Electronic Communications of the EASST
Volume 5 (2006)

Proceedings of the Sixth OCL Workshop OCL for (Meta-)Models in Multiple Application Domains (OCLApps 2006)

Towards Sharing Rules Between OWL/SWRL and UML/OCL

Milan Milanović, Dragan Gašević, Adrian Giurca, Gerd Wagner and Vladan Devedžić

18 Pages

Guest Editors: Dan Chiorean, Birgit Demuth, Martin Gogolla, Jos Warmer
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Towards Sharing Rules Between OWL/SWRL and UML/OCL

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

[1]milan@milanovic.org, devedzic@etf.bg.ac.yu
FON-School of Business Administration, University of Belgrade, Serbia

[2]dgasevic@acm.org
School of Computing and Information Systems, Athabasca University, Canada
School of Interactive Arts and Technology, Simon Fraser University Surrey, Canada

[3]Giurca@tu-cottbus.de, G.Wagner@tu-cottbus.de
Institute of Informatics, Brandenburg Technical University at Cottbus, Germany

Abstract: The paper presents a metamodel-driven model transformation approach to interchanging rules between the Semantic Web Rule Language along with the Web Ontology Language (OWL/SWRL) and Object Constraint Language (OCL) along with UML (UML/OCL). The solution is based on the REWERSE Rule Markup Language (R2ML), a MOF-defined general rule language, as a pivotal metamodel and the bi-directional transformations between OWL/SWRL and R2ML and between UML/OCL and R2ML. Besides describing mapping rules between three rule languages, the paper proposes the implementation by using ATLAS Transformation language (ATL) and describes the whole transformation process involving several MOF-based metamodels, XML schemas, EBNF grammars.

Keywords: OWL, SWRL, OCL, UML, MOF, XML, EBNF, R2ML, Model transformations, ATL.

1 Introduction

The benefits of bridging Semantic Web and Model-Driven Architecture (MDA) technologies have been recognized by researchers awhile ago. On one hand, ontologies are a backbone of the Semantic Web defined for sharing knowledge based on explicit definitions of domain conceptualization. The Web Ontology Language (OWL) has been adopted as a de facto language standard for specifying ontologies on the Web. On the other hand, models are the central concepts of Model Driven Architecture (MDA). Having defined a model as a set of statements about the system under study, software developers can create software systems that are verified with respect to their models. Such created software artifacts can easily be reused and retargeted to different platforms (e.g., J2EE or .NET). UML is the most famous modeling language from the pile of MDA standards, which is defined by a metamodel specified by using Meta-Object Facility (MOF), while MOF is a metamodeling language for specifying metamodels, i.e. models of modeling languages. Considering that MDA models and Semantic Web ontologies have different purposes, the researchers identified that they have a lot in

common such as similar language constructs (e.g., classes, relations, and properties), very often represent the same/similar domain, and use similar development methodologies [GDD06]. The bottom line is the OMG's Ontology Definition Metamodel (ODM) specification that defines an OWL-based metamodel (i.e. ODM) by using MOF, an ontology UML profile, and a set of transformations between ODM and languages such as UML, OWL, ER model, topic maps, and common logics [ODM06]. In this way, one can reuse the present UML models when building ontologies.

In this paper, we further extend the research in approaching the Semantic Web and MDA by proposing a solution to interchanging rules between two technologies. More specifically, we address the problem of mapping between the Object Constraint Language (OCL), a language for defining constrains and rules on UML and MOF models and metamodels, and the Semantic Web Rule Language (SWRL), a language complementing the OWL language with features for defining rules. In fact, our proposal covers the mapping between OCL along with UML (i.e., UML/OCL) and SWRL along with OWL (OWL/SWRL). The main idea of the solution is to employ the REWERSE Rule Markup Language (R2ML) [R2ML06], [WGL05], [WGL*06] (a MOF-defined general rule language capturing integrity, derivation, production, and reaction rules), as a pivotal metamodel for interchanging between OWL/SWRL and UML/OCL. This means that we have to provide a two way mappings for either of two rule languages with R2ML. The main benefit of such an approach is that we can actually map UML/OCL rules into all other rule languages (e.g., Jess, F-Logic, and Prolog) that have mappings defined with R2ML. Since various abstract and concrete syntax are used for representing and sharing all three metamodels (e.g., R2ML XMI, R2ML XML, OWL XML, OCL XMI, UML XMI, OCL text-based syntax), the implementation is done by using Atlas Transformation Language (ATL) [ATL06] and by applying the metamodel-driven model transformation principle [Béz01].

2 Motivation

In this section, we give a simple example of sharing OWL/SWRL and UML/OCL rules, in order to motivate our work. Let us consider an example of a UML model representing relations between members of a family. For a given class *Person*, we can define a UML association with the *Person* class itself modeling that one person is a parent of another one. This is represented by the *hasFather* association end. In the similar way, we can represent relations that one person has a brother by adding another association with the *hasBrother* association end to our model. However, if we one wants to represent that a person has an uncle, i.e. the *hasUncle* association end, this should be derived based on *hasFather* and *hasBrother* association ends, by saying if a person has a father, and the father has a brother, then the father's brother is an uncle of the person. This can be expressed by the UML class diagram and OCL-text based concrete syntax as it is shown in Figure 1.

The same model can be represented as an OWL ontology consisting of the *Person* class and object properties *hasFather*, *hasBrother*, and *hasUncle* (see

Figure 2a). Like in the UML model where the OCL rule has been defined for the *hasUncle* association end, a SWRL rule has to be defined on the OWL ontology for inferring the value for the *hasUncle* object property. This SWRL rule is given in Figure 2b.

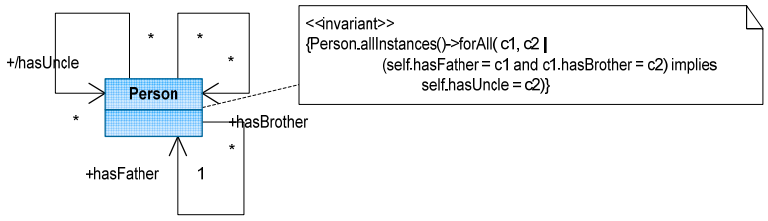

Figure 1. The family UML model and a OCL invariant on the *Person* class

```
<rdf:RDF>
   <owl:Ontology rdf:about=""/>
   <owl:Class rdf:ID="Person"/>
   <owl:ObjectProperty rdf:ID="hasUncle">
      <rdfs:domain rdf:resource="#Person"/>
      <rdfs:range rdf:resource="#Person"/>
   </owl:ObjectProperty>
   <owl:ObjectProperty rdf:ID="hasFather">
      <rdfs:domain rdf:resource="#Person"/>
      <rdfs:range rdf:resource="#Person"/>
   </owl:ObjectProperty>
   <owl:ObjectProperty rdf:ID="hasBrother">
      <rdfs:domain rdf:resource="#Person"/>
      <rdfs:range rdf:resource="#Person"/>
   </owl:ObjectProperty>
</rdf:RDF>
```

(a) The family OWL ontology

```
<ruleml:imp>
  <ruleml:_body>
    <swrlx:individualPropertyAtom swrlx:property="hasFather">
      <ruleml:var>x1</ruleml:var>
      <ruleml:var>x2</ruleml:var>
    </swrlx:individualPropertyAtom>
    <swrlx:individualPropertyAtom swrlx:property="hasBrother">
      <ruleml:var>x2</ruleml:var>
      <ruleml:var>x3</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_body>
  <ruleml:_head>
    <swrlx:individualPropertyAtom swrlx:property="hasUncle">
      <ruleml:var>x1</ruleml:var>
      <ruleml:var>x3</ruleml:var>
    </swrlx:individualPropertyAtom>
  </ruleml:_head>
</ruleml:imp>
```

(b) SWRL rule in the XML concrete syntax

Figure 2. The family OWL ontology and a SWRL rule in the XML concrete syntax

Even from this rather simple example, one can easily recognize many different languages that are directly involved in the process of interchanging OWL/SWRL and UML/OCL. Given the solution based on the use of the R2ML metamodel as a pivotal metamodel, we can identify the following languages: i) OWL and SWRL abstract syntax, OWL and SWRL XML syntax, and OWL RDF/XML syntax is used for OWL/SWRL; ii) R2ML abstract syntax, R2ML XMI concrete syntax, R2ML XML concrete syntax are used for R2ML; and iii) UML abstract syntax, OCL abstract syntax, UML XMI concrete syntax, OCL XMI concrete syntax, and OCL text-based concrete syntax are used for UML/OCL. Here we advocate a solution that is based on defining mappings between abstract syntax of the three languages where each syntax is represented by MOF, i.e. by a MOF-based metamodel. This means that we can exploit transformation tools (e.g., ATL) for MOF-based model to enable interchange between OWL/SWRL and UML/OCL. In the rest of the paper, we first describe R2ML as the core of our solution, and later we give a full process (conceptual mappings at the level of abstract syntax and implementation details) of transforming between R2ML and OWL/SWRL, and between R2ML and UML/OCL, and thus between OWL/SWRL and UML/OCL.

3 The Interchange Format R2ML

This section is devoted to the description of integrity rules of R2ML [R2ML06], [WGL05], [WGL*06] developed by the REWERSE WG I11 that is used as a basis for interchanging between OWL/SWRL and UML/OCL.

[1] REWERSE Working Group I1–Rule Markup, http://www.rewerse.net/I1

R2ML supports four kinds of rules, namely, integrity rules, derivation rules, production rules, and reaction rules. R2ML covers almost all of the use cases requirements of the W3C RIF WG [RIF06]. Since both SWRL rules and OCL constraints are integrity rules, we just describe R2ML integrity rules here. An *integrity rule*, also known as *(integrity) constraint*, consists of a constraint assertion, which is a sentence in a logical language such as first-order predicate logic or OCL [OCL06] (see Figure 3). The R2ML framework 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*."

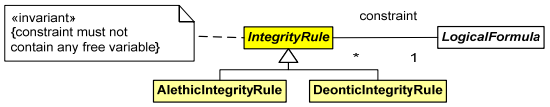

Figure 3. The R2ML definition of integrity rules

The corresponding *LogicalFormula* must have no free variables, i.e. all the variables from this formula must be quantified. The metamodel of *LogicalFormula* is depicted in Figure 4. All first order logic constructs for formulas are supported, i.e. conjunctions, disjunctions, and implications.

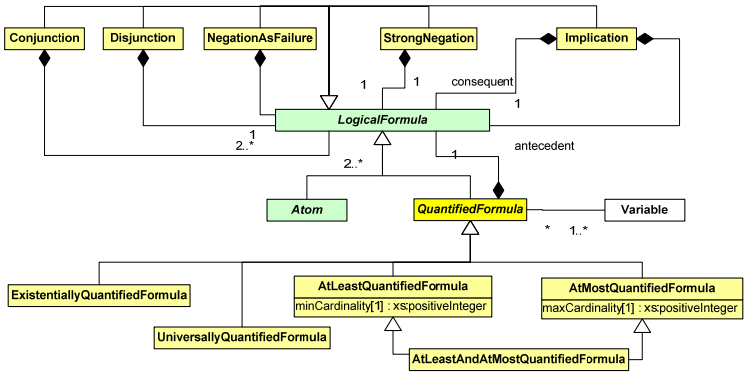

Figure 4. R2ML logical formula

The distinction between a weak and strong negation is used in several computational languages: it is presented in an explicit form in extended logic programs [GL91], only implicitly in SQL and OCL, as was shown in [Wag03]. Intuitively, a weak negation captures the absence of positive information, while a strong negation captures the presence of explicit negative information (in terms of Kleene's 3-valued logic). Under the minimal/stable models [GL88], a weak negation captures the computational concept of negation-as-failure (or closed-world negation) [Cla78].

Quantified formulas, i.e. formulas in which all variables are quantified, represent the core of integrity constraints. Since expressing cardinality restrictions with plain logical formulas leads to cumbersome constructions, R2ML introduces *"at least/most n"* quantified formulas.

Atoms are basic constituents for formulas in R2ML. Atoms are compatible with all important concepts of OWL/SWRL. R2ML distinguishes object atoms (see Figure 5) and data atoms (see Figure 6). The design of atoms is tailored to the UML [UML06] and OCL [OCL06]

concepts as well as to OWL [PSH04] and SWRL [HPB*04] concepts. Here we present just R2ML atoms necessary for our goal. See [WGL*06] for a complete description and use of all supported atoms. An *ObjectClassificationAtom* refers to a class and consists of an object term. Its role is for object classification, i.e. an *ObjectTerm* is an instance of the referred class. A *ReferencePropertyAtom* associates an object term as *"subject"* with other object term as "*object*." This atom corresponds to the UML concept of object evaluated property, to the concept of an RDF [KC04] triple with a non-literal object, to an OWL object property, and to the OWL concept of value for an individual-valued property.

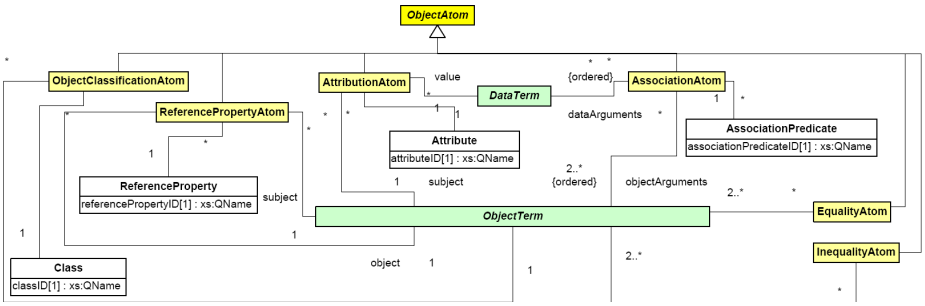

Figure 5. Object Atoms

An *AttributionAtom* consists of a reference to an attribute, an object term as "*subject*," and a data term as "*value*." It corresponds to the UML concept of attribute and to the OWL concept of value for a data-valued property.

In order to support common fact types of natural language directly, it is important to have *n*-ary predicates (for n > 2). R2ML's *AssociationAtom* is constructed by using an *n*-ary predicate as an association predicate, an ordered collection of data terms as "*dataArguments*," and an ordered collection of object terms as "*objectArguments*." It corresponds to the *n*-ary association concept from UML.

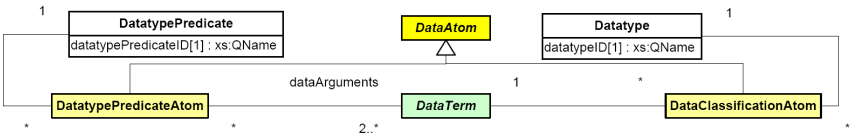

Figure 6. Data Atoms

R2ML *EqualityAtom* and *InequalityAtom* consist of two or more object terms. They correspond to the *SameIndividual* and *DifferentIndividuals* OWL concepts. An R2ML *DataClassificationAtom* consists of a data term and refers to a datatype. Its role is to classify data terms. An R2ML *DataPredicateAtom* refers to a datatype predicate, and consists of a number of data terms as data arguments. Its role is to provide user-defined built-in atoms. It corresponds to the built-in atom concept of SWRL.

Terms are the basic constituents of atoms. As well as UML, the R2ML language distinguishes between object terms (Figure 7) and data terms (see Figure 8). An *ObjectTerm* is an *ObjectVariable*, an *Object*, or an *object function term*, which can be of two different types:

1. An *ObjectOperationTerm* is formed with the help of a *contextArgument*, a user-defined operation, and an ordered collection of arguments. This term can be mapped to an OCL

FeatureCallExp by calling an object valued operation in the context of a specific object described by the *contextArgument*.

2. The *RoleFunctionTerm* corresponds to a functional association end (of a binary association) in a UML class model.

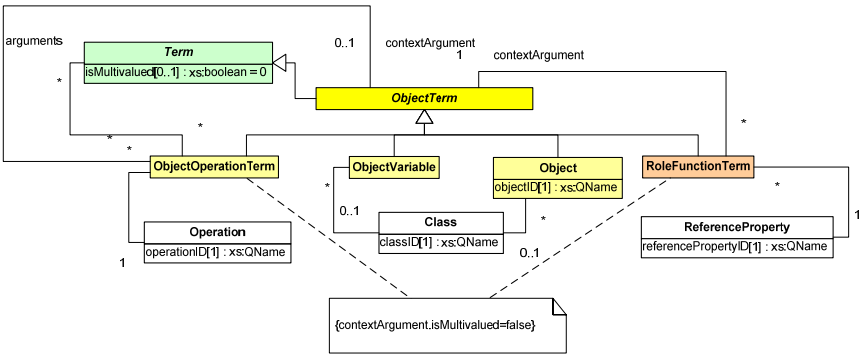

Figure 7. Object Terms

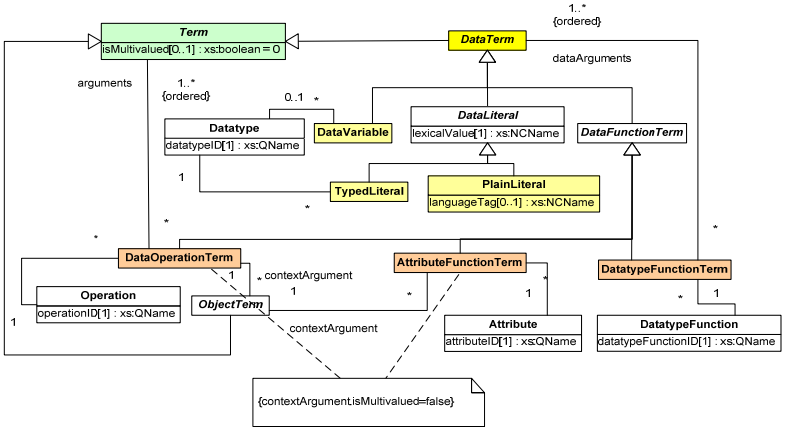

Figure 8. Data Terms

Objects in R2ML are the same artifacts like in UML. They also correspond to the *Individual* concept of OWL. *Variables* are provided in the form of *ObjectVariable* (i.e. variables that can be only instantiated by objects) and *DataVariable* (i.e. variables that can be only instantiated by data literals).

The concept of data value in R2ML is related to the RDF concept of data literal. As well as RDF, R2ML distinguishes between plain and typed literals (see *DataLiteral* and its subclasses in Figure 8). They also correspond to the OCL concept of *LiteralExp*.

A *DataTerm* (Figure 8) is either a data *DataVariable*, a *DataLiteral*, or a *data function term*, which can be of three different types:

1. A *DatatypeFunctionTerm* formed with the help of a user-defined *DatatypeFunction* and a nonempty, ordered collection of *dataArguments*.
2. An *AttributeFunctionTerm* formed with the help of a *contextArgument* and a user-defined *Attribute*.

3. A *DataOperationTerm* formed with the help of a *contextArgument*, a user-defined *operation* that takes as *arguments* an ordered collection of terms.

All of them are useful for the representation of OCL expressions, for example, in *FeatureCallExp* involving data valued operations.

4 Transforming OWL/SWRL to R2ML

In this section, we explain the transformation steps undertaken to transform OWL/SWRL rules into R2ML. In a nutshell, this mapping consists of two transformations. The first one is from OWL/SWRL rules represented in the OWL/SWRL XML format [HEP03] into the models compliant to the RDM (Rule Definition Metamodel) [BH06]. Second, such RDM-based models are transformed into R2ML models, which are compliant to the R2ML metamodel and this represents the core of the transformation between the OWL/SWRL and R2ML.

The rationale for introducing one more metamodel, i.e. RDM, is that it represents an abstract syntax of the SWRL (with OWL) language in the MOF technical space. As well as SWRL is based on OWL, RDM is also relies on the most recent ODM specification [ODM06]. However, OWL/SWRL is usually represented and used in the XML concrete syntax that is a combination of the OWL XML Presentation Syntax [HEP03] and the SWRL XML concrete syntax [HPB*04], i.e. in the XML technical space. However, the RDM 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 XSTL. However, the present practice has demonstrated that the use of XSLT as a solution is hard to maintain [FSS03] [JG05], since small modifications in the input and output XML formats can completely invalidate the present 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 [ATL06] that are easier to maintain and have better tools for managing MOF-based models. This approach has another important benefit, namely, MOF-based models can automatically be transformed into XMI. We decide to develop the solution in the MOF technical space by using the ATL transformation language. The transformation process consists of three steps as follows. Speaking in terms of ATL, the first step is injection of the SWRL XML files into models conforming to the XML metamodel (Figure 9). The second step is to create RDM models from XML models, and this process is shown in the right part of Figure 9 . The third step is transforming such RDM models into R2ML models in the MOF technical space (i.e. the core transformation of the abstract syntax).

Step 1. This step consists of injecting OWL/SWRL rules from the XML technical space into the MOF technical space. Such a process is shown in detail for R2ML XML and the R2ML metamodel in [MGG*06]. This step means that we have to represent OWL/SWRL XML documents (Rules.xml from Figure 9) into the form compliant to MOF. We use the XML injector that transforms R2ML XML documents into the models conforming to the MOF-based XML metamodel that defines XML elements such as XML Node, Element, and Attribute. This XML injector is distributed as a tool along with the ATL engine. The result of this injection is an XML model that can be represented in the XML XMI format, which can be later used as the input for the ATL transformation. We start our transformation process from the SWRL rule defined in Section 2 and shown in the OWL/SWRL XML concrete syntax (

Figure 2). Figure 10 shows the XML model which is injected from the SWRL given in Figure 2.

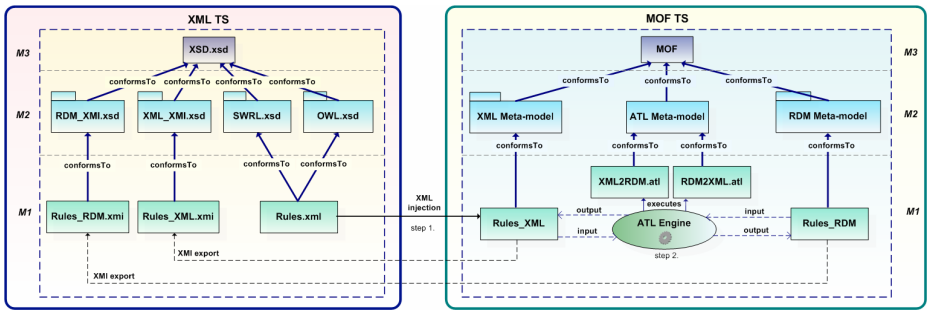

Figure 9. The first and second steps in the transformation scenario: the OWL/SWRL XML format into the instances of the RDM metamodel

```
<XML.Element xmi.id = 'a8' name = 'swrlx:individualPropertyAtom'
 value = ''>
   <XML.Element.children>
      <XML.Attribute xmi.id = 'a9' name = 'swrlx:property'
       value = 'hasUncle'/>
      <XML.Element xmi.id = 'a10' name = 'ruleml:var' value = ''>
         <XML.Element.children>
            <XML.Text xmi.id = 'a11' name = '#text' value = 'x1'/>
         </XML.Element.children>
      </XML.Element>
      <XML.Element xmi.id = 'a12' name = 'ruleml:var' value = ''>
         <XML.Element.children>
            <XML.Text xmi.id = 'a13' name = '#text' value = 'x3'/>
         </XML.Element.children>
      </XML.Element>
   </XML.Element.children>
</XML.Element>
```

Figure 10. The IndividualPropertyAtom from Figure 2 as an instance of the XML metamodel in its XMI format

Step 2. In this step, we transform the XML model (*Rules XML* from Figure 9) into the RDM-compliant model (*Rules RDM* from Figure 9). This transformation is done by using the ATL transformation named *XML2RDM.atl*. The output RDM model (*Rules RDM*) conforms to the RDM metamodel. An excerpt of the RDM model for the rule from

Figure 2 is shown in Figure 11. It is important to say that we can not exploit the standardized QVT transformation between UML and OWL from [ODM06], since our input rules are a combination of SWRL and OWL (i.e., RDM nad ODM).

In the XML2RDM.atl transformation, source elements from the XML metamodel are transformed into target elements of the RDM metamodel. The XML2RDM.atl transformation is done on the M1 level (i.e. the model level). 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 RDM 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 their M2 metamodels. This principle is well-know as metamodel-driven model transformations [Béz01].

```
<XMI xmi.version = '1.2' timestamp = 'Wed Jul 19 23:01:46 CEST 2006'>
  <!--...-->
    <RDM.IndividualVariable xmi.id = 'a1' name = 'x2'/>
    <RDM.IndividualVariable xmi.id = 'a2' name = 'x3'/>
    <RDM.IndividualVariable xmi.id = 'a3' name = 'x1'/>
    <RDM.Antecedent xmi.id = 'a4'>
      <RDM.Antecedent.containsAtom>
        <RDM.Atom xmi.idref = 'a5'/>
        <RDM.Atom xmi.idref = 'a6'/>
      </RDM.Antecedent.containsAtom>
    </RDM.Antecedent>
    <RDM.Consequent xmi.id = 'a7'>
      <RDM.Consequent.containsAtom>
        <RDM.Atom xmi.idref = 'a8'/>
      </RDM.Consequent.containsAtom>
    </RDM.Consequent>
    <RDM.Atom xmi.id = 'a5' name = 'IndividualPropertyAtom'>
      <!--...-->
    </RDM.Atom>
   <!--...-->
    <RDM.ODM.ObjectProperty xmi.id = 'a9' name = 'hasBrother' deprecated = 'false'
     functional = 'false' transitive = 'false' symmetric = 'false'
     inverseFunctional = 'false'
      complex = 'false'/>
   <!--...-->
    <RDM.ODM.Rule xmi.id = 'a13'>
       <RDM.ODM.Rule.hasConsequent>
          <RDM.Consequent xmi.idref = 'a7'/>
       </RDM.ODM.Rule.hasConsequent>
       <RDM.ODM.Rule.hasAntecedent>
          <RDM.Antecedent xmi.idref = 'a4'/>
       </RDM.ODM.Rule.hasAntecedent>
     </RDM.ODM.Rule>
  </XMI.content>
</XMI>
```

Figure 11. The RDM XMI representation of the rule shown in Figure 2

Step 3. The last step in this transformation process is the most important transformation where we transforming RDM model to R2ML model (Figure 12). This means that this step represents the transformation of the OWL/SWRL abstract syntax into the R2ML abstract syntax.

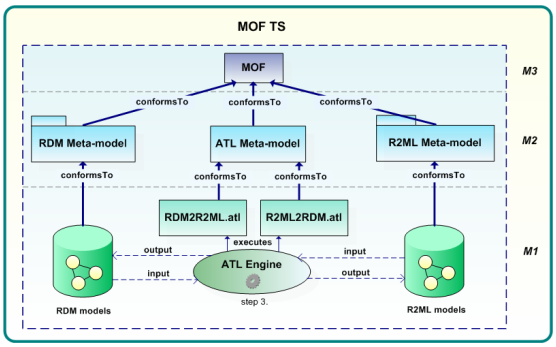

Figure 12. The transformation of the models compliant to the RDM metamodel into the models compliant to the R2ML metamodel

This transformation step is fully based on the conceptual mappings between the elements of the RDM and R2ML metamodel. In Table 1, we give an excerpt of mappings between the SWRL XML Schema, XML metamodel, RDM metamodel and R2ML metamodel. Due to the size limitation for this paper, we selected a few characteristic examples of mapping rules. The current mapping specification contains 26 rules.

Table 1. An excerpt of mappings between the OWL/SWRL XML schema, XML metamodel, RDM metamodel, and the R2ML metamodel

OWL/SWRL	XML metamodel	RDM metamodel	R2ML metamodel
individualPropertyAtom	Element name = 'swrlx:individualProperty Atom'	Atom	Universally Quantified Formula
OneOf	Element name = 'owlx:OneOf'	EnumeratedClass	Disjunction
var	Element name = 'ruleml:var'	IndividualVariable	ObjectVariable
sameIndividualAtom	Element name = 'swrlx:sameIndividualAtom'	Atom	EqualityAtom
maxcardinality	Element name = 'owlx:maxcardinality'	MaxCardinality Restriction	AtMostQuantified Formula

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 RDM metamodel by using ATL, and then to instances of R2ML metamodel. In the second step (the XML2RDM transformation), we created some relatively complex helpers, although this transformation is rather straightforward. One significantly problem that we had to deal with in this transformation was the creation of unique elements in the output RDM model from multiple elements in the input XML model that referred to the same element (e.g., the Variable element). When the XML ruleml:var element is transformed to the RDM InvididualVariable element, we use the helper getDefaultVariable. The helper always returns the same input ruleml:var element with the given name, because we always want to transform it uniquely to the same output element. When the creation of this variable is called for the first time, the output IndividualVariable is created, but every other time when it is called, we just got a reference to the already created IndividualVariable. The helper operation getDefaultVariable first need to find in which rule the input variable is located, so that we can have multiple rules in the input file. Finding rules with the ruleml:var element is done by the ATL getRuleForElement helper, which tests every input rule element (e.g. his sub elements) whether they have the input variable located somewhere in their child elements. When a rule is located, we always get the first variable with the given name for children elements of that rule. This is done by the ATL allSubElements attribute helper, which recursivelly goes to the model graph and returns all sub elements (flatten) for the called element.

The actual transformation between the RDM metamodel and elements of the R2ML metamodel are defined as a sequence of rules in the ATL language (*RDM2R2ML.atl* in Figure 12). These rules use additional helpers 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. ATL in fact instantiate the R2ML metamodel (M2 level), i.e. it creates R2ML models. In this ATL transformation, we use so-called ATL matched rules. A matched rule matches a given type of a source model element, and generates one or more kinds of target model elements. Figure 13 gives an example of a matched rule which is, in fact, an excerpt of the *RDM2R2ML.atl* transformation for the IndividualPropertyAtom class of the RDM metamodel.

```
rule IndividualPropertyAtom2UniversallyQuantifiedFormula {
        from i : RDM!Atom (
                        i.name = 'IndividualPropertyAtom'
                )
        to
                o : R2ML!UniversallyQuantifiedFormula (
                                variables <- i.terms,
                                formula <- refpropat

                        ),
                refpropat : R2ML!ReferencePropertyAtom (
                                referenceProperty <- refprop,
                                subject <- i.terms->first(),
                                object <- i.terms->last()
                        ),
                refprop : R2ML!ReferenceProperty (
                                refPropertyID <- i.hasPredicateSymbol.name
                        )
}
```

Figure 13. An excerpt of the ATL transformation: A matched rule that transforms an RDM IndividualPropertyAtom to an R2ML UniversallyQuantifiedFormula

As an illustration why we need to use ATL helpers, let us consider the example of transforming RDM ClassAtom with Class as a predicate symbol to R2ML ObjectClassificationAtom. However, when transforming an RDM Class, we do not have a direct reference to the RDM ClassAtom. Since more than one RDM ClassAtom may refer to the same RDM Class, we have to find the exact ClassAtom. To solve this situation, we use an ATL helper that returns a reference to the needed ClassAtom.

For example, the R2ML model shown in Figure 14 is the output of the RDM to R2ML transformation for the RDM model (IndividualPropertAtom) given in Figure 11. This is actually the end of the transformation between abstract syntax of OWL/SWRL and R2ML.

```
<R2ML>
  <!--...-->
  <R2ML.Formulas.UniversallyQuantifiedFormula xmi.id = 'a12'>
    <R2ML.Formulas.QuantifiedFormula.formula>
      <R2ML.RelAt.ReferencePropertyAtom xmi.id = 'a13'
       isNegated = 'false'>
        <R2ML.RelAt.ReferencePropertyAtom.object>
          <R2ML.BasContVoc.ObjectVariable xmi.idref = 'a11'/>
        </R2ML.RelAt.ReferencePropertyAtom.object>
        <R2ML.RelAt.ReferencePropertyAtom.referenceProperty>
          <R2ML.BasContVoc.ReferenceProperty xmi.idref = 'a14'/>
        </R2ML.RelAt.ReferencePropertyAtom.referenceProperty>
        <R2ML.RelAt.ReferencePropertyAtom.subject>
          <R2ML.BasContVoc.ObjectVariable xmi.idref = 'a6'/>
        </R2ML.RelAt.ReferencePropertyAtom.subject>
      </R2ML.RelAt.ReferencePropertyAtom>
    </R2ML.Formulas.QuantifiedFormula.formula>
    <R2ML.Formulas.QuantifiedFormula.variables>
      <R2ML.BasContVoc.ObjectVariable xmi.idref = 'a6'/>
      <R2ML.BasContVoc.ObjectVariable xmi.idref = 'a11'/>
    </R2ML.Formulas.QuantifiedFormula.variables>
  </R2ML.Formulas.UniversallyQuantifiedFormula>
  <!--...-->
</R2ML>
```

Figure 14. An excerpt of the R2ML XMI representation of the RDM rule shown in Figure 11

Note also that an additional step (besides the three ones explained in this section) can be to transform rules from R2ML into the R2ML XML concrete syntax. For example, the R2ML model shown in Figure 14 can now be transformed to elements of the XML metamodel (R2ML2XML), and then automatically transformed to the R2ML XML concrete syntax by using the XML extractor that is included the ATL engine. The result of the XML extraction of the R2ML model (from Figure 14) is shown in Figure 15.

```
<r2ml:Implication>
  <r2ml:consequent>
    <r2ml:UniversallyQuantifiedFormula>
      <r2ml:ObjectVariable r2ml:name="x1"/>
      <r2ml:ObjectVariable r2ml:name="x3"/>
      <r2ml:ReferencePropertyAtom r2ml:refPropertyID="hasUncle">
        <r2ml:subject>
          <r2ml:ObjectVariable r2ml:name="x1"/>
        </r2ml:subject>
        <r2ml:object>
          <r2ml:ObjectVariable r2ml:name="x3"/>
        </r2ml:object>
      </r2ml:ReferencePropertyAtom>
    </r2ml:UniversallyQuantifiedFormula>
  </r2ml:consequent>
  <!--...-->
</r2ml:Implication>
```

Figure 15. An excerpt of the R2ML XML representation of the SWRL rule shown in Figure 2

5 Mapping R2ML Integrity constraints to OCL

In previous section, we have shown how one can get a valid R2ML model from any RDM model. The final objective of this section is to explain the transformation of R2ML models (rules) into OCL models [OCL06]. To do so, we have defined mappings for transforming elements of the OCL metamodel into elements of the R2ML metamodel (see Figure 16). OCL has its own abstract and concrete syntax, and for transformation process we use its abstract syntax defined in the form of a MOF-based metamodel [OCL06]. Since the R2ML and OCL metamodels are both located in the MOF technical space and there is an metamodel for OCL defined in the OCL specification, the transformation by ATL is straightforward in terms of technological requirements, i.e. we do not have to introduce an additional metamodel like we have done with RDM.

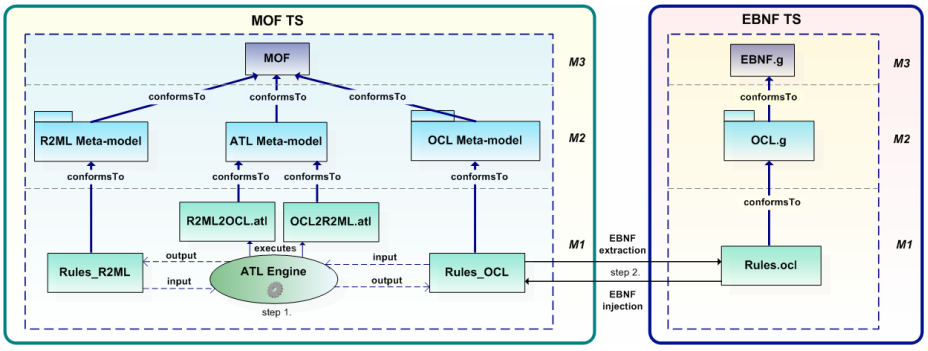

Figure 16. The transformation scenario: R2ML metamodel to and from OCL metamodel, with EBNF injection/extraction of OCL code

Step 1. We transform an R2ML model (*Rules_R2ML* from Figure 16) into an OCL model (*Rules_OCL*) by using an ATL transformation named *R2ML2OCL.atl*. The output OCL model (*Rules_OCL*) conforms to the OCL metamodel. In Table 2, we give an excerpt of mappings between the R2ML metamodel and OCL metamodel on which this ATL transformation is based. The current version of the transformation contains 39 mapping rules.

For element of the R2ML metamodel, an instance of the OCL metamodel is created in the model repository. The ATL transformation is done for classes, attributes, and references. For this transformation we have used integrity and derivation rules of the R2ML metamodel in its current version (0.3). For the R2ML model (rule) shown in Figure 14, we get an OCL model

represented in the OCL XMI concrete syntax given in Figure 17. The figure shows an OCL iterator expression (forAll) that has Boolean as the return type, and an operation call expression as its source. That operation call expression then calls an operation, which has the set type as its return type and a class as its source.

Table 2. An excerpt of mappings between the R2ML metamodel, OCL metamodel, and OCL code

R2ML metamodel	OCL metamodel	OCL code
Conjuction	OperationCallExp referredOperation (name = 'and')	Operand and Operand
Implication	OperationCallExp referredOperation (name = 'implies')	Expression implies Expression
AttributionAtom	OperationCallExp referredOperation (name = '=') PropertyCallExp (subject)	Subject.attribute = value
ObjectVariable	Variable	Variable name
EqualityAtom	OperationCallExp referredOperation (name = '=')	Expression1 = Expression2 and Expression2 = Expression3, ...
RoleFunctionTerm	PropertyCallExp referredProperty (name = 'property') source Variable	Variable.property
AtMostQuantifiedFormula	OperationCallExp referredOperation (name = '<=') argument maxvalue	Expression <= maxvalue

Step 2. Because the OCL concrete syntax is located in the EBNF technical space, we need to get an instance of the OCL metamodel (abstract syntax) into EBNF technical space. There are three possible solutions to this problem. The first one is creating another transformation from the OCL metamodel to the ATL metamodel (which extends a modified standard OCL metamodel), e.g., to its query expression, and then by using the tool "Extract ATL model to ATL file" included in ATL, we can get the OCL code. However, the disadvantage of this solution is the creation of a new transformation from the OCL metamodel to the ATL metamodel, which is a time consuming and more general task overcoming the scope of our research. The second solution is to create ATL query expression on OCL metamodel elements, which will then generate OCL code in a file. The disadvantage of this solution is that this solution can not be used for all OCL metamodel elements, because it will be complex to define all mappings. The third solution is to use a TCS (Textual Concrete Syntax) interpreter [Jou06] based on the TCS syntax definition of OCL. A TCS represents a domain specific language for the specification of textual concrete syntaxes in MDE, and it is a part of the ATL tool suite. It can be used to parse text-to-model and to serialize model-to-text. The concrete syntax of OCL has been implemented in TCS according to the syntax specified in [OCL06]. Because of the specific requirements of TCS (and ANTLR 2, which TCS uses for generating parsers), we

have needed to adapt the standard OCL metamodel, that is, we adapted its EnhancedOCL package. We have added the OperatorCallExp element for expressions, which uses these two operators: CollectionOperationCallExp for calling operations on collections and the Iterator class which is a specific type of Iterator Variable. We have also added classes for every primitive type such as StringType and IntegerType. For defining "def" and "inv" elements of OCL, we have introduced the DefOclModuleElement class for "def" elements (operations, class OclOperation and attributes, class OclProperty) and the Invariant class for "inv" elements. These two types inherit the abstract class OclModuleElement. OclModuleElement contains the context definition of the type OclContextDefinition, while OclContextDefinition contains Class as the context element. The OclModule class is added as the top-most element in order to capture different OclModuleElement's. In the future, we will add support for these OCL constructs: derive, postcondition, and precondition.

```
<OCL.EssentialOCL.IteratorExp xmi.id = 'a11' name = 'forAll'>
   <UML.TypedElement.type>
        <OCL.EssentialOCL.BooleanType xmi.id = 'a4'/>
   </UML.TypedElement.type>
   <OCL.EssentialOCL.CallExp.source>
      <OCL.EssentialOCL.OperationCallExp xmi.id = 'a18'>
           <UML.TypedElement.type>
             <OCL.EssentialOCL.SetType xmi.idref = 'a19'/>
           </UML.TypedElement.type>
           <OCL.EssentialOCL.CallExp.source>
             <UML.Class xmi.idref = 'a17'/>
           </OCL.EssentialOCL.CallExp.source>
           <OCL.EssentialOCL.OperationCallExp.referredOperation>
             <UML.Operation xmi.idref = 'a25'/>
               <!--...-->
           <OCL.EssentialOCL.OperationCallExp.referredOperation>
      </OCL.EssentialOCL.OperationCallExp>
   </OCL.EssentialOCL.CallExp.source>
   <!--...-->
</OCL.EssentialOCL.IteratorExp>
```

Figure 17. An excerpt of the OCL XMI representation of the R2ML model shown in Figure 14

Figure 18 shows the mapping from the OCL metamodel (in the KM3 format [JB06]) to its corresponding TCS. Using the TCS interpreter and defined mapping rules (as in Figure 18 for element Class), we have done an EBNF extraction from the OCL model to the OCL code. Our starting example shown in Figure 1 is actually the OCL code that represents the OCL model from Figure 17. This OCL code is also the transformed SWRL rule from Figure 2.

In the opposite direction, from OCL to R2ML, for Step 1 (the ENBF injection), we also have two solutions. The first one is to use the OCL Parser from the Dresden OCL Toolkit [DOT06] for parsing OCL code and creating OCL model from it. This solution needs a predefined UML model (in the UML XMI format) as the input on which OCL code is defined, and this is not what we want, because for the input we want only OCL code without the UML model on which it is defined. The second solution is by using TCS for creating model from code. Since we used this solution for generating code from model and it supports generation of OCL code without UML model, we decided to us it, for this direction.

The generation of the ANTLR-based OCL Parser by using TCS is done by following steps:

1. By using ATL's EBNF Injector class and the TCS metamodel, we get a TCS model (an instance of the TCS metamodel) from the OCL textual concrete syntax.
2. The TCS2ANTLR transformation is used to transform to the TCS-based model (an instance of the TCS metamodel) for the OCL textual concrete syntax into a model compliant to the ANTLR metamodel.

3. The ANTLR model which is transformed from TCS model for the OCL textual concrete syntax (obtained in the previous step) along with the TCS definition of ANTLR (ANTLR.tcs) and ATL's EBNF Extractor is used to generate a grammar file of OCL (i.e., OCL.g).
4. Finally, ANTLR is called to create OCLLexer and OCLParser classes (i.e., lexer and parser of for parsing OCL text-based code) from the generated grammar file.

When the OCL model is generated form the OCL code, we use OCL2R2ML.atl transformation for transforming this model into the corresponding R2ML model (as shown in Figure 16).

```
package OCL {
   //...
      class Class extends Type {
           reference ownedOperation[*] : Operation;
           reference ownedAttribute[*] : Property;
           attribute isAbstract : Boolean;
      }
   //...
}
```

a) OCL metamodel

```
syntax OCL {
   //...
      template Class context
           : (isAbstract ? "abstract") "class" name
                "{"
                     ownedOperation ownedAttribute
                "}"
           ;
   //...
}
```

b) OCL Textual Concrete Syntax

Figure 18. The transformation of elements of the OCL metamodel into its corresponding OCL textual concrete syntax (TCS)

6 Conclusions

The presented approach to interchanging OWL/SWRL and UML/OCL is based on the pivotal (R2ML) metamodel that addresses the complexity of mappings between two languages, which contain many diverse concepts. In this paper, we have not focused only on mapping rules between OWL/SWRL and R2ML and between UML/OCL and R2ML [WGL*06], but we have also described the whole transformation process based on the use of the ATL model transformation language and several other XML Schemas and MOF metamodels. Besides bridging two OWL/SWRL and UML/OCL, the use of R2ML allows us to reuse (i.e., apply on OWL/SWRL and UML/OCL) the previously implemented transformations between R2ML and R2ML XML concrete syntax, F-Logic, Jess, and RuleML, thus further interchanging OWL/SWRL and UML/OCL.

In this paper, we have shown how to transform rules between SWRL and OCL by using R2ML as an intermediary general rule markup language. To do so, we have created a number of transformations. The first group of transformations is the transformations between the XML and MOF technical spaces, which are done by using the XML Injector and Extractor of the ATL model transformation tool suite. Namely, we need to proved mappings between MOF-based and XML-based representations of the following languages: the SWRL XML concrete syntax and the RDM (XML2RDM and RDM2XML) as a MOF-based metamodel of SWRL; and the R2ML XML Schema and the R2ML MOF-based metamodel (XML2R2ML and R2ML2XML). The second group is the transformations between the RDM, R2ML and OCL metamodels (i.e., the following transformations RDM2R2ML, R2ML2RDM, R2ML2OCL and OCL2R2ML). In the current implementation, we support the translation between OCL invariants and R2ML integrity rules. The third group of transformations is done by using the

EBNF Injector and Extractor of the ATL tools suite. We have used this in order provide mappings between the EBNF technical space and the MOF technical space, that is, to transform between the OCL concrete syntax to the OCL metamodel. The current version of transformations supports OCL invariant's and def's. The practical implementation is is done by using the OCL Textual Concrete Syntax and the subsequently-generated OCL Parser and Lexer.

The presented research is a next step towards the further reconciliation of MDA and Semantic Web languages, and hence continues the work established by the OMG's ODM specification that only addressed mappings between OWL and UML, while we extended it on the accompanying rule languages, i.e., SWRL and OCL.

We are now in the phase of the evolution of the results of the translation between OCL and SWRL via the R2ML language. In our future publications, we are going to report on transformation implementation in more detail and evaluation results. We also plan to support the translation between the UML parts related to classes and the R2ML vocabulary and between the ODM metamodel and the R2ML vocabulary. Besides integrity rules support in the present transformations, we are going to support translations of derivation rules of OCL and SWRL. We will also extend our rule transformation framework in order to support other OMG's specification covering rules, i.e., the ones for business and production rules.

Bibliography

[ATL06] ATLAS Transformation Language (ATL), http://www.sciences.univ-nantes.fr/lina/atl, 2006.

[Béz01] J. Bézivin. From Object Composition to Model Transformation with the MDA, *In Proc. of the 39th Int. Conf. and Exh. on Tech. of OO Lang. and Sys.*, pp. 350-355, 2001.

[BH06] S. Brockmans, P. Haase. A Metamodel and UML Profile for Rule-extended OWL DL Ontologies - A Complete Reference, Universität Karlsruhe (TH) - Technical Report, 2006.

[Cla78] K. L. Clark. Negation as Failure, *In Gallaire, H., and Minker, J.* (eds.), *Logic and Data Bases*, Plenum Press, NY, pp. 293-322, 1978.

[DOT06] Dresden OCL Toolkit, Technische Unversität Dresden, Software Engineering Group, http://dresden-ocl.sourceforge.net, 2006.

[FSS03] K. Falkovych, M. Sabou, H. Stuckenschmidt, UML for the Semantic Web: Transformation-based approaches, *Knowledge Transformation for the Semantic Web, eds., Frontiers in Artificial Intelligence and Applications, vol. 95*, IOS, Amsterdam, pp. 92-106, 2003.

[GDD06] D. Gašević, D. Djurić, V. Devedžić. Model Driven Architecture and Ontology Development, Springer, Berlin, 2006.

[GL88] M. Gelfond, V. Lifschitz. The stable model semantics for logic programming, *In Proc. of ICLP-88*, pp. 1070-1080, 1988.

[GL91] M. Gelfond, V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases, *New Generation Computing*, vol. 9, pp. 365-385, 1991.

[HEP03] M. Hori, J. Euzenat, F. P. Patel-Schneider. OWL Web Ontology Language XML Presentation Syntax, W3C Note, 2003.

[HPB*04] I. Horrocks, P. F. Patel-Scheider, H. Boley, S. Tabet, B. Grosof, M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C Member Submission, http://www.w3.org/Submission/SWRL/, 2004.

[JB06] F. Jouault, J. Bézivin. KM3: a DSL for Metamodel Specification, *In Proceedings of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems*, Bologna, Italy, pp. 171-185, 2006.

[JG05] J. Jovanović, D. Gašević. XML/XSLT-Based Knowledge Sharing, *Expert Systems with Applications*, Vol. 29, No. 3, 2005, pp. 535-553.

[Jou06] F. Jouault. TCS: Textual Concrete Syntax, *In Proceedings of the 2nd AMMA/ATL Workshop ATLAS group (INRIA & LINA)*, Nantes, France, 2006.

[KC04] G. Klyne, J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Rec. 10 February 2004, http://www.w3.org/TR/rdf-concepts/.

[MGG*06] M. Milanović, D. Gašević, A. Giurca, G. Wagner, S. Lukichev, V. Devedžić. Validating Rule Language Metamodels with the Help of Model Transformations," *2nd Int. Conf. of Rules and Rule Markup Languages for the Semantic Web*, Athens, USA, 2006 (submitted).

[OCL06] OMG OCL. Object Constraint Language, *OMG Specification, Version 2.0, formal/06-05-01*, http://www.omg.org/docs/formal/06-05-01.pdf, 2006.

[ODM06] OMG ODM. Ontology Definition Metamodel, 6th Revised Submission, 2006.

[PSH04] P. F. Patel-Schneider, I. Horrocks. OWL Web Ontology Language Semantic and Abstract Syntax, http://www.w3.org/2004/OWL, 2004.

[R2ML06] R2ML. *The REWERSE I1 Rule Language*, http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/6, 2006.

[RIF06] Rule Interchange Format (RIF) use cases and requirements, W3C Working Draft, http://www.w3.org/TR/rif-ucr/, 2006.

[UML06] OMG, Unified Modeling Language (UML) 2.0, Docs. formal/05-07-04 & formal/05-07-05, 2006.

[Wag03] G. Wagner. Web Rules Need Two Kind of Negations, *In Proc. of the Workshop on Principles and Practice of Semantic Web Reasoning*, pp. 33-50, 2003.

[WGL05] G. Wagner, A. Giurca, S. Lukichev. R2ML: A General Approach for Marking-up Rules, *In Proceedings of Dagstuhl Seminar 05371, in F. Bry, F. Fages, M. Marchiori,H. Ohlbach (Eds.) Principles and Practices of Semantic Web Reasoning*, http://drops.dagstuhl.de/opus/volltexte/2006/479/, 2005.

[WGL*06] G. Wagner, A. Giurca, S. Lukichev, G. Antoniou, C. V. Damasio, N. E. Fuchs. Language Improvements and Extensions, *REWERSE I1-D8 deliverable*, http://rewerse.net/deliverables.html, 2006.