A Feasibility Study on the Validation of Domain Specific Languages Using OWL 2 Reasoners

Ye Liu, Sören Höglund, Ali Hanzala Khan, and Ivan Porres

TUCS Turku Centre for Computer Science
D. of Information Technologies, Åbo Akademi University
Joukahaisenkatu 3-5, FI-20520 Turku, Finland
e-mail:name.surname@abo.fi

Abstract. In this article we report on our experiences using the OWL 2 reasoners HermiT and Pellet to validate domain-specific languages defined using UML-like metamodels. Currently there exists few tools to validate metamodels. Using ontologies and reasoners to define and validate metamodels is a possible solution to this problem. We evaluate the reasoners according to expressiveness, correctness, performance and problem reporting capabilities. We use metamodels from the Atlantic Zoo metamodel repository as input for our comparison, and report on our experiences with the zoo.

1 Introduction

Model Driven Engineering (MDE) [8] advocates the use of models to represent the most relevant design decisions in a software development project. Each software model is described using a particular modeling language, such as the Unified Modeling Language (UML) [13] or a domain-specific language. The definition of a modeling language is given using a metamodeling language or a language to define modeling languages. These metamodeling languages share common fundamental concepts such as classes, properties and the specialization of classes and properties, we call such languages UML-like metamodeling languages.

The study of metamodeling languages has led to a number of practical tools such as model repositories, diagram editors, model transformation tools and code generation tools that simplify enormously the creation of new development tool chains that use UML or domain-specific modeling languages in software development projects. However, the topic of metamodel validation is seldom discussed and there is a lack of tool support for this task.

In our previous work [6], we described a mapping from a UML-based metamodeling language to OWL 2. In this article we discuss our practical experiences in validating metamodels from a public repository using two OWL 2 reasoners. The main objective of these experiments is to obtain empirical evidence that the idea of using OWL 2 reasoners to validate metamodels in practice is viable.

We proceed as follows: in Section 2 we discuss the need for metamodel validation. In Section 3 we detail the objectives of the study. In Section 4, Section 5 and Section 6 we present the materials used in the investigation. Section 7 presents the results, and Section 8 is the conclusion.

2 The need for Metamodel Validation

The creation of a new metamodel is not a simple task since it requires a good knowledge of the problem domain and how to capture its properties using a small set of metamodeling concepts. A metamodel often includes many constraints on how concepts in a model can be related to each other, such as multiplicity, domain and range, composition and subset constraints. If not chosen carefully, these constraints may lead to contradictions.

For example, lets assume we are creating a metamodel for a Statechart language that includes concepts such as simple states, composite states (that can contain other states) and transitions between states. A fragment of such a metamodel is shown in the top of Figure 1. This metamodel contains a rather obvious contradiction: there is an association with the minimum multiplicity larger than the maximum multiplicity. This contradiction means that there are no valid models that can make use of this association.

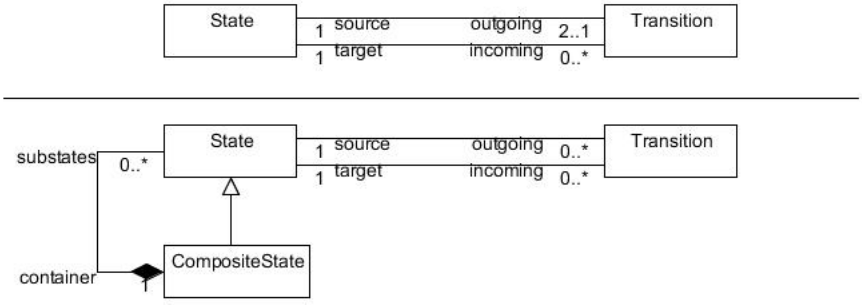

Fig. 1. Examples of invalid metamodels. Top: Invalid due to multiplicity. Bottom: Invalid due to composition error

Another possible contradiction is shown in the bottom of Figure 1. The depicted metamodel requires each instance of State to be in a composite relationship with an instance of CompositeState. However, as CompositeState is a subclass of State it inherits this requirement, which leads to each instance of CompositeState having to be in a composite relationship with another instance of CompositeState. Given a finite set of instances of CompositeState, it is impossible not to violate one of the composition requirements — elements having only one owner and there being no cycles in the compositon. As a consequence there is no model with a finite number of elements that conforms to the given metamodel.

As shown in the examples, we consider it necessary to validate a new metamodel to ensure that no such problems exist before actual tools and models are created.

2.1 Metamodel Validation using OWL2 Reasoners

To tackle the previous problem, we are working towards the integration of an automatic metamodel validation tool in a metamodel editor. The validation is based on the idea of translating a metamodel into an ontology and using a reasoner to check the ontology for consistency and class satisfiability. We have decided to use the OWL 2 Web Ontology Language [1] as our metamodel representation language.

OWL 2 provides classes, properties, individuals, and data values, that we can use to define the main concepts of UML-like metamodeling languages as shown by the authors [6]. OWL 2 has different concrete syntaxes (functional-style syntax, Manchester syntax, RDF/XML, etc.). In this article, we use the OWL 2 functional-style syntax, because it allows ontologies to be written in a compact form and preserves readability.

We have noticed that OWL 2 is not expressive enough for some restrictions in metamodels. For instance, there is no decidable combination of axioms in OWL 2 to express UML composition and its relevant restrictions. This problem could be solved by writing extra rules in another language. One possibility is to use the OCL Object Constraint Language [12]. It is part of the UML standard and it is used to describe metamodel and model constraints. However, OCL operates on the syntactic level, which leads to a very extensive ruleset for UML elements. To avoid this problem, we have decided to use SWRL[7]. SWRL combines OWL and the RuleML (Rule Markup language). It extends the set of OWL axioms to include rules, thus restrictions in metamodels can be represented in SWRL rules.

To validate an ontology expressed in OWL 2 and SWRL, we need a reasoner that supports both languages. There are tools which provide reasoning services for OWL 2 ontologies as well as support of DL-safe SWRL rules [9], such as Pellet and HermiT. We introduce them in Section 6.

It is still an open question if this idea is feasible in practice. In the next section we present a study that tries to answer this question.

3 Objectives of the Study

The main goal of this article is to study the feasibility of using OWL 2 and SWRL reasoners to validate metamodels for domain specific languages. The feasibility criteria is as follows:

1. **Expressiveness of OWL2 and SWRL**
   One of the main objectives of our research is to judge the expressiveness of OWL 2 and SWRL in terms of expressing the concepts of Domain Specific Languages developed by using an UML-like Metamodeling language.
2. **Correctness of Reasoners** Some of the existing reasoners are experimental research tools under constant development. We would like to know whether the current implementation of each reasoner is mature enough to produce reliable results.

3. **Performance**
   We want to know how much processing time is required to validate a metamodel. Can a modern desktop computer perform this task with ease? Can the validation be performed on the fly while editing a metamodel or when we save the metamodel into a file?
4. **Problem Reporting**
   Are the messages produced by an OWL2 reasoner intelligible by the metamodel creator? Can we trace back inconsistencies in a generated ontology to the original metamodel elements?
5. **Problems in published metamodels** Since we are also planning to validate many existing metamodels, we are also interested to know if there are problems in published metamodels.

3.1 Study Execution

In order to answer the previous questions we select a number of metamodels, transform them into an ontology and check them for consistency and satisfiability using a reasoner. We measure execution time and observe eventual error messages.

In order to asses the maturity of the reasoner implementation, we have decided to select two independently developed reasoners and compare their output.

To ensure that the metamodels to validate are representative we have not created the metamodels ourselves. Instead we have decided to process all the metamodels in a public repository: the Atlantic Metamodel Zoo [16].

We describe the materials for our study in the next three sections.

4 The Atlantic Metamodel Zoo

The Atlantic Metamodel Zoo is a library of metamodels maintained by the AtlanMod team. At the time of writing this text, it contains 286 metamodels representing diverse domains such as LateX, Java, HTML. The metamodels are available in different languages, including UML 2, OWL, KM3 etc.. We choose to validate the metamodels expressed in UML 2.

Metamodels expressed in UML 2 are identified by their headers and the elements for describing its structure. The headers must include the namespace URL for the UML metamodel and elements prefixed with `uml:`. An example of such a metamodel is shown in Listing 1.1. This example shows two classes: `Book` and `Chapter` associated by a simple composition.

The metamodels in the Atlantic Zoo vary considerably in size and complexity. In any case, these metamodels are considerably smaller than the UML metamodel and they do not contain OCL constraints. We consider that the metamodels in the Atlantic Zoo provide a good sample of metamodels for domain-specific languages.

5 A Metamodel to OWL2 Transformation

The transformation from a metamodel expressed using UML to OWL 2 is implemented using the Model-to-Text transformation tool MOFScript [5] [11].

MOFScript consists of two parts: the MOFScript tool and the MOFScript language. The MOFScript tool is an implementation of the MOFScript language and it provides ways of editing, compiling and executing MOFScript transformation code. MOFScript transformations are MOFScript language programs that define a set of rules that can translate metamodel elements and relations between them to expected output through print statements. The MOFScript transformation code is written based on one or more input metamodels, then compiled and executed on one or more loaded input files which contains models conforming to the input metamodels.

The input metamodel in our implementation is UML2 2.1.0, the input file is a .uml file which contains a UML metamodel in XML syntax. There are two output files: one contains an OWL 2 ontology written in OWL 2 functional syntax [2] and the other contains SWRL rules in OWL RDF/XML syntax.

As mentioned in the previous sections, there are some restrictions of metamodeling concepts that have to be written in SWRL rules. However, ontology reasoners can not process OWL 2 functional syntax with SWRL rules embedded. As a solution, we create extra SWRL rules written in RDF syntax in a separate OWL document and import this ontology in the main ontology written in OWL 2 functional syntax. For instance, the generated ontologies of the metamodel shown in Listing 1.1 are Listing 1.2 and Listing 1.3.

6 OWL 2 Reasoners

Once the ontology of a metamodel has been generated by using the transformation described before, we use a reasoner to check the ontology consistency and satisfiability. In terms of an ontology representing a metamodel:

1. Metamodel consistency means that there exists at least one model that conforms to that metamodel.
2. Metaclass satisfiability means there is at least one model element that can belong to the class (without making the metamodel inconsistent). In the context of a metamodel, instantiating an unsatisfiable metaclass leads to a contradiction and therefore the metaclass cannot be used in any model. This is also refered as concept satisfiability in the services provided by the reasoners.

The selection criteria for a reasoner were complete support for OWL2, SWRL and also that the reasoner is freely available as open source. The first two requirements are motivated by the ontologies used in our study. The last requirement ensures that the study is easily repeatable by others.

Based on these criteria, we have chosen the following two reasoners:

1. Pellet: An open source Java-based ontology reasoner developed by a Clark& Parsia LLC, which is an R&D firm, specializing in Semantic Web and advanced systems[4].
2. HermiT: An open source reasoner that is implemented in Java, and developed by Information Systems Group of Oxford University [10].

These reasoners are either already available as a Protege (an open source ontology editor) plugin or will be supported. They can also all be used from the command line.

In our experiment, we run HermiT 1.2.1 from the command line and use the options `-k` and `-U` to check the consistency and concept satisfiability respectively. In the case of Pellet, we use version 2.0.1 from the command line and use the options `consistency` and `unsat` to check the consistency and concept satisfiability respectively.

7 Study Results

Since metamodel validation involves both reasoners and metamodels, the results of our study depends on two aspects: the reasoning tools an the input metamodels. First, we evaluate the reasoners in terms of maturity, performance, expressiveness, and their problem reporting mechanism. Then we investigate any reported problems in the published metamodels.

7.1 Expressivness

The expressiveness is evaluated in two aspects: the concepts contained in the metamodels and the Description Logic which the reasoners are based on.

On one hand, the metamodels in the Atlantic Zoo are simple in the sense that they do not cover all the concepts that are within the ability of the UML metamodeling language. For example, there are no metamodels that contain any ordered properties, ordered composition or ordered subset properties. The constraints of these concepts are expressed in SWRL rules.

On the other hand, we notice problems with UML composition: translating composition to OWL 2 requires the transitivity axiom and irreflexive axiom, whereas the OWL 2 specification forbids the two axioms used on the same object property. This can be solved by expressing them in SWRL, although SWRL rules are only applied on individuals rather than classes.

7.2 Maturity of reasoners

Both HermiT and Pellet can process all the ontologies generated from the Atlantic Zoo without generating any run time error or getting into an infinite loop. Furthermore, both reasoners always produce the same report for all the metamodels. This is significant because they have been implemented independently by two different development groups. Based on this, we claim that the current

implementation of both HermiT and Pellet are mature enough for the task of validating metamodels.

During our experiment, we found that Pellet 2.0.1 supports OWL 1.1 functional syntax [3] which is different from OWL 2 in terms of prefix declaration and datatype maps. HermiT supports strict OWL 2 functional syntax. So the ontologies to be processed by HermiT and Pellet are slightly different syntactically, Accordingly, the MOFScript transformation for each syntax should be different too.

7.3 Performance

We run our performance experiments using a desktop computer with an Intel Core 2 6400 processor running at 2.13GHz, 2GB of RAM, Linux Fedora Core 10 and Java 1.6.0_10_rc2. We process each metamodel three times and report the average execution time.

Figure 2 shows the time required for HermiT and Pellet to process each metamodel. The x-axis represents the number of axioms in a generated ontology. The y-axis represents the total time in seconds necessary to load an ontology representing a metamodel, and check it for consistency and satisfiability. As for the counting of number of axioms, though, certain expressions are a combination of OWL 2 axioms. For instance, the minimum cardinality in the Figure 1 is expressed as follows, which is a combination of axiom `SubClassOf` and `ObjectMinCardinality`, we count this as one axiom.

```
SubClassOf(State ObjectMinCardinality(2 State_outgoing ))
```

As we can observe from the figure, HermiT is consistently faster than Pellet, but both tools can process each metamodel in less than four seconds.

Based on this, we consider that the effiency of the two reasoners is satisfactory for the given problem and an average desktop computer.

7.4 Problem Reporting

Pellet produces an error message if a given input contains a syntax error or the wrong file header. Such input cannot be further checked for consistency and unsatisfiability. When checking consistency, Pellet simply shows if the input ontology is consistent or not. When checking unsatisfiability, given a consistent ontology, Pellet reports the number of elements checked, time used and number of unsatisfiable elements. If there are unsatisfiable elements, the specific class names are shown.

Pellet provides an option to print verbose information while reasoning. The printed information indicating input size, specific numbler of classes, properties and individuals, expressivity, used time summary, etc.. Once an element that leads to an inconsistency is found, Pellet stops further processing and points out a possible reason, but the actual reason still needs be verified by users.

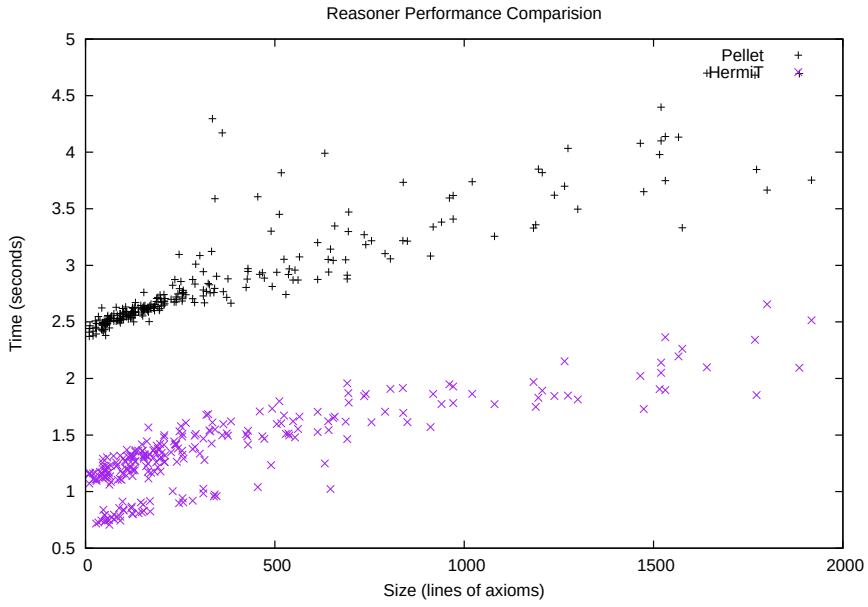

Fig. 2. Reasoners Performance Comparision

HermiT also provides verbose infomation printing, such as the file under processing, timing for parsing, etc.. In contrast to Pellet, when an input is inconsistent, HermiT shows no possible reasons. As far as processing procedure is concerned, HermiT differs from Pellet in that it allows checking unsatisfiablity of an inconsistent metamodel.

Regardless of how the two reasoners report the results, the information they give can be difficult to interpret when a metamodel is inconsistent or has unsatisfiable concepts. In the best case, they show the name of the unsatisfiable concepts but no further explanations. More information on what makes a concept unsatisfiable would be extremely helpful.

7.5 Problems in Published Metamodels

During our experiment with the 286 metamodels in the Atlantic zoo, we found 279 metamodels that are consistent and satisfiable and 7 with validation problems. The 7 unsatisfiable metamodels are `ekaw`, `paperdyne`, `pcs`, `micro`, `sigkdd`, `Openconf` and `iasted`. These are all generated from ontologies dealing with conference organization by the OntoFarm Project [14] .

We investigated the reason of the unsatisfiablity and found problems in the metamodels in the source .uml files. The original metamodels have two classes `Evaluated_Paper` and `Assigned_Paper`, which both have a self association with a minimum cardinality of 3 and a maximum cardinality of 1.

We also examined the original ontologies [15] from which the UML metamodels are generated. The ontologies only state that the minimum cardinality is 3, but do not contain any axiom about the maximum cardinality. We suspect the problem lies in the transformation used to convert each ontology to UML.

8 Conclusions

In this article we have studied the feasibility of an approach to validate metamodels using a OWL 2 reasoners. The first step of validation is transforming UML metamodels to OWL 2 ontologies. We use MOFScript to implement the transformation from UML to OWL 2 and SWRL. The second step is to validate the generated ontologies with a reasoner. This second step was the focus of this article.

We have processed all the metamodels in the Atlantic Zoo. This comprises 286 metamodels for domain-specific languages created independently. To our knowledge this is the most comprehensive empirical evaluation on the use of OWL 2 in metamodeling to the date.

The size of the metamodels can be considered small when compared to the metamodel of the UML language. None of the metamodels use OCL constraints or advanced metamodeling features such as subset properties. However, we consider that the Atlantic Zoo provides a good sample of how metamodel in domain-specific languages are used in practice.

Based on our experiences, we consider that OWL 2 combined with SWRL can be used to represent all the metamodels in the repository. Also, the Pellet and HermiT reasoners could process each metamodel in less than four seconds and always provided the same results.

References

1. Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler, and Michael Smith. *OWL 2 Web Ontology Language Document Overview*. W3 Recommendation Available at `http://www.w3.org/TR/owl2-overview/`.
2. Conrad Bock, Achille Fokoue, Peter Haase, Rinke Hoekstra, Ian Horrocks, Alan Ruttenberg, Uli Sattler, and Michael Smith. *OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax*. W3 recommendation, available at `http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/`.
3. Boris Motic, Peter F. Patel-Scheneider, Ian Horrocks. *OWL 1.1 Web Ontology Language Structural Specification and Functional-Sytle Syntax*, May 2007. Available at `http://www.webont.org/owl/1.1/owl_specification.html`.
4. Clark and Parsia. *Pellet: OWL 2 Reasoner for Java*. Homepage, available at `http://clarkparsia.com/pellet`.

5. Eclipse Foundation. *MOFScript Homepage*. Available at `http://www.eclipse.org/gmt/mofscript/`.
6. Sören Höglund, Ali H. Khan, Ye Liu, and Ivan Porres. Representing and Validating Metamodels using OWL 2 and SWRL. Technical Report 973, D. of Information Technologies,Åbo Akademi University, Joukahaisenkatu 3-5, FI-20520 Turku, Finland, 2010.
7. Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, 2004. Availible at `http://www.w3.org/Submission/SWRL/`.
8. Stuart Kent. Model Driven Engineering. In *Proc. of IFM International Formal Methods 2002*, volume 2335 of *LNCS*. Springer-Verlag, 2002.
9. Vladimir Kolovski, Bijan Parsia, and Evren Sirin. Extending the shoiq(d) tableaux with dl-safe rules: First results. In Bijan Parsia, Ulrike Sattler, and David Toman, editors, *Description Logics*, volume 189 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2006.
10. Information System of Oxford University. *Hermit OWL Reasoner*.
11. Jon Oldevik. *MOFScript User Guide*, December 2009. Document available at `http://www.eclipse.org/gmt/mofscript/doc/MOFScript-User-Guide-0.8.pdf`.
12. OMG. *UML 2.0 OCL Specification*, October 2003. Document, available at `http://www.omg.org/`.
13. OMG. *UML 2.2 Superstructure Specification*, February 2009. Available at `http://www.omg.org/`.
14. OntoFarm Project. Ontofarm project homepage. Available at `http://nb.vse.cz/~svatek/ontofarm.html`.
15. OntoFarm Project. Conference track. Available at `http://nb.vse.cz/~svabo/oaei2009/`.
16. AtlanMod Team. AtlanMod metamodel zoo. Available at `http://www.emn.fr/z-info/atlanmod/index.php/Zoos`.

Appendix

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <uml:Model xmi:version="2.1"
       xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
       xmlns:uml="http://www.eclipse.org/uml2/3.0.0/UML"
       xmi:id="_dkpFAOiaEd6gMtZRCjS81g" name="Metamodel">
3    <packagedElement xmi:type="uml:Package"
         xmi:id="_dkpFAeiaEd6gMtZRCjS81g" name="Book">
4      <packagedElement xmi:type="uml:Association"
           xmi:id="_dkpFCuiaEd6gMtZRCjS81g"
           name="A_Book_Chapter"
           memberEnd="_dkpFCeiaEd6gMtZRCjS81g
           _dkpFC-iaEd6gMtZRCjS81g">
5        <ownedEnd xmi:id="_dkpFC-iaEd6gMtZRCjS81g" name="book"
             type="_dkpFA-iaEd6gMtZRCjS81g" isUnique="false"
             association="_dkpFCuiaEd6gMtZRCjS81g">
6          <upperValue xmi:type="uml:LiteralUnlimitedNatural"
               xmi:id="_dkpsEuiaEd6gMtZRCjS81g" value="1"/>
```

```
7           <lowerValue xmi:type="uml:LiteralInteger"
                xmi:id="_dkpsE-iaEd6gMtZRCjS81g" value="1"/>
8         </ownedEnd>
9       </packagedElement>
10      <packagedElement xmi:type="uml:Class"
            xmi:id="_dkpFA-iaEd6gMtZRCjS81g" name="Book">
11        <ownedAttribute xmi:id="_dkpFBeiaEd6gMtZRCjS81g"
              name="title" type="_dkpFDeiaEd6gMtZRCjS81g"
              isUnique="false"/>
12        <ownedAttribute xmi:id="_dkpFCeiaEd6gMtZRCjS81g"
              name="chapters" type="_dkpFBOiaEd6gMtZRCjS81g"
              isUnique="false" aggregation="composite"
              association="_dkpFCuiaEd6gMtZRCjS81g">
13          <upperValue xmi:type="uml:LiteralUnlimitedNatural"
                xmi:id="_dkpsEOiaEd6gMtZRCjS81g" value="*"/>
14          <lowerValue xmi:type="uml:LiteralInteger"
                xmi:id="_dkpsEeiaEd6gMtZRCjS81g"/>
15        </ownedAttribute>
16      </packagedElement>
17      <packagedElement xmi:type="uml:Class"
            xmi:id="_dkpFBOiaEd6gMtZRCjS81g" name="Chapter">
18        <ownedAttribute xmi:id="_dkpFBuiaEd6gMtZRCjS81g"
              name="title" type="_dkpFDeiaEd6gMtZRCjS81g"
              isUnique="false"/>
19        <ownedAttribute xmi:id="_dkpFB-iaEd6gMtZRCjS81g"
              name="nbPages" type="_dkpFDOiaEd6gMtZRCjS81g"
              isUnique="false"/>
20        <ownedAttribute xmi:id="_dkpFCOiaEd6gMtZRCjS81g"
              name="author" type="_dkpFDeiaEd6gMtZRCjS81g"
              isUnique="false"/>
21      </packagedElement>
22    </packagedElement>
23    <packagedElement xmi:type="uml:Package"
          xmi:id="_dkpFAuiaEd6gMtZRCjS81g" name="PrimitiveTypes">
24      <packagedElement xmi:type="uml:PrimitiveType"
            xmi:id="_dkpFDOiaEd6gMtZRCjS81g" name="Integer"/>
25      <packagedElement xmi:type="uml:PrimitiveType"
            xmi:id="_dkpFDeiaEd6gMtZRCjS81g" name="String"/>
26      <packagedElement xmi:type="uml:PrimitiveType"
            xmi:id="_dkpFDuiaEd6gMtZRCjS81g" name="Boolean"/>
27    </packagedElement>
28  </uml:Model>
```

Listing 1.1. An example of metamodel presented in UML 2

```
1  Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema#>)
2  Prefix(owl2xml:=<http://www.w3.org/2006/12/owl2-xml#>)
3  Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
4  Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
5  Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
```

```
6  Prefix(:=<http://www.zoo.org/examples#>)
7
8  Ontology( <http://www.zoo.org/examples#>
9   Import( <file://Book.owl>)
10
11 Declaration(Class(:Book_Book))
12 FunctionalDataProperty(:Book_Book_title)
13 DataPropertyDomain(:Book_Book_title :Book_Book)
14 DataPropertyRange(:Book_Book_title xsd:string)
15
16 Declaration(Class(:Book_Chapter))
17 FunctionalDataProperty(:Book_Chapter_title)
18 DataPropertyDomain(:Book_Chapter_title :Book_Chapter)
19 DataPropertyRange(:Book_Chapter_title xsd:string)
20
21 FunctionalDataProperty(:Book_Chapter_nbPages)
22 DataPropertyDomain(:Book_Chapter_nbPages :Book_Chapter)
23 DataPropertyRange(:Book_Chapter_nbPages xsd:int)
24
25 FunctionalDataProperty(:Book_Chapter_author)
26 DataPropertyDomain(:Book_Chapter_author :Book_Chapter)
27 DataPropertyRange(:Book_Chapter_author xsd:string)
28
29 Declaration(ObjectProperty(:contains))
30             IrreflexiveObjectProperty( :contains )
31             SubObjectPropertyOf(:owns :contains)
32             InverseFunctionalObjectProperty(:owns)
33             SubClassOf(:Book_Book ObjectAllValuesFrom(:owns
                   :Book_Chapter))
34             SubClassOf(:Book_Chapter_DInstance
                   ObjectAllValuesFrom(:owns owl:Nothing))
35 Declaration(ObjectProperty(:Book_Book_chapters))
36 ObjectPropertyDomain(:Book_Book_chapters :Book_Book)
37 ObjectPropertyRange(:Book_Book_chapters :Book_Chapter)
38 SubClassOf(:Book_Book ObjectMinCardinality(0
       :Book_Book_chapters))
39
40 SubClassOf(:Book_Book_DInstance ObjectAllValuesFrom(:owns
       owl:Nothing))
41
42 Declaration(ObjectProperty(:Book_Book_book))
43 ObjectPropertyDomain(:Book_Book_book :Book_Book)
44 ObjectPropertyRange(:Book_Book_book :Book_Book)
45 SubClassOf(:Book_Book ObjectMinCardinality(1 :Book_Book_book))
46 SubClassOf(:Book_Book ObjectMaxCardinality(1 :Book_Book_book))
47 )
```

Listing 1.2. Ontology in OWL 2 functional syntax

```
1  <?xml version="1.0" encoding="UTF-8"?>
```

```
<rdf:RDF
xmlns:rdf = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'
xmlns:rdfs = 'http://www.w3.org/2000/01/rdf-schema#'
xmlns:xsd = 'http://www.w3.org/2001/XMLSchema#'
xmlns:owl = 'http://www.w3.org/2002/07/owl#'
xmlns:swrlx = 'http://www.w3.org/2003/11/swrlx'
xmlns:swrl='http://www.w3.org/2003/11/swrl#'
xmlns:ruleml ='http://www.w3.org/2003/11/ruleml'
xmlns:owlx = 'http://www.w3.org/2003/05/owl-xml'
xmlns = 'http://example.org/compoRule#'
>
<owl:Ontology rdf:about="compoRule"/>

   <owl:ObjectProperty rdf:ID="contains"/>
    <swrl:Variable rdf:ID="x"/>
    <swrl:Variable rdf:ID="y"/>
    <swrl:Variable rdf:ID="z"/>
    <swrl:Variable rdf:ID="a"/>
    <swrl:Variable rdf:ID="b"/>
    <ruleml:Imp>
      <ruleml:body >
        <swrl:IndividualPropertyAtom>
           <swrl:propertyPredicate rdf:resource="#contans"/>
           <swrl:argument1 rdf:resource="#x"/>
           <swrl:argument2 rdf:resource="#y"/>
 </swrl:IndividualPropertyAtom>
 <swrl:IndividualPropertyAtom>
           <swrl:propertyPredicate rdf:resource="#contans"/>
           <swrl:argument1 rdf:resource="#y"/>
           <swrl:argument2 rdf:resource="#z"/>
 </swrl:IndividualPropertyAtom>
      </ruleml:body>
      <ruleml:head>
        <swrl:IndividualPropertyAtom>
           <swrl:propertyPredicate rdf:resource="#contans"/>
           <swrl:argument1 rdf:resource="#x"/>
           <swrl:argument2 rdf:resource="#z"/>
 </swrl:IndividualPropertyAtom>
      </ruleml:head>
    </ruleml:Imp>
</rdf:RDF>
```

Listing 1.3. SWRL rules written in OWL RDF syntax