Bridging Concrete and Abstract Syntax of Web Rule Languages

Milan Milanović[1], Dragan Gašević[2], Adrian Giurca[3], Gerd Wagner[3], Sergey Lukichev[3], and Vladan Devedžić[1]

[1] FON-School of Business Administration, University of Belgrade, Serbia
milan@milanovic.org, devedzic@etf.bg.ac.yu
[2] School of Computing and Information Systems, Athabasca University, Canada
dgasevic@acm.org
[3] Institute of Informatics, Brandenburg Technical University at Cottbus, Germany
{Giurca, G.Wagner Lukichev}@tu-cottbus.de

Abstract. This paper proposes a solution for bridging abstract and concrete syntax of a Web rule language by using model transformations. Current specifications of Web rule languages such as Semantic Web Rule Language (SWRL) define its abstract syntax (e.g., EBNF notation) and concrete syntax (e.g., XML schema) separately. Although the recent research in the area of Model-Driven Engineering demonstrates that such a separation of two types of syntax is a good practice (due to the complexity of languages), one should also have tools that check validity of rules written in a concrete syntax with respect to the abstract syntax of the rule language. In this study, we use analyze the REWERSE I1 Rule Markup Language (R2ML) whose abstract syntax is defined by using metamodeling, while its textual concrete syntax is defined by using XML schema. We bridge this gap by a bi-directional transformation defined in a model transformation language (i.e., ATL).

1 Introduction

Using and sharing rules on the Web are some of the main challenges that the Web community tries to solve. The first important stream of research in this area is related to the Semantic Web technologies where researchers try to provide formally-defined rule languages (e.g., Semantic Web Rule Language, SWRL [7]) that are used for reasoning over Semantic Web ontologies. The main issue to be solved is the type (e.g., open of closed world) of reasoning that will be used, so that formal-semantics of such languages can be defined. However, as in constructing any other language, defining abstract syntax (independent of machine encoding) and concrete syntax (machine-dependent representation) is an unavoidable part of the language definition. An important characteristic of Semantic Web rule languages is that they are primarily not dealing with interchange of rules between various types of rules on the Web. This means that Semantic Web rule languages do not tend to compromise their reasoning characteristics for the broader syntactic expressivity. This is actually the main focus on the second stream of research on the Web that is chiefly articulated through the W3C effort called Rule Interchange Format (RIF) [6], while the most known effort in

M. Marchiori, J.Z. Pan, and C. de Sainte Marie (Eds.): RR 2007, LNCS 4524, pp. 309–318, 2007.

that area is the RuleML language [4]. The primary result expected from this research stream is to define an XML-based concrete syntax for sharing rules on the Web. Although the XML syntax for such a language is certainly the pragmatic expectation of the Web users, for a good definition of such a language is also important to have a well-designed abstract syntax.

In this paper, we try to address the problem of bridging the gap between an abstract and concrete syntax of a Web rule interchange language, i.e., the REWERSE I1 Rule Markup Language (R2ML) [18], one of the most-known RIF proposals. Since this language leverages the benefits of a new software engineering discipline Model-Driven Engineering (MDE) [3], the abstract syntax R2ML is defined by a metamodel. Furthermore, the R2ML XML schema, i.e., R2ML concrete syntax, has been developed for encoding rules by domain experts. However, there is no solution that enables transforming XML documents compliant to the R2ML XML documents into representation compliant to the R2ML metamodel (simply R2ML models). This gap between the R2ML metamodel and the R2ML XML schema causes the following problems:

1. Rules represented in the R2ML XML format cannot be stored in MOF-based model repositories, thus cannot be validated w.r.t. the R2ML metamodel.
2. The R2ML metamodel can not be instantiated based on rules encoded in the R2ML XML schema, and thus the R2ML metamodel can not be validated with real-world rules.

2 Model Driven Engineering

Model Driven Engineering is a new software engineering discipline in which the process heavily relies on the use of models [3]. A model defined is a set of statements about some system under study [16]. Models are usually specified by using modeling languages (e.g., UML), while modeling languages can be defined by metamodels. A metamodel is a model of a modeling language. That is, a metamodel makes statements about what can be expressed in the valid models of a certain modeling language [16]. The OMG's Model Driven Architecture (MDA) is one possible architecture for MDE [11]. One important characterestic of MDA is its organization. In fact, it consists of three layers, namely: M1 layer or model layer where models are defined by using modeling languages; M2 layer or metamodel layer where models of modeling languages (i.e. metamodels) are defined (e.g., UML) by using metamodel languages; and M3 layer or metametamodel layer where only one metamodeling language is defined (i.e. MOF) by itself [12].

The relations between different MDA layers can be considered as instance-of or conformant-to, which means that a model is an instance of a metamodel, and a metamodel is an instance of a metametamodel. The rationale for having only one language on the M3 layer is to have a unique grammar space for defining various modeling languages on the M2 layer. Thus, various modeling language can be processesed in the same way by using the same API. An example of such an API's are Java Metadata Interface (JMI)[1] that enables the implementation of a dynamic, platform-independent

[1] http://java.sun.com/products/jmi/

infrastructure to manage the creation, storage, access, discovery, and exchange of metadata. The most comprehensive implementation of JMI is NetBeans Metadata Repository (MDR).

Although MDE principles of defining modeling languages seems quite promising, the reality is that languages related can be defined and represented by using various technologies such as XML, databases, and MOF. In fact, the MDE theory introduces a concept of technical spaces, where a technical space is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities [9]. Although some technical spaces are difficult to define, they can be easily recognized (e.g. XML, MDA). In the case of the problem analyzed in this paper, we have to bridge between two technical spaces, since the R2ML metamodel and R2ML XML schema are defined in the MOF and XML technical spaces, respectively.

We should also mention the model transformations that represent the central operation for handling models in the MDA. Model transformations are the process of producing one model from another model of the same system [11]. In our research, we have decided to use ATLAS Transformation Language (ATL) [1] as the model transformations tool, which is based on OMG's QVT specification [13].

3 R2ML Metamodel and R2ML XML Schema

This section is devoted to the description of the R2ML language [15] [18] by explaining the R2ML abstract syntax and R2ML XML-based concrete syntax. Due to the size of the R2ML language, we only give an excerpt of the language related to integrity rules in this section. For the complete definition of the R2ML metamodel and R2ML XML schema, we advise readers to see [15].

3.1 The R2ML Abstract Syntax: R2ML Metamodel

The R2ML metamodel is defined by using the MOF metamodeling language. In Fig. 1, we give a UML class diagram depicting the MOF definition of integrity rules. An integrity rule, also known as (integrity) constraint, consists of a constraint assertion, which is a sentence (or formula without free variables) in a logical language such as first-order predicate logic. R2ML supports two kinds of integrity rules: the *alethic* and *deontic* ones. An alethic integrity rule can be expressed by a phrase, such as "*it is necessarily the case that*" and a deontic one can be expressed by phrases, such as "*it is obligatory that*" or "*it should be the case that*."

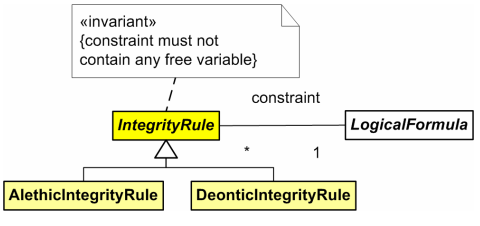

Fig. 1. The metamodel of integrity rules

Example 1 (Integrity rule). *If rental is not a one way rental then return branch of rental must be the same as pick-up branch of rental.*

R2ML defines the general concept of *LogicalFormula* (see Fig. 2) that can be Conjunction, Disjunction, NegationAsFailure, StrongNegation, and Implication. The

concept of a *QuantifiedFormula* is essential for R2ML integrity rules, and it subsumes existentially quantified formulas and universally quantified formulas. Fig. 2 also contains elements such as *AtLeastQuantifiedFormula*, *AtMostQuantifiedFormula*, and *AtLeastAndAtMostQuantifiedFormula* that allow defining cardinality constrains in the R2ML rules.

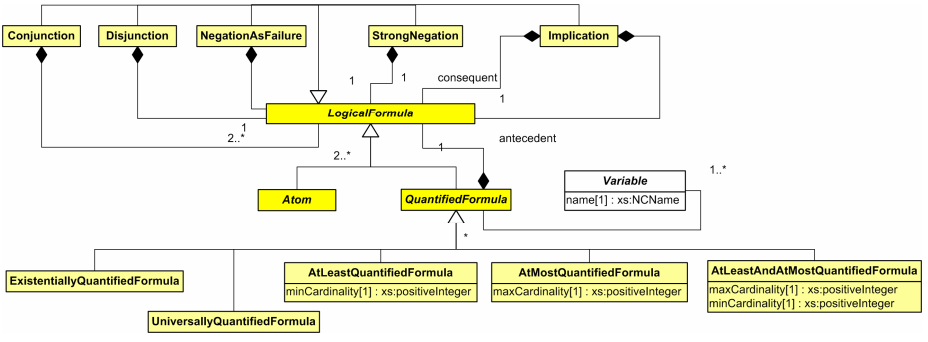

Fig. 2. The MOF model of LogicalFormula

3.2 R2ML XML Schema

The concrete syntax of the R2ML language is defined in a form of an XML schema. This XML schema is defined based on the R2ML MOF-based metamodel by using the following mapping rules presented in Table 1, while the full definition of the R2ML XML schema can be found [15]. In Fig. 3, we give the integrity rules defined in Example 1 in a form of an XML document compliant to the R2ML XML schema.

```
<r2ml:AlethicIntegrityRule r2ml:id="IR001">
  <r2ml:constraint>
    <r2ml:UniversallyQuantifiedFormula>
      <r2ml:ObjectVariable r2ml:name="r1" r2ml:classID="Rental"/>
      <r2ml:Implication>
        <r2ml:antecedent>
          <r2ml:NegationAsFailure>
            <r2ml:ObjectClassificationAtom r2ml:classID="OneWayRental">
              <r2ml:ObjectVariable r2ml:name="r1"/>
            </r2ml:ObjectClassificationAtom>
          </r2ml:NegationAsFailure>
        </r2ml:antecedent>
        <r2ml:consequent>
          <r2ml:EqualityAtom>
            <r2ml:ReferencePropertyFunctionTerm
              r2ml:referencePropertyID="returnBranch">
              <r2ml:contextArgument>
                <r2ml:ObjectVariable r2ml:name="r1"/>
              </r2ml:contextArgument>
            </r2ml:ReferencePropertyFunctionTerm>
            <r2ml:ReferencePropertyFunctionTerm
              r2ml:referencePropertyID="pickupBranch">
              <r2ml:contextArgument>
                <r2ml:ObjectVariable r2ml:name="r1"/>
              </r2ml:contextArgument>
            </r2ml:ReferencePropertyFunctionTerm>
          </r2ml:EqualityAtom>
        </r2ml:consequent>
      </r2ml:Implication>
    </r2ml:UniversallyQuantifiedFormula>
  </r2ml:constraint>
</r2ml:AlethicIntegrityRule>
```

Fig. 3. R2ML XML representation of the integrity rule from Example 1

One may raise a natural question: What do we need an XML schema for R2ML and the above design rules when there is XMI and rules how to produce an XMI schema from MOF-based models, metamodels, and metametamodels [14]. We decided to build this XML schema, as XMI is too complex for our needs and the XMI schema model goes into an extremely verbose XML syntax, hard to be used by humans, which is not in our design goals. However, the benefit of the use of XMI is that they can be processed by model repositories, thus we can test out the validity of XMI documents w.r.t. MOF-based metamodels.

4 Transformations Between the R2ML XML Schema and the R2ML Metamodel

In this section, we explain the transformation steps undertaken to transform R2ML XML documents into the models compliant to the R2ML metamodel. The R2ML concrete syntax is located in the XML technical space. However, the R2ML metamodel is defined by MOF, so the metamodel is located in the MOF technical space. To develop transformations between these two rule representations, we should put them into the same technical space. One alternative is to develop transformations in the XML technical space by using XSLT. This means that documents in the R2ML XML formant have to be transformed into the documents represented in the XMI format, compliant to the R2ML metamodel. However, the present practice has demonstrated that the use of XSLT as a solution is hard to maintain [8], since small modifications of the input and output XML formats can completely invalidate the XSLT transformation. This is especially amplified when transforming highly verbose XML formats such as XMI. On the other hand, we can perform this transformation in the MOF technical space by using model transformation languages such as ATL that are easier to maintain and have better tools for managing MOF-based models. We base our solution on the second alternative, i.e., in the MOF technical space by using ATL. The overall organization of the transformation process is shown in Fig. 4. It is obvious that transformation between the R2ML XML schema and the R2ML metamodel consists of two transformations, namely: 1. From the R2ML metamodel to the R2ML XML schema (i.e., from the XML technical space to the MOF technical space); and 2. From the R2ML XML schema to the R2ML metamodel.

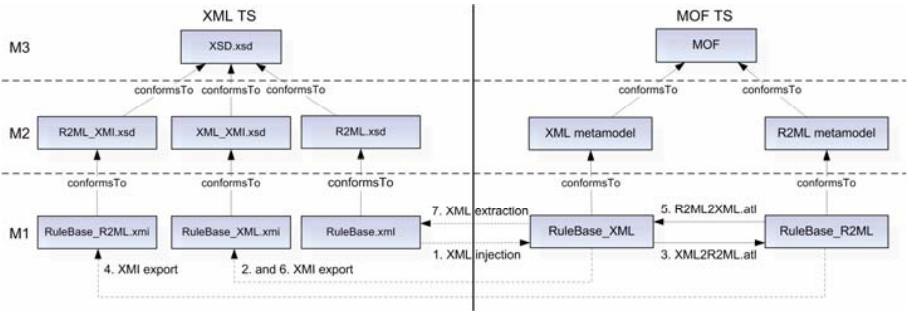

Fig. 4. The transformation scenario: R2ML XML into the R2ML metamodel and vice versa

4.1 Transforming the R2ML XML Schema into the R2ML Metamodel

The transformation process consists of two primary steps as follows.

Step 1. XML injection from the XML technical space to the MOF technical space. This means that we have to represent R2ML XML documents (RuleBase.xml from Fig. 4) into the form compliant to MOF. We use the XML injector that transforms R2ML XML documents (written w.r.t. the R2ML XML Schema, i.e., R2ML.xsd from Fig. 4) into the models conforming to the MOF-based XML metamodel (step 1 in Fig. 4). This

has an extremely low cost, since the XML injector is distributed as a general-purpose tool together with ATL, which performs the XML injection automatically. An XML model (RuleBase_XML in Fig. 4), created by the XML injector, is located on the M1 layer of the MDA. This means that the XML injector instantiates the MOF-based XML metamodel (i.e., abstract syntax of XML). We can manipulate with these models like with any other type of MOF-based metamodels. Thus, such XML models can be represented in the XMI format (step 2 in Fig. 4). This XMI format can be regarded as an implicitly defined XML schema (XML_XMI.xsd) compliant to the XML metamodel.

Step 2. A transformation of XML models into R2ML models. We transform an XML model (RuleBase_XML) created in Step 1 into an R2ML model (RuleBase_R2ML) by using an ATL transformation named XML2R2ML.atl (step 3 in Fig. 4). The output R2ML model (RuleBase_R2ML) conforms to the R2ML metamodel. In the XML2R2ML.atl transformation, source elements from the XML metamodel are transformed into target elements of the R2ML metamodel. The XML2R2ML.atl transformation is done on the M1 level (i.e., the model level) of the MDA. This transformation uses the information about elements from the M2 (metamodel) level, i.e., metamodels defined on the M2 level (i.e., the XML and R2ML metamodels) in order to provide transformations of models on the level M1. It is important to point out that M1 models (both source and target ones) must be conformant to the M2 metamodels. This principle is well-know as metamodel-driven model transformations [2]. In Table 1, we give an excerpt of mappings between the R2ML XML Schema, XML metamodel, and R2ML metamodel. For XML Schema complex types, an instance of the XML metamodel element is created through the XML injection described in Step 1 above. Such an XML element is then transformed into an instance of the R2ML metamodel element by using the XML2R2ML.atl transformation (Step 2).

Table 1. An excerpt of mappings between the R2ML XML schema and the R2ML metamodel

R2ML schema	XML metamodel	R2ML metamodel	Description
`IntegrityRuleSet`	`Element name = 'r2ml:IntegrityRuleSet'`	`IntegrityRuleSet`	Captures a set of integrity rules.
`AlethicIntegrityRule`	`Element name = 'r2ml:AlethicIntegrityRule'`	`AlethicIntegrityRule`	Represents an alethic integrity rule.
`ObjectVariable`	`Element name = 'r2ml:ObjectVariable'`	`basContVoc.ObjectVariable`	Represents an object variable.

Mappings between elements of the XML metamodel and elements of the R2ML metamodel are defined as a sequence of rules in the ATL language. These rules use additional helpers functions in defining mappings. Each rule in the ATL has one input element (i.e., an instance of a metaclass from a MOF-based metamodel) and one or more output elements. In fact, the ATL transformation takes an input XML model from a model repository and creates a new model compliant to the R2ML metamodel.

After applying the above ATL rules to the input XML models, R2ML models (RuleBase_R2ML) are stored in the model repository. Such R2ML models can be exported in the form of R2ML XMI documents (e.g., RuleBase_R2ML.xmi in Fig. 4).

4.2 Transforming the R2ML Metamodel into the R2ML XML Schema

Along with the transformation of the R2ML XML schema to the R2ML metamodel, we have also defined a transformation in the opposite direction, i.e., from the R2ML metamodel to the R2ML XML schema (R2ML2XML). This transformation process consists also of two primary steps as follows.

Step 1. The transformation of R2ML models to XML models. We transform an R2ML model (RuleBase_R2ML from Fig. 4) into an XML model (RuleBase_XML) by using an ATL transformation named R2ML2XML.atl (step 5 in Fig. 4). After applying this transformation to the input R2ML models, XML models (RuleBase_XML) are stored in the model repository (RuleBase_XML.xmi in Fig. 4). The output XML model conforms to XML metamodel. Mappings from Table 1 apply here with no changes. So, for the R2ML rules given the R2ML XMI format, we get an XML model which can be serialized back into the XML XMI format (step 6 in Fig. 4).

Step 2. The XML extraction from the MOF technical space to the XML technical space. In this step, we transforms XML model (RuleBase_XML in Fig. 4) which conforms to MOF-based XML metamodel and is generated in step 1 above, to RuleBase.xml document (Step 7 in Fig. 4). The XML extractor is a part of the ATL toolkit.

Creating a transformation from the R2ML metamodel to the R2ML XML schema (R2ML2XML), appeared to be easier to implement than the XML2R2ML transformation. For the R2ML2XML transformation, we needed only one helper for checking the negation of Atoms. All the ATL matched transformation rules are defined straightforward similar to the XML2R2ML transformation, except for unique elements (like *ObjectVariable*).

5 Experiences

The transformation is tested on a set of real world rules collected by the REWERSE Working Group I1 at the Brandenburg University of Technology at Cottbus. In this section, we report on some lessons we learned in developing and applying the transformation. These lessons also helped us to validate the R2ML MOF-based metamodel as well as to propose some changes of the R2ML metamodel.

Missing associations. Our goal was to transform rules from the R2ML XML format into the R2ML metamodel. This helped us identify some associations missing in the R2ML metamodel without which we could not represent all relations existing in the R2ML XML format. For example, the IntegrityRuleSet and DerivationRuleSet complex types are sequences of IntegrityRule and DerivationRule, respectively, in the R2ML XML schema. This implicated that in the R2ML metamodel we had to add an association between IntegrityRuleSet and IntegrityRule as well as another association between DerivationRuleSet and DerivationRule.

Abstract classes. Originally, some classes of the R2ML metamodel were defined as abstract classes (e.g., Disjunction, Conjunction, and Implication) [18]. When we attempted to transform rules form the R2ML XML format into the R2ML metamodel, we faced the problem that ATL engine refused executing the ATL transformation. The problem was that some classes should not actually be abstract, as the MDR model repository prevented their instantiation by strictly following the R2ML metamodel definition. This was an obvious indicator to change such classes not to be abstract.

Conflicting compositions. Since the meaning of MOF compositions is fully related to instances of classes connected by compositions, it is very hard to validate the use of compositions in MOF-based metamodels without instantiating metamodels. This means that for a class A that composes a class B, an instance of the class B can be only composed by one and only one instance of the class A. It is also correct to say that a class C also composes the class B. However, the same instance of the class B can not be composed by two other instances, regardless of the fact that one of them is a instance of the class A and another one of the class C. Since ATL uses the MDR as model repository, MDR does not allow us to execute ATL transformations that break the MOF semantics including the part related to compositions. This actually helped us identify some classes (e.g., *term* association from the ObjectClassificationAtom class to the ObjectTerm class, *objectArguments* association from the AssociationAtom class to the ObjectTerm class, etc.) in the R2ML metamodel breaking this rule. To overcome this problem, we have changed ("relaxed") the composition with a regular association relation. This makes sense, since a variable should be declared once, while all other elements should refer to that variable (not compose it).

Multiple inheritance conflict. During the implementation of the injection and transformation from the R2ML XML to the R2ML metamodel, we noticed the well-known "diamond" problem [17], i.e. a multiple inheritance conflict, in the object-oriented paradigm. Such a conflict arises when a class, say N, obtains the same attribute *attr* from two or more parent class; let us say classes A and B. These both parent classes A and B have the same parent class C from which both of them inherit the *attr*, thus there is a conflict to determine from which of them the attribute is inherited and how to access it at the class N. In the previous version of the R2ML metamodel, we defined three types of Variables: ObjectVariable, DataVariable and Variable which is parent from first two Variables. The problem occurred because ObjectVariable inherited ObjectTerm (which inherited Term), but it also inherited Variable, which also inherited Term, as shown in Fig. 5a. In this way, ObjectVariable inherited the class Term's attributes (i.e., *isMultivalued*) from two parents, namely, ObjectTerm and Variable. The same situation was with DataVariable and DataTerm. We solved this situation (Fig. 5b), as follows. First, we introduced the GenericTerm class which inherits the Term class, and the GenericVariable class which inherits GenericTerm. Next, we changed the Variable class, which is now an abstract class and it is a parent class for the GenericVariable and ObjectVariable classes. In this way, ObjectVariable only inherits Term's attributes from one parent only (ObjectTerm). Finally, we should note that we have a similar solution for DataVariable.

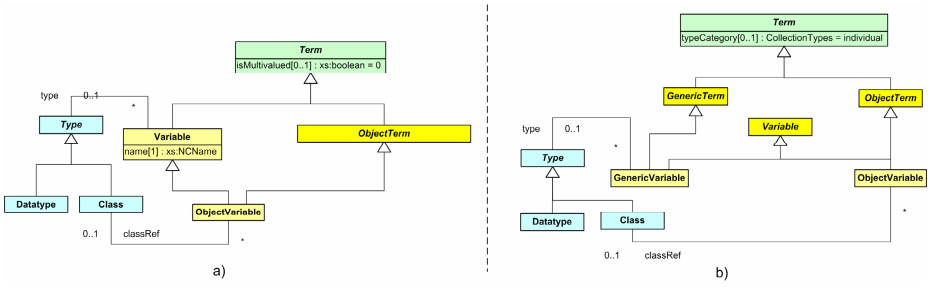

Fig. 5. The multiple inheritance conflict with (a) and its solution (b)

6 Conclusions

To the best of our knowledge, there is no solution to transforming rule languages based on model transformation languages. Most of previous solutions to transforming rule languages such as RuleML and SWRL are implemented by using XSLT or programming languages (Java) [5]. By the nature, our solution is the most similar to those based on the use of XSLT, as a general purpose transformation language for the XML technical space. Examples of transformations for the R2ML which are developed by using XSLT [15] such as translators from R2ML to F-Logic, between the F-Logic XML format and R2ML, from R2ML to Jess (rule engine), R2ML to RuleML, etc.

In this paper, we have demonstrated potentials of model transformations for transforming rule languages. First, the use of model transformation languages forces us to use valid source and target models. This means that the transformation can not be executed properly if either of rule models is not fully conformant to its metamodel. In our case, the source R2ML XML rules have to be conformant to the XML metamodel, while R2ML models have to be conformant to the R2ML metamodel. Second, every time we execute the model transformation, the elements of the target model are instantiated in the model repository. This means that the model transformation provided us with the mechanism for instantiation of the R2ML metamodel. This helped us detect some issues in the R2ML metamodel such as conflicting compositions and inappropriate abstract classes. Third, instances of rule metamodels are stored into MOF-based repositories. Since model repositories have generic tools for exporting/importing (meta)models in the XMI format, we employ them to export instances of the R2ML metamodel in the XMI format, and thus share R2ML models with other MOF-compliant applications. Finally, the use of ATL is more appropriate than XSLT when transforming rules between the XML and MOF technical spaces, since ATL supports advanced features for transforming languages based on metamodels.

In the future work, we will use real-world rules that we have transformed into the R2ML metamodel to evaluation transformations between the R2ML metamodel and other rule languages. Currently, we are implementing a bi-directional model transformation between the R2ML metamodel and the MOF-based OCL metamodel and between the R2ML metamodel and the SWRL language whose abstract syntax is defined by a metamodel. Of course, in this research we have to address even more

challenges, since we need to bridge between three technical spaces, namely, XML (SWRL concrete syntax), EBNF (OCL concrete syntax), and MOF (metamodels of R2ML, OCL, and SWRL) [10].

References

1. ATLAS Transformation Language (ATL), http://www.sciences.univ-nantes.fr/lina/atl
2. Bézivin, J.: From Object Composition to Model Transformation with the MDA. In: Proceedings of the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems, Santa Barbara, USA, pp. 350–355 (2001)
3. Bézivin, J.: On the unification power of models. Software and System Modeling 4(2), 171–188 (2005)
4. Boley, H.: The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations. In: INAP 2001. LNCS (LNAI), vol. 2543, pp. 5–22. Springer, Heidelberg (2003)
5. Gandhe, M., Finin, T., Grosof, B.: SweetJess: Translating DamlRuleML to Jess. In: Proceedings of the International Workshop on Rule Markup Languages for Business Rules on the Semantic Web at 1st International Semantic Web Conference, the Sardinia, Italy (2002)
6. Ginsberg, A.: RIF Use Cases and Requirements, W3C Working Draft, http://www.w3.org/TR/rif-ucr (2006)
7. Horrocks I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C Member Submission, http://www.w3.org/Submission/SWRL (2004)
8. Jovanović, J., Gašević, D.: XML/XSLT-Based Knowledge Sharing. Expert Systems with Applications 29(3), 535–553 (2005)
9. Kurtev, I., Bézivin, J., Aksit, M.: Technological Spaces: an Initial Appraisal, CoopIS, DOA'2002, Industrial track (2002)
10. Milanović, M., Gašević, D., Guirca, A., Wagner, G., Devedžić, V.: On Interchanging between OWL/SWRL and UML/OCL. In: Proceedings of 6th Workshop on OCL for (Meta-) Models in Multiple Application Domains (OCLApps) at the 9th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MoDELS), Genoa, Italy, pp. 81–95 (2006)
11. Miller, J., Mukerji, J. (eds.): MDA Guide Version 1.0.1, OMG (2003)
12. Meta Object Facility (MOF) Core, v2.0, OMG Document formal/06-01-01, http://www.omg.org/cgi-bin/doc?formal/2006-01-01 (2005)
13. MOF QVT Final Adopted Specification, OMG document 05-11-01, (2005)
14. Meta Object Facility (MOF) 2.0 XMI Mapping Specification, v2.1, OMG Document formal/2005-09-01, http://www.omg.org/cgi-bin/doc?formal/2005-09-01 (2005)
15. The REWERSE I1 Rule Markup Language (R2ML), http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/6 (2006)
16. Seidewitz, E.: What Models Mean, IEEE Software, pp. 26–32 (2003)
17. Simons, A.J.H.: The Theory of Classification, Part 17: Multiple Inheritance and the Resolution of Inheritance Conflicts. Journal of Object Technology 4(2), 15–26 (2005)
18. Wagner, G., Giurca, A., Lukichev, S., R2ML: A General Approach for Marking-up Rules", In Proceedings of Dagstuhl Seminar 05371. In Bry, F., Fages, F., Marchiori, M., Ohlbach, H. (Eds.) Principles and Practices of Semantic Web Reasoning, http://drops.dagstuhl.de/opus/volltexte/2006/479 (2005)