineers, thus increasing their productivity by offering functionally rich programming environments and efficient instance-level validators (like Eclipse OCL or EMF-INCQUERY) that can quickly detect violations of design rules or constraints on the instance-level.

Ontologies provide a natural formalism for capturing requirements or sketching the concepts of a system on a high level of abstraction. They can be written in several textual concrete syntaxes, including the machine processable RDF/XML, close to metamodel functional syntax or human-friendly Manchester syntax. Even controlled natural language representations are developed like the Attempto Controlled English, or the FluentEditor for OWL, which can help in language description or knowledge documentation. During the design one coherent metalanguage is used (as the SWRL is an organic extension of OWL2). Ideas can be formalized incrementally as they come thanks to ontology's open world assumption. Distributed design is also supported, as unification and consistency check can be done automatically with the help of reasoners.

DSM tools can be efficiently used by domain experts for the detailed design of the system. EMF [16] became the de facto standard in the industry, as it provides class diagram like structural description language with high level tool support. Instance data can be created and manipulated with easy to use graphical editors (that builds on GMF, GEF or Graphiti) or textual editors with Imp, Xtext, EMFText. Consistency requirements can be captured in OCL or graph pattern language, which can be continuously evaluated, and errors can be displayed by marking inconsistent elements.

The overall motivation behind our research is to develop a domain-specific modeling environment, with the combination of OWL and EMF based tools to benefit from each world. More specifically, we argue that (i) domain specific language engineering can be efficiently supported when starting from semantic web technologies, and (ii) certain instance-level validation tasks can be run on the domain specific model by an efficient, incremental model query framework, especially when the underlying knowledge base is changing or evolving.

The main contribution of the paper is to propose a method for deriving initial versions of EMF based DSM environments from ontologies describing the domain. Thus we provide an automated

[*]This work was partially supported by the CERTIMOT Research Project, the MOGENTES Research Project, the TÁMOP - 4.2.2.B-10/1 2010-0009 grant and János Bolyai Scholarship .

OCL'12 September 30 2012, Innsbruck, Austria

mapping from standard OWL2 and SWRL descriptions (capturing high-level requirements of a domain) to EMF metamodels and graph patterns captured in the textual IQPL (IncQuery Pattern Language) language [4]. This target language was chosen (instead of OCL), as both SWRL and IQPL are graph pattern based languages, thus we need to bridge a smaller semantic gap between the two languages. Furthermore we found the SWRL to OCL mapping problematic, as in OCL the results are instances of a context element (1-ary), while in SWRL and IQPL multiple pattern variables can be used, allowing n-ary result set elements.

Our motivating example uses a domain-specific model of a railway system to demonstrate what requirements can be captured in ontology, and how can they be translated to EMF, to allow efficient closed world validation during the evolution of instance models.

The rest of the paper is structured as follows. First, we give an overview of the main phases and artifacts of the mapping process in Sec. 2. Sec. 3 presents a brief introduction to the source and target technologies and introduces the running example of the paper. The detailed description of the mapping is given in Sec. 4. Finally, Sec. 5 discusses related work and Sec. 6 concludes our paper.

2. OVERVIEW OF THE MAPPING

The conceptual and technological overview of the entire mapping is provided in Fig. 1. Formulating textual requirement specifications in ontology is described in previous papers [6] [7], so first, we assume that domain engineers capture domain-specific knowledge and requirements using the SWRL extended OWL2 ontology language. In this phase ontology can be edited with Protégé [2] and reasoners (like Pellet) can be used for constraint consistency checking. Then the proposed model transformation (implemented in the VIATRA2 framework [19]) automatically derives EMF metamodel for basic structures. From complex constraints which cannot be expressed directly in the EMF metamodel IQPL graph patterns [4] are generated. Our mapping allows the domain engineer to formulate and check consistency of requirements in an early design phase. The initial version of the target platform can be built automatically from these generated artifacts for instance model editing.

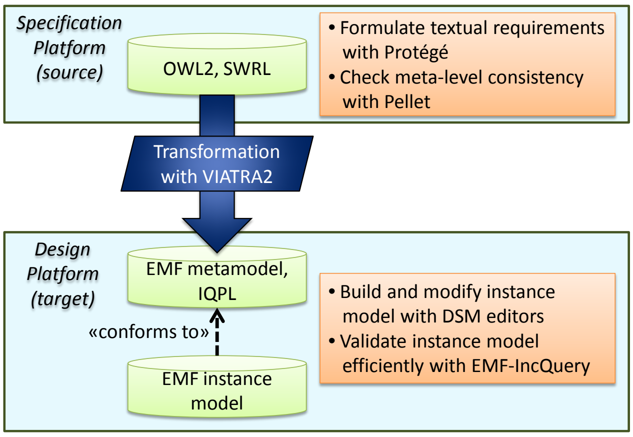

Specification Platform (source)
OWL2, SWRL
• Formulate textual requirements with Protégé
• Check meta-level consistency with Pellet
Transformation with VIATRA2
Design Platform (target)
EMF metamodel, IQPL
«conforms to»
EMF instance model
• Build and modify instance model with DSM editors
• Validate instance model efficiently with EMF-IncQuery

Figure 1: The ontology to domain-specific platform transformation process

Finally, the domain engineer can create and modify a custom instance model with advanced EMF tools, while EMF-INCQUERY performs efficient [15] continuous instance level well-formedness validation in the background by incrementally evaluating IQPL graph patterns after each edit operation.

3. CASE STUDY AND BACKGROUND

3.1 Motivating example

We demonstrate our approach using an example from the railway domain. The domain metamodel originates from the MOGENTES EU FP7 [18] project, and the requirements were defined by railway domain experts.

The EMF representation of the domain, which is derived from the ontology is demonstrated in Fig. 2. A train *route* can be defined by a set of *sensors*. Sensors are associated with *track elements*, which can be a track *segment* or a *switch*. The status of a switch can be *left*, *right* or *failure*. A route can have associated *switch positions*, which describe the required state of a switch belonging to the route. Different route definitions can specify different states for a specific switch.

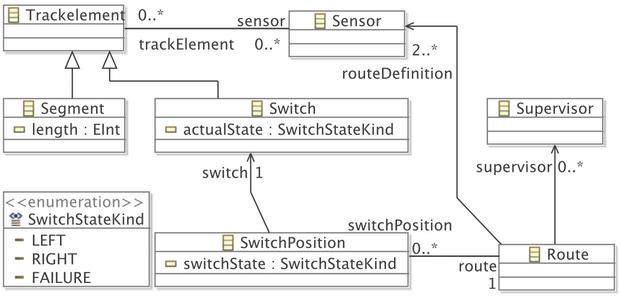


Figure 2: EMF representation of the Train metamodel

Several high-level requirements can be specified that must hold for any valid instance models of a train system such as:

REQ1 Every switch must have at least one sensor connected to it.

REQ2 A segment must have positive length.

REQ3 For every sensor that is associated with a switch and belongs to a route must also be associated directly with the same route.

REQ4 A route must have at least two supervisors unless all switches belonging to the route points LEFT or RIGHT.

REQ5 A segment that is more than 10 km long cannot share sensors with a switch.

3.2 Specification platform

OWL2 (Web Ontology Language) [11] is a W3C standard designed to describe semantic web models. The OWL2 language allows to define classes, called concepts in description logics (DL) and various properties between classes, called roles in DL terminology. Classes and properties are arranged into a subsumption hierarchy, while each property can have a domain and a range class. Furthermore, OWL2 allows to express additional restrictions (constraints) for classes and properties. These constraints are less expressive than first order logic (FOL), but the description logic foundation of ontologies guarantees decidability to provide advanced reasoning capabilities over OWL2 specifications using the direct semantics interpretation.

In Fig. 3 *SwitchPosition* and *Switch* are concepts (classes), which can be related by the *switch* (case sensitive!) relation *(ObjectProperty)*. *SwitchStateKind* is a nominal type (known as enumeration in object oriented languages), which can be *LEFT*, *RIGHT* or *FAILURE*. Every *SwitchPosition* must have exactly one *Thing* which is a *Switch*, and every *Switch* has one *actualState*.

```
ObjectProperty: switch              Class: Switch
 Domain: SwitchPosition              SubClassOf:
 Range: Switch                         Switch_actualState
Class: SwitchPosition                   exactly 1 owl:Thing
 SubClassOf: switch                 Class: SwitchStateKind
      exactly 1 owl:Thing            EquivalentTo:
ObjectProperty: actualState           {SwitchStateKind_LEFT,
  Domain: Switch                        SwitchStateKind_RIGHT,
  Range: SwitchStateKind                SwitchStateKind_FAILURE}
```

Figure 3: Excerpt of the train ontology in Manchester syntax

From a metamodeling viewpoint, the language supports two-level metamodeling, where instances are stored in the ABox (which is the model or instance level knowledge base), while concepts and roles are stored in the TBox (language or meta-level store).

The **Semantic Web Rule Language (SWRL)** [8] is an extension of OWL2 to enhance expressiveness for capturing more complex design constraints for ontologies. An SWRL expression (shown later in Fig. 6) is composed of variables (denoted by a '?' prefix), class expressions and property expressions (expressed as predicates over variables in a Prolog-like notation), and other built-ins, which are primarily used for attribute checking. The basic form of a rule is: antecedent → consequent. If the antecedent is true (for any variable substitution), the consequent must hold, which is conceptually close to queries in logic programming languages.

3.3 Design platform

Eclipse Modeling Framework (EMF) uses *Ecore* metamodels to describe the abstract syntax of a modeling language. Such models can be edited by framework provided graphical editors, which are similar to UML class diagram editors. The main elements of the Ecore language are *EClass* (depicted as a box), *EReference* (or associations) between EClasses (depicted as an edge between boxes), and *EAttribute* of EClasses (depicted in the middle compartment of a box). EReferences and EAttributes can be single-valued or multi-valued and they can be ordered or unordered. EReferences may additionally imply containment, which is mainly used during model serialization. Inheritance may be defined between classes (depicted by an empty arrowhead), which means that the inherited class has all the properties its parent has, and its instances are also instances of the ancestor class, but it may further define some extra features. Note, that inheritance between relations is not supported by the language. An *EObject* instance has exactly one EClass type, and the metamodel is stored separately from instance models. The EMF metamodel of the running example is shown in Fig. 2, which is converted automatically from simple ontology axioms with the tool described in the paper.

IncQuery Pattern Language (IQPL) [4] is based on the formalism of *graph patterns* and has a textual notation that can be seen in Fig. 7. A graph pattern (GP) identifies parts of the instance model that fulfill given conditions (or constraints). A basic graph pattern consists of *structural constraints* prescribing the existence of nodes and edges of a given type. A *negative application condition* (NAC) defines cases when the original pattern is *not* valid (even if all positive constraints are met), in the form of a negative sub-pattern. A match of a graph pattern is a group of model elements that have the exact same structure as the pattern, satisfying all the constraints (except for NACs, which must be violated). The core graph pattern formalism has the expressive power of first order logic, but the query language of EMF-INCQUERY provides semantic extensions such as *recursive patterns*, and *match counting* or practical extensions like *attribute constraints*.

4. MAPPING ONTOLOGIES TO DOMAIN-SPECIFIC LANGUAGES

In this section a bridge is created between ontology descriptions and domain-specific models. The mapping is divided into three steps. First, a part of the OWL2 ontology is mapped into an EMF metamodel, which provides the basic metamodel structure. Afterwards complex OWL2 axioms are mapped into graph patterns and finally SWRL rules are also mapped into graph patterns.

4.1 Mapping OWL2 to EMF

When representing knowledge in a domain-specific model (UML, EMF), the metamodel can be derived from the hierarchy of concepts defined in the TBox of an ontology. Such mappings have been defined by several authors [10,17]. In this paper we rely on the OWL2 to EMF transformation proposed by [13], which is briefly summarized below.

OWL2 classes are mapped to EClasses. Their object properties are transformed into EReferences, and datatype properties to EAttributes. Most OWL2 datatypes have their Ecore datatype counterparts. Both languages support generalization, cardinality constraints and enumeration types. The naming conventions of the mapping are not detailed here, but examples in this paper use self-describing names. Fig. 2 depicts the EMF metamodel derived from the ontology based on the domain.

4.2 Mapping OWL2 to graph patterns

An OWL2 ontology is built up from *ontology axioms*, which are statements about the relationship between two *ontology expressions*. Expressions are composites, consisting of simple expressions, *ontology declarations* and possibly other composites, making up a complex construct that can be represented as a canonical tree. Essentially, the nodes of this tree are n-ary (higher order) functions that can be embedded into each other at arbitrary depth and together express a complex condition that defines the valid members of the instance model universe.

In our approach, complex constructs are mapped into declarative graph pattern language, as they cannot be expressed in the EMF metamodel. Graph patterns are generic queries that rely on a combination of elementary structural constraints as well as attribute value constraints to express a condition over the instance model. A match of a graph pattern is a set of those model element combinations that satisfy the constraints of the pattern. Graph patterns are identified by pattern signatures (name of the pattern, and the name of its parameters), and are registered into a flat namespace. They may reuse (call) each other using the *find* keyword similarly to the semantics of embedded queries (where the result of a sub-query can be used within the scope of the calling query).

When mapping OWL2 constructs to graph patterns, we use an *indirect validation* approach, whereby results of the graph query correspond to those elements, which *violate* the given constraint. In other words, a negated query is generated from OWL2 axioms, in order to efficiently retrieve the set of violations.

Mapping algorithm overview.

The algorithm that processes OWL2 constructs and maps them to graph patterns consists of two main phases. In the first phase, the *metamodel* (type axioms) of the OWL2 input is processed to create the EMF metamodel as described in Section 4.1, as well as auxiliary helper patterns that will be used in complex well-formedness queries to refer to elementary types of the metamodel. In the second phase, complex OWL2 axioms are processed according to a depth-first tree traversal along the canonical decomposition. At each tree node (corresponding to an OWL2 axiom or expression), a transformation is prepared that maps the given OWL2 construct to a corresponding graph pattern that matches exactly when a *violation* of the axiom is found in the instance model using the closed world assumption. As IQPL graph patterns reside in a flat namespace, patterns generated from the same axiom type are distinguished with index numbers attached to their names.

Running example.

In this section, REQ4 is used to describe an example input-output pair of the mapping algorithm. This requirement can be formulated using the OWL2 functional syntax as follows:

```
1  SubClassOf(ObjectIntersectionOf(ObjectComplementOf(
2  ObjectAllValuesFrom(:switchPosition
3  ObjectAllValuesFrom(:switch
4  ObjectSomeValuesFrom(:actualState
5  ObjectOneOf(:SwitchStateKind_RIGHT
6                 :SwitchStateKind_LEFT)))))  :Route)
7  ObjectMinCardinality(2  :supervisor))
```

This axiom operates with two main expressions. The first defines the set of routes that are not in the set (line 1) that collects routes with switches (lines 2-3) positioned either LEFT or RIGHT (lines 4-6). The second expression defines the routes that have at least two supervisors (lines 7) connected to it. The axiom defines that the set defined with the first expression must be subset of the set defined with the second expression (*SubClassOf*).

This OWL2 construct will be transformed into a collection of interconnected graph patterns, where the call tree structure of the graph patterns closely corresponds the structure of the OWL2 formula. The top of the call tree is depicted in Fig. 4, where squared nodes are the patterns of declarations, the top node is the axiom to be checked, and other oval nodes are expressions. Arrows represent pattern composition.

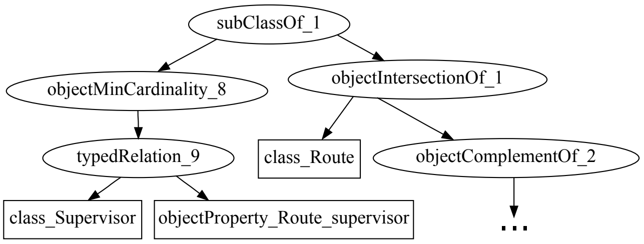


Figure 4: Pattern call tree of REQ4

The overall result of the mapping of example 4.2 is shown in Fig. 5, where the OWL2 tree representation, and the correspondingly generated graph patterns are shown side-by-side.

Metamodel post-processing.

In the first phase of the mapping, OWL2 declarations (that define concept, relation and individual names, i.e. the “metamodel") are processed. As output, auxiliary graph patterns are generated that will be used as elementary type and relationship subqueries in complex expressions. Examples for such auxiliary patterns that can be found in Fig. 5 at lines 18–20 (*switchPosition* role), 27–29 and 38–42 (*switch* role), 48–50 (*Route* concept), and 55–57 (*supervisor* role).

Processing OWL2 axioms and expressions.

Complex OWL2 constructs are processed, according to the depth-first traversal. The following examples highlight some important mapping rules supported by our transformation:

- *SubClassOf axioms* describe subsumption or subset relation between expressions. If an instance is member of the subset, then it must also be a member of the superset. The graph pattern shown in lines 1–4 matches instances that are elements of the subset but are not elements of its superset.
- *Intersections* are Boolean connectives. For instance, bad routes are routes that have not only switches in their left or right position. This can be formalized using *ObjectIntersectionOf*, and can be mapped as described in lines 5–8.
- *Negation* helps e.g. describing individuals without only left or right switches (lines 9–11).
- *Universal quantification* in OWL2 can be mapped by applying the $\forall P.C \equiv \neg\exists P.(\neg C)$ equivalence (double negation). For instance, to find individuals with switches only in the left or right position, the property restriction can be expressed with *ObjectAllValuesFrom*, and can be mapped to the graph pattern as shown in lines 21–26.
- *Existential quantification* is directly supported by the target graph pattern formalism. For example, an ontology expression that matches instances with left or right *actualState* can be written in OWL2 as: *ObjectSomeValuesFrom(actualState ObjectOneOf(LEFT RIGHT))*. This expression matches individuals that have an *actualState*, which is achieved by a pattern reuse/composition, as can be seen in lines 30–33.
- *Enumerations* (expressed by *ObjectOneOf*) are mapped according to the example in lines 42–47, where a pattern is created that matches all instance model elements that are *LEFT* or *RIGHT* as described in lines 37–41.
- *Cardinality and subsumption expressions*, in simple cases, are mapped to an EMF metamodel (Section 4.1).
  OWL2 allows qualified number restrictions, where a number constraint can be applied for a role, and the type of the target can also be constrained. For example, for individuals having at least two supervisors attached (of type Supervisor), the ontology expression supervisor min 2 Supervisor can be mapped using the aggregation feature of the graph pattern language, as illustrated in lines 51–54. Here, the *Individual* pattern variable is bound, and the number of different possible *Target* substitutions are counted in *Count* (relying on the auxiliary pattern shown in lines 55–57). The cardinality constraint is satisfied, if it has at least two appropriate target individuals (as expressed by the *check* condition).

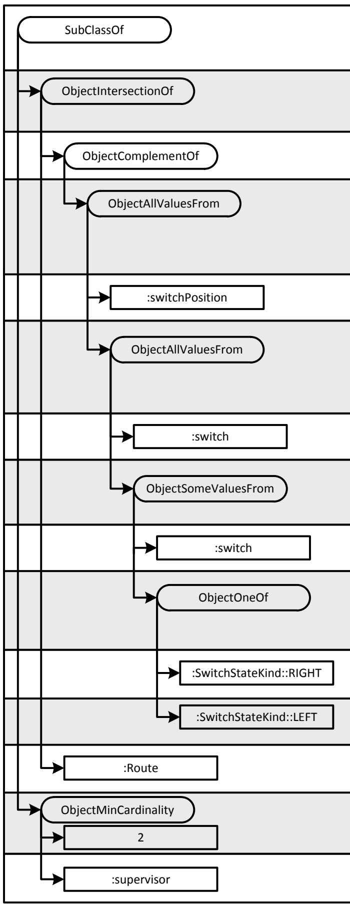
4.3 Mapping SWRL to graph patterns

Mapping the SWRL part of the ontology to graph patterns is based upon the previously defined helper graph patterns.

The constraint REQ5, namely, a segment that is more than 10 km long cannot share sensors with a switch (informally described in Sec. 3.1) can be represented with the SWRL rule in Fig. 6. The violations of this axiom are groups of elements in the instance model that satisfy the entire antecedent, but not the consequent.

Therefore this rule is mapped to the graph pattern in Fig. 7, which is a direct translation of the antecedent, with the consequent added as a NAC. Variables of the SWRL axiom is mapped to graph pattern parameters, so violating elements can be returned in a tuple set. As for transcribing the contents of the antecedent and consequent parts, concept terms and role terms are mapped to pattern calls of class declaration patterns (lines 2, 5, 7, 8) and role declaration patterns (lines 3, 6, 9), respectively.

The SWRL constructs *SameAs* and *DifferentFrom* check whether two variables represent the same entities. In the example *DifferentFrom* is used, which can be transcribed to the pattern language as shown in line 12 of Fig. 7.



```
01 pattern subClassOf_1(Individual)={
02          find objectIntersectionOf_1(Individual);
03          neg find objectMinCardinality_1(Individual);
04 }
05 pattern objectIntersectionOf_1(Individual)={
06          find objectComplementOf_1(Individual);
07          find class_Route(Individual);
08 }
09 pattern objectComplementOf_1(Individual)={
10          neg find objectAllValuesFrom_1(Individual);
11 }
12 pattern objectAllValuesFrom_1(Individual)={
13          neg shareable pattern objectAllValuesFrom_BadRange (Individual) ={
14                    find objectProperty_switchPosition(Individual, Target);
15                    neg find objectAllValuesFrom_2(Target);
16          }
17 }
18 pattern objectProperty_switchPosition(Source : Route, Target : SwitchPosition)={
19          Route.switchPosition(Source, Target);
20 }
21 pattern objectAllValuesFrom_2(Individual)={
22          neg shareable pattern objectAllValuesFrom_BadRange (Individual) ={
23                    find objectProperty_switch(Individual, Target);
24                    neg find objectSomeValuesFrom_1(Target);
25          }
26 }
27 pattern objectProperty_switch(Source : SwitchPosition, Target : Switch)={
28          SwitchPosition.switch(Source, Target);
29 }
30 pattern objectSomeValuesFrom_1(Individual)={
31          find objectProperty_switch (Individual, Target);
32          find objectOneOf_1(Target);
33 }
34
35 // See pattern objectProperty_switch(Source, Target) at lines 29-33
36
37 pattern objectOneOf_1(Individual)={
38          find individual_RIGHT(Individual);
39 } or {
40          find individual_LEFT(Individual);
41 }
42 pattern individual_RIGHT(Individual : SwitchStateKind)={
43          Individual == SwitchStateKind::RIGHT;
44 }
45 pattern individual_LEFT(Individual : SwitchStateKind)={
46          Individual == SwitchStateKind::LEFT;
47 }
48 pattern class_Route(Individual)={
49          Route(Individual);
50 }
51 pattern objectMinCardinality_1(Individual)={
52          Count == count find objProperty_Route_supervisor(Individual, Target);
53          check ((Count as Integer) >= 2 );
54 }
55 pattern objectProperty_supervisor(Source : Route, Target : Supervisor)={
56          Route.supervisor(Source, Target);
57 }
```

Figure 5: The ontology mapping to graph patterns

```
Segment(?seg),
length(?seg, ?sl),
greaterThan(?sl, 10),
Sensor(?sen1),
sensor(?seg, ?sen1),
Switch(?sw),
Sensor(?sen2),
sensor(?sw, ?sen2)

->

DifferentFrom(?sen1, ?sen2)
```

Figure 6: The example SWRL rule

```
pattern swrlRule_1(Sw, Seg, Sen1, S, Sen2, Sl) = {
  find class_Segment(Seg);
  find dataProperty_length(S, Sl);
  find swrlComparison_3(Sl);
  find class_Sensor(Sen1);
  find objectProperty_sensor(Seg, Sen1);
  find class_Switch(Sw);
  find class_Sensor(Sen2);
  find objectProperty_sensor(Sw, Sen2);

  neg pattern head(Sen1 : Thing, Sen2 : Thing) = {
    Sen1 =/= Sen2;
  }
}

pattern swrlComparison_3(Sl : Integer) = {
  check(Sl > 10);
}
```

Figure 7: Mapping SWRL rule ontology axiom

4.4 Discussion of the mapping

Large part of the OWL2 and SWRL languages (around 50 axioms and 45 expressions) are successfully mapped and transformed to EMF. Some expressions were excluded, like DatatypeDefinition or n-ary data range operators, and only major XML and EMF datatype definitions are matched. Some SWRL built-ins and annotations are not implemented in the current transformation, although it could be done by adding new helper graph patterns.

The mapping described in the paper guarantees one-way transformation. Converting basic structures (EMF metamodel) back to ontology can be done, but transforming graph patterns back is not possible entirely, as graph patterns are more expressive than FOL, while ontologies (based on description logics) represent a decidable fragment of logics, thus being less expressive than FOL [14].

A difference between ontology and EMF based applications is how they treat missing information. Ontologies use *open world assumption* (OWA), which means that the truth value of missing information is unknown, they are not used during reasoning. This means that ontologies deal with underspecification and uncertainties helping the language development process. Oppositely, DSM tools use closed world assumption (CWA), which means that missing information is treated as false leading to much more implicit assertions. This approach of data processing assumes complete dataset at a specific time, but DSM tools allow information modification, deletion as well as addition, changing previously false assertions to true. On the other hand ontology reasoners are prepared for monotonic information processing, allowing only the addition of new assertions. This is why an OWA → CWA semantic shift is practical during the transformation of requirements captured in ontologies and constraints used in DSM instance model editing.

There are certain limitations originating from the differences between ontologies and DSM languages. First, in ontologies, different formulae may have the same semantics. For example, the OWL expression $Switch \sqsubseteq \geq 1 sensor$ (every switch must have at least one sensor) is equal to the formula $Switch \sqsubseteq \exists sensor$ (a sensor exists for every switch). Should the Switch class own the sensor relation, it could be described in the EMF metamodel. But as the parent class owns the relation, the constraint can be described only with a graph pattern.

Another difference is that in EMF an EObject is a direct instance of an EClass, and cannot be instance of multiple metamodel elements. As a major difference, in ontologies this kind of multityping is allowed, which cannot be handled with the current transformation.

In EMF there is no relation inheritance, which is common in ontologies. The constraint can be captured by graph patterns, but the programmer is responsible to connect two EMF objects by the actual relation and its ancestors.

Avoiding these problematic cases, deriving an initial version of a DSM using this approach for ensuring structural consistency is practically feasible, as demonstrated by the running example.

5. RELATED WORK

The interaction of semantic web and model-based technologies has already been examined in many, different settings. The EMFTriple [9] project can be considered as a representative of tight integration of these domains, as it provides an EMF-compliant RDF data repository backend as a set of Eclipse plugins. EMFTriple provides no own solution for model validation, but such support can be achieved by using additional modules (e.g. EMFQuery, EMF-IncQuery or OCL).

In the OntoDSL [20] approach, DSL models are enriched by formal class descriptions, which are together checked by an ontology based framework. In this way, the consistency of DSL models can be verified by a reasoning service even in an early design phase. Note that this approach provides services of the ontology domain to the process of DSL specification, in contrast to our solution, which offers instance-level model validation techniques in the domain-specific domain.

In addition to a nice overview on the fundamentals of both the OWL and EMF domains, [13] proposes an automatic transformation from OWL2 ontologies to EMF models and corresponding OCL constraints, and in this sense, this approach shows the largest similarity to ours. The conversion is implemented as an Eclipse plugin, and it is adjustable on both meta and instance level. They mapped OWL2 axioms (without SWRL) to OCL constraints.

The SWRL Drools Tab [3] is a plug-in implemented to Protégé, which maps SWRL extended OWL2 axioms to Drools rules. Here the purpose of the mapping is to perform OWL2 RL reasoning and execute rules, and not to guarantee closed world consistency.

Stardog ICV [5] translates OWL2 and SWRL rules to SPARQL graph patterns, in a formally defined way. This shows the importance of closed world consistency checking of models described in ontology, but this solution remains in the ontology domain, where traditional model-driven development tools cannot be used, and incremental query engine is not available.

6. CONCLUSIONS AND FUTURE WORK

In the current paper, we proposed to adapt advanced meta-level consistency checking techniques offered by ontology reasoners to the DSM design process. For this purpose, we defined a mapping (and implemented in a transformation) from the SWRL extended OWL2 ontology language to EMF metamodels and IncQuery Pattern Language, which can be used to automatically derive a prototype of the DSM system. As a result, we obtain a synergic validation approach: metamodel-level validation is performed by advanced (TBox-level) ontology reasoners (like Pellet or RacerPro), while efficient [15] instance model validation can be carried out using the Eclipse Modeling Framework and EMF-IncQuery.

A developer with graph pattern and EMF knowledge can fine tune or supplement the constraints by editing the generated artifacts of the prototype constraint checking system. These changes cannot be written back into the ontology, which would enable constraint consistency checking of the final system. Practical roud-trip engineering could be implemented in the future, but complete reverse mapping cannot be done due to the greater expressivity of IQPL.

The current implementation transforms TBox axioms to EMF metamodel, while instance model is created by the user at the target platform using DSL tools. The transformation could be easily extended to support transforming ABox individuals to EMF instance model, which would allow transition from existing ontologies.

7. REFERENCES

[1] Pellet: OWL 2 reasoner for Java. http://clarkparsia.com/pellet/.

[2] Protégé ontology editor. http://protege.stanford.edu/.

[3] SWRL Drools Tab, 2012. http://protege.cim3.net/cgi-bin/wiki.pl?SWRLDroolsTab.

[4] G. Bergmann, Z. Ujhelyi, I. Ráth, and D. Varró. A Graph Query Language for EMF models. Springer, 2011.

[5] K. Clark and B. Parsia. Stardog: A commercial RDF database, 2011. http://stardog.com/.

[6] G. Dobson, S. Hall, and G. Kotonya. A domain-independent ontology for non-functional requirements. 2007.

[7] G. Dobson, R. Lock, and I. Sommerville. Quality of service requirements specification using an ontology. 2005.

[8] B. Glimm, M. Horridge, B. Parsia, and P. F. Patel-Schneider. A syntax for rules in OWL 2, 2009.

[9] G. Hillairet. EMFTriple: a tool that brings semantic web languages to the EMF, 2011. http://code.google.com/a/eclipselabs.org/p/emftriple/.

[10] Object Management Group. Ontology Definition Metamodel (OMG) Version 1.0. Technical report, May 2009.

[11] OWL Working Group. OWL 2 Web Ontology Language. http://www.w3.org/2007/OWL/, 2009.

[12] Racer Systems GmbH. Racerpro. http://www.racer-systems.com/products/racerpro/.

[13] T. Rahmani, D. Oberle, and M. Dahms. An adjustable transformation from OWL to Ecore. Springer, 2010.

[14] A. Rensink. Representing first-order logic using graphs. 2004.

[15] I. Ráth. High performance queries and their novel applications, 2012.

[16] The Eclipse Project. Eclipse Modeling Framework. http://www.eclipse.org/emf/.

[17] The Eclipse Project. EMF Ontology Definition Metamodel. http://www.eclipse.org/modeling/mdt/eodm/docs/articles/EODM_Documentation/, 2011.

[18] The MOGENTES project. Model-Based Generation of Tests for Dependable Embedded Systems. http://www.mogentes.eu/.

[19] D. Varró and A. Balogh. The model transformation language of the VIATRA2 framework. October 2007.

[20] T. Walter, F. Silva Parreiras, and S. Staab. OntoDSL: An ontology-based framework for domain-specific languages. Springer, 2009.