UDC 004.423.2, DOI: 10.2298/csis0902047M

Model Transformations to Bridge 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] GOOD OLD AI Network, FON-School of Business Administration, University of Belgrade, Serbia, Jove Ilića 154, 11000 Belgrade
milan@milanovic.org, devedzic@fon.rs
[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 presents a solution to bridging the abstract and concrete syntax of a Web rule languages by using model transformations. Current specifications of Web rule languages such as Semantic Web Rule Language (SWRL) or RuleML define their abstract syntax (e.g., metamodel) and concrete syntax (e.g., XML schema) separately. Although the recent research in the area of Model-Driven Engineering (MDE) 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 the REWERSE I1 Rule Markup Language (R2ML), SWRL, and Object Constraint Language (OCL), whose abstract syntax is defined by using metamodeling, while their textual concrete syntax is defined by using either XML/RDF schema or Extended Backus-Naur Form (EBNF) syntax. We bridge this gap by a bi-directional transformation defined in a model transformation language (ATLAS Transformation Language, ATL). This transformation allowed us to discover a number of issues in both web rule language metamodels and their corresponding concrete syntax, and thus make them fully compatible. This solution also enables for sharing web rules between different web rule languages.

Keywords: rules, MDE, syntax, transformations, languages, R2ML, OCL, SWRL.

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 [29]) that are used for reasoning over Semantic Web ontologies. The main issue to be solved is the type (e.g., open or 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 and concrete syntax is an unavoidable part of the language definition. An important characteristic of Semantic Web rule languages is that they do primarily not deal with interchange of rules between various types of rule languages 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) [23], while the most known proposals are REWERSE Rule Markup Language (R2ML) [57] and RuleML [11]. The primary result expected from this research stream is an XML-based concrete syntax for defining 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 it is also important to have a well-designed abstract syntax [12].

Although the work on both types of Web rules is very important, even a further impact will be achieved once we define technique, which will allow for the development of Web applications that are using Web rules. Model-driven engineering (MDE) is a promising approach to this problem, where, for example, Object Constraint Language (OCL) can be used to model and specify different aspects of Web applications [24]. In our particular case, Web rules can be specified by using OCL, which are used together with UML [46]. In UML, various model elements such as classes or state machines can be annotated by logical constraints defined by using OCL. In this way, UML models constrained by OCL expressions are more accurate and complete. OCL is today used in a number of tools, and it is accepted as a standard by the OMG (Object Management Group) [44]; it can be also used to define constraints on MOF (Meta Object Facility)-based metamodels [42]. The OCL 2.0 specification [0] explicitly defines an abstract and concrete syntax of the language, that is, a MOF-based metamodel and a textual concrete syntax, respectively.

The current practice of rule language design demonstrates that there is usually no tool support that connects an abstract syntax with a concrete syntax of a rule language. The main reason of having such a tool is to enable for the validation of rule expressions encoded in various concrete (e.g., XML-based or EBNF-based) syntax w.r.t. the abstract syntax. Not should a rule only follow a concrete syntax of the language, but it also must be valid in terms of the abstract syntax. An optimal situation will be, if we can provide a transformation between a concrete syntax and an abstract syntax. On the other hand, such a transformer may also help us check the correctness and appropriateness of the abstract syntax based on the expressions encoded in the concrete syntax.

In this paper, we try to address this problem of bridging the gap between an abstract and concrete syntax of Web rule languages. In our study we analyze the R2ML [57], SWRL [29] and OCL languages whose concrete syntax is

defined by using XML, Resource Definition Framework (RDF), and EBNF, respectively. Since R2ML and OCL languages leverage the benefits of a new software engineering discipline MDE [9], the abstract syntax of R2ML and OCL is defined by a metamodel that is specified by using the Meta-Object Facility (MOF) metamodeling language [42]. SWRL specification [29] does not propose an abstract syntax defined by a metamodel, but it is proposed in a research literature [14]. In [43] is said that a metamodel specifies syntax of a modeling language by omitting some aspects of the graphical or textual appearance of the language, such as geometric shapes or punctuation. For example, a metamodel might have an element for kinds of rules and another for logical formulas, but no mention of how generalization appears in a graphical or textual syntax. This is sometimes called “abstract syntax”, as distinguished from “concrete syntax”, which includes the detailed graphical or textual appearances. The use of the MDE approach for defining abstract syntax of rule languages enables us to have first leverage metamodels as means for checking validity of concrete expressions in rule languages. Moreover, we can define more advanced constraints over metamodels by using a constraint language such as OCL. This enables us to leverage a richer set for defining well-formedness rules of Web rule languages. This is a more expressive mechanisms comparing to the XML syntax definition mechanisms (e.g., XML scheme), which do not have this kind of mechanism.

Along with their abstract syntax, R2ML, SWRL and OCL have concrete syntax that has been developed for encoding rules by domain experts in various tools. The R2ML concrete syntax is defined by an XML Schema, the SWRL concrete syntax is defined by XML and RDF Schemas, while the OCL concrete syntax is defined by an EBNF textual concrete syntax. However, there is no solution that enables mappings and transforming concrete syntax of these languages into their compliant metamodels. This gap between the rule languages metamodels and their concrete syntax causes the following problems:

1. Rules represented in the R2ML, SWRL and OCL concrete syntax require a means in order to be stored in MOF-based model repositories. This will enable their validation w.r.t. their metamodels. In order to be automatically verified, constraints for checking models coherence must be written in a language for which automatic translation to an executable form is possible. The well known OCL solution is used here.
2. The rule language metamodels cannot be instantiated based on rules encoded in their concrete syntax with the present tools, and thus their metamodels cannot be validated with real-world rules.

In this paper, we demonstrate how MDE tools and model transformation languages can be used to overcome R2ML, SWRL and OCL issues listed above. To describe this approach, we first give a short introduction into the main initiatives for sharing Web rules and into the basics of MDE concepts that our solution is based on. In Section 3, we describe the rule languages, i.e., abstract and contract syntax of R2ML, SWRL, and OCL. In Section 4, we explain our conceptual solution to bridging between each rule language’s

abstract and concrete syntax, and also we explain our implementation approach, while in Section 5 we give some general and technically-specific best practices to bridge between abstract and concrete syntax of rule languages. In Section 6, we report on the main experience that we obtained when implementing transformations and consequences on the R2ML, SWRL and OCL metamodels, while we highlight the related work on rule language transformations in Section 7. We conclude in Section 8.

2. Background

This section introduces the basic concepts and technologies that are used in our solution to bridging the gap between abstract and concrete syntax of Web rule languages. Namely, we introduce the W3C's Rule Interchange Format initiative and fundamental concepts of MDE such as metamodeling, Model-Driven Architecture (MDA), XMI, technical spaces, and model transformations.

2.1. Rule Interchange Format

Rule Interchange Format (RIF) [23] is the W3C's initiative that aims at defining an intermediary language between various rule languages. Its goal is not to provide a formally defined semantic foundation for reasoning on the Web such as OWL for ontologies. The current state of this initiative is that it defines a set of requirements and use cases for sharing rules on the Web. However, there is no official submission to this initiative yet.

RuleML is a language that tends to be a proposal for RIF. RuleML is a markup language for publishing and sharing rule bases on the World Wide Web [27]. RuleML builds a hierarchy of rule sublanguages upon XML, RDF, XSLT, and OWL. The current RuleML hierarchy consists of derivation (e.g., SWRL and First Order Logic-FOL), integrity, reaction, transformation and production rules (e.g., Jess). RuleML is based on Datalog. RuleML rules are defined in the form of an implication between an antecedent and consequent, with the meaning whenever the logical expression in the antecedent holds, then the consequent must also hold. However, an important constraint of RuleML is that it cannot fully represent all the constructs of various languages such as Object Constraint Language (OCL) [44] or SWRL [29] and it cannot satisfy many RIF requirements yet. It is important to mention that is a proposal for the RuleML abstract syntax [12] by means of a metamodel, but there has not been any solution allowing for bridging between that abstract syntax and the concrete syntax of RuleML.

In this paper, we use REWERSE I1 Rule Markup Language (R2ML) as a language that covers all RIF requirements for sharing rules, and hence solves some of the RuleML constraints. We describe its abstract and concrete syntax in Section 3. As R2ML has separately defined abstract and concrete syntax,

we demonstrate how model transformations can be used to bridge between its abstract and concrete syntax.

2.2. Model Driven Engineering

MDE is a new software engineering discipline in which the process heavily relies on the use of models [9] [18], while the OMG's MDA [40] is considered a possible metamodeling architecture enabling the use of the MDE principles. The core concept in MDE is model. A model defined is a set of statements about some system under study [53]. 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 [53].

MDE 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 or Ontology Definition Metamodel, ODM [45]) by using metamodeling languages such as MOF;
- M3 layer or metametamodel layer where the only metamodeling language is defined (i.e. MOF) by itself [42].

The relations between different MDE layers can be considered as instance-of or conformant-to, which means that a model is an instance of (i.e., conformant to) a metamodel, and a metamodel is an instance of (i.e., conformant to) 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 processed in the same way, for example, by using the same API. An example of such APIs are Java Metadata Interface (JMI) [32] and Eclipse Modeling Framework (EMF)[1] that enables the implementation of a dynamic, platform-independent infrastructure to manage the creation, storage, access, discovery, and exchange of MOF-based metadata. The most comprehensive implementation of JMI is NetBeans' Metadata Repository (MDR)[2] that contains implementation of a MOF-based repository including persistent storage mechanism for storing the MOF-based metadata. An important feature of MOF is that it inherits the graphical syntax of UML, that is, MOF is a subset of UML class models, so that an abstract syntax of a modeling language defined by MOF can be represented by using UML class diagrams.

We should mention also the well-formedness rules in terms of metamodeling architectures. Typically, they are defined on the metamodel level, by using a combination of a metamodeling language (MOF) and a constraint language

[1] http://www.eclipse.org/emf/

[2] NetBeans MDR: http://mdr.netbeans.org/

(OCL). The constraints are defined over metamodels, most commonly, in a form of integrity constraints such as OCL invariants. These rules regulate a set of valid models (i.e., expressions of a modeling language), and only those models that are fully compliant with such well-formedness rules are considered well-formed. For example, well-formedness of R2ML is defined by using MOF and OCL, while all R2ML rules (i.e., models) must comply to the well-formedness rules of R2ML

Note also that for each MOF-based metamodel and model, one can automatically generate their corresponding XML schema (so called XML Metadata Interchange–XMI) by following the OMG's XMI specification [48]. This enables sharing MOF-based models and metamodels among different MOF-based model repositories. MDR implements functionalities for exporting/importing XMI documents compliant with various MOF-based metamodels (e.g., the R2ML metamodel).

2.3. Technical spaces

Although MDE principles for defining modeling languages seems quite promising, the reality is, that languages 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 [6], where a technical space (TS) is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities [34]. Although some technical spaces are difficult to define, they can easily be recognized (e.g., XML, MDA, databases, and Semantic Web). In the case of the problem analyzed in this paper, we have to bridge between two technical spaces – the MDE technical space (as we assume that abstract syntax of the rule languages is defined by using metamodeling) and technical spaces, such as XML, RDF and EBNF that are used for defining concrete syntax of rule languages.

2.4. Model Transformations

Model transformations represent the central operation for handling models in the MDA [0]. Model transformations are the process of producing one model from another model of the same system [40]. In fact, a model transformation means converting an input model, which conforms to one metamodel, to another model, which conforms to another metamodel (see Fig. 1). This conversion is done by defining rules that match and/or navigate elements of source models resulting in the production of elements of the target model. The transformation itself is a model, which conforms to some transformation metamodel [7]. Languages for defining model transformations are generally declarative, and include approaches based on relations [1], or those based on patterns of logical constraints [16]. In November 2005, the OMG published the final specification of the MOF2 Query View Transformation (QVT) standard [47]. MOF2

QVT is defined by a MOF-based metamodel (in the similar way as the R2ML metamodel is defined), and thus QVT is located on the M2 layer of the MDA (i.e., transformation language in Fig. 1).

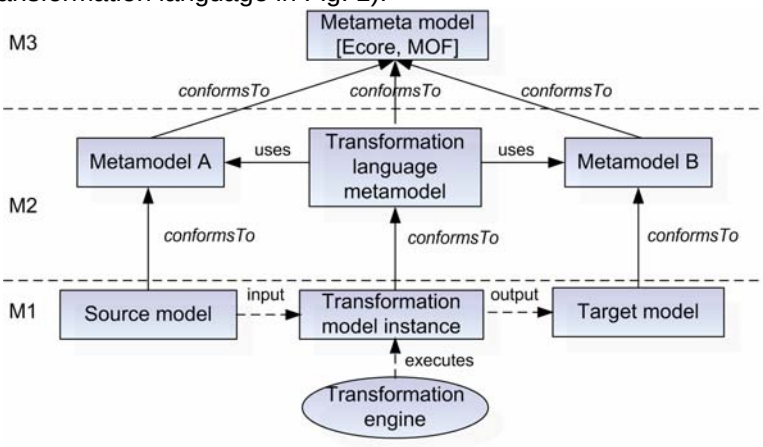

Fig. 1. An overview of model transformations

QVT may be used for many kinds of model transformations such as from object model to data model or from business model to object model. Although it is very important to have a standard such as MOF2 QVT, it is equally important to use an appropriate tool that allows us to represent models and metamodels being transformed between different technical spaces (e.g., MOF and XML). In our research, we have decided to use ATLAS Transformation Language (ATL) [3] as the primary language and tool for model transformations based on the following arguments: an open-source software, the biggest user community, a solid developer support, a rich knowledge base of model transformation examples and projects, and a very mature support for technical spaces (e.g., XML, EMF, and MOF).

ATL is a hybrid (i.e., declarative and imperative) transformation language, and it is based on the OMG OCL norm [44] for both its data types and its declarative expressions. ATL and QVT share some common features, as they initially shared the same set of requirements defined in QVT Request for proposals [47]. However, the actual ATL implementation is different from the QVT standard, although the QVT standard defines requirements primarily for tools, and not for languages. ATL is implemented as an Eclipse plug-in and integrates the notion of technical spaces. Although mainly intended to deal with MDA models, this framework should also handle other kinds of models from different technical spaces (e.g., Java programs, XML documents, and databases). The ATL Eclipse perspective provides tools for importing/exporting various XML formats into/from ATL's MOF based model repository. This is implemented in the form of the following tools:

- *XML injection*: takes an XML file as the input and produces its equivalent model that is conformant to the XML metamodel defined in MOF (the next section gives more details about this metamodel). We should also mention

that XML injection can be used for RDF too, because we consider that RDF can be represented by XML (and in fact, one of the most commonly used RDF concrete syntax is XML-based), but in general case RDF enables more freedom in syntax defining of RDF expressions.

- *XML extraction*: produces an XML file from the model conformant to the metamodel defined in either Ecore or MOF. We should also say here that as for XML injection, XML extractor can be used for RDF files, too.
- *EBNF injection:* takes a textual file (written in the textual concrete syntax of a rule language) as the input and produces an equivalent model that is conformant to the rule language's metamodel defined in either Ecore or MOF.
- *EBNF extraction:* produces a textual file (written in the textual concrete syntax of a rule language) from the model conformant to the rule language's metamodel defined in either Ecore or MOF.

In addition, ATL toolkit also includes TCS (Textual Concrete Syntax) tools [30]. TCS represents domain specific language (DSL) for defining textual concrete syntaxes in MDE. As a part of ATL toolkit it can be used for parsing text-to-model and serialization model-to-text. In our work presented in this paper, we used TCS for EBNF injection/extraction.

3. Rule Languages Representations

In this section, we describe briefly the three rule languages under study (i.e., R2ML, SWRL and OCL), by means of their abstract and concrete syntax.

3.1. R2ML Metamodel and R2ML XML Schema

This section is devoted to the description of the R2ML language [57][59] 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 and derivation rules in this section. For the complete definition of the R2ML metamodel and R2ML XML schema, we refer readers to [50].

3.1.1. The R2ML Abstract Syntax: R2ML Metamodel

The R2ML metamodel is defined by using the MOF metamodeling language. In Fig. 2, we give a UML class diagram depicting the MOF definition of integrity rules with the definition of *RuleBase* and *RuleSet*. An R2ML *RuleBase* contains *IntegrityRuleSet*-s or *DerivationRuleSet*-s, while either of these two *RuleSet*-s contains specific rules, integrity or derivation, respectively. 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 lan-

guage such as first-order predicate logic or OCL [44]. 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*." In Fig. 2 we, define constraint on the *IntegrityRule* class, by means of OCL constraints to check well-formendess of the R2ML models, and we describe these well-formendess rules in more detail later in this subsection.

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

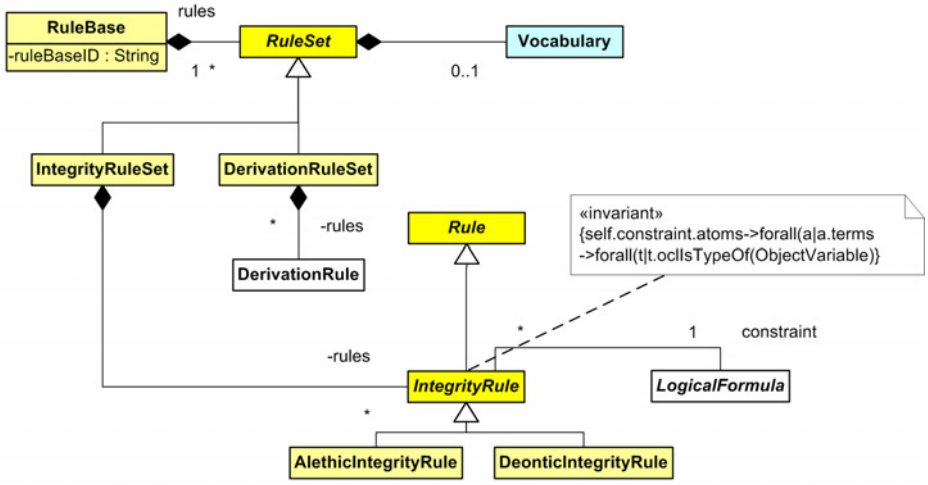

Fig. 2. The metamodel of integrity rules with *RuleBase* and *RuleSet*

R2ML defines the general concept of *LogicalFormula* 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. *LogicalFormula* can also be *AtLeastQuantifiedFormula*, *AtMostQuantifiedFormula*, and *AtLeastAndAtMostQuantifiedFormula* that allow defining cardinality constrains in R2ML rules.

A *derivation rule* has *conditions* and a *conclusion* (see Fig. 3) with the ordinary meaning that the conclusion can be derived whenever the conditions hold. Conditions of a derivation rule are instances of the *AndOrNafNegFormula* class representing quantifier-free logical formulas with conjunction, disjunction, and negation. Conclusions are restricted to quantifier-free disjunctive normal forms (*LiteralConjunction*) without NAF (Negation as Failure, i.e. weak negation).

Example 2 (Derivation rule). The discount for a customer buying a product is 7.5 percent if the customer is premium and the product is luxury.

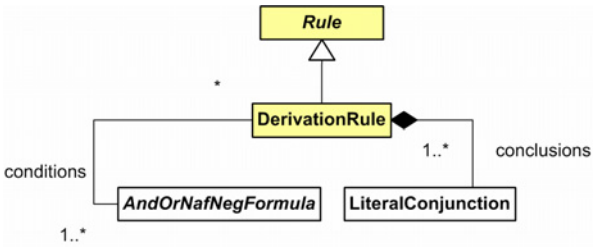

Fig. 3. An excerpt of the R2ML metamodel for derivation rules

Terms are the basic constituents of atoms in R2ML. Similar to atoms, the R2ML language distinguishes between object terms, data terms, and generic terms. An *ObjectTerm* is an *ObjectVariable*, an *ObjectName*, a *ReferencePropertyFunctionTerm*, or an *ObjectOperationTerm*. R2ML also has *DataTerms*, where a *DataTerm* is a *DataLiteral*, *DataVariable*, or *DataFunctionTerm*, while *DataFunctionTerm* can be *DataOperationTerm*, *AttributeFunctionTerm*, or *DatatypeFunctionTerm*.

R2ML supports *Variables* too. Variables are provided in the form of *ObjectVariable* (i.e. variables that stand for objects), *DataVariable* (i.e. variables that stand for data literals), and *GenericVariable* (i.e. variables that do not have a type).

3.1.2. 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:

1. Every metamodel class is represented by an XML element and a complexType in the XML schema. The names of the XML element and complex type are the same as the class name. If the class is abstract, the corresponding element is also abstract. The corresponding XML element contains XML attributes for each data type attributes from the model class. The R2ML metamodel contains only optional or required attributes which are mapped to *optional* and *required* attributes, respectively in the XML Schema. The metamodel does not contain any attribute whose type is a class.

2. A MOF association is mapped to an XML attribute that is part of the content model of the XML complexType generated from the class referencing this association. If a name of an association end is defined, then this name is used as an XML attribute name. If the association end name is not defined, then referenced class name is used as an XML attribute name.
3. Composite and n-ary MOF associations are always serialized by using XML e*lements. A com*posite associat*ion is mapp*ed to an XML element that is a part of the content model of the XML complexType generated from the class referencing this association. If an association end name is provided,

then this name is used for the corresponding XML element name. If an association end is undefined then the referenced class name is used.

In Fig. 4, we give the integrity (Fig. 4a) and derivation (Fig. 4b) rules defined in Example 1 and Example 2, respectively, in a form of XML documents following the R2ML XML schema.

```
<r2ml:RuleBase>
<!--Namespace definitions are omitted to reduce the size of this example-->
 <r2ml:IntegrityRuleSet>
  <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>
 </r2ml:IntegrityRuleSet>
</r2ml:RuleBase>
```

a)

```
<r2ml:RuleBase>
<!--Namespace definitions are omitted to reduce the size of this
example-->
 <r2ml:DerivationRuleSet>
  <r2ml:DerivationRule r2ml:id="DR004">
   <r2ml:conditions>
    <r2ml:ObjectClassificationAtom
     r2ml:classID="PremiumCustomer">
     <r2ml:ObjectVariable r2ml:name="customer"
                          r2ml:classID="Customer"/>
    </r2ml:ObjectClassificationAtom>
    <r2ml:ObjectClassificationAtom
     r2ml:classID="LuxuryProduct">
     <r2ml:ObjectVariable r2ml:name="product"
                          r2ml:classID="Product"/>
    </r2ml:ObjectClassificationAtom>
    <r2ml:AssociationAtom
     r2ml:associationPredicateID="buy">
     <r2ml:objectArguments>
      <r2ml:ObjectVariable r2ml:name="customer"/>
      <r2ml:ObjectVariable r2ml:name="product"/>
     </r2ml:objectArguments>
    </r2ml:AssociationAtom>
   </r2ml:conditions>
   <r2ml:conclusion>
    <r2ml:AttributionAtom r2ml:attributeID="discount">
     <r2ml:subject>
      <r2ml:ObjectVariable r2ml:name="customer"/>
     </r2ml:subject>
     <r2ml:value>
      <r2ml:TypedLiteral r2ml:datatype="xs:decimal"
       r2ml:lexicalValue="7.5"/>
     </r2ml:value>
    </r2ml:AttributionAtom>
   </r2ml:conclusion>
  </r2ml:DerivationRule>
 </r2ml:DerivationRuleSet>
</r2ml:RuleBase>
```

b)

Fig. 4. R2ML XML representation of the integrity rule from Example 1 (a) and the derivation rule from Example 2 (b)

3.2. OCL Metamodel and OCL EBNF-based Concrete Syntax

In this subsection, we describe the OCL language by explaining its MOF-based abstract syntax and EBNF-based textual concrete syntax. In this section, we describe only those OCL elements that are relevant to our discussion, while its complete description could be found in [44].

3.2.1. The OCL Abstract Syntax: OCL Metamodel

The OCL metamodel (i.e., abstract syntax for OCL v2.0) is also defined by using MOF [3]. In this abstract syntax, a number of meta-classes from the

UML 2.0 metamodel are imported [3]. The OCL metamodel is divided into several packages:

- The *Types* package describes the concepts that define the type system of OCL. It shows the types predefined in OCL as well as the types that are deduced from the UML models.
- The *Expressions* package describes the structure of OCL expressions.
- The *EnhancedOCL* package that we have added to the standard OCL metamodel to represent invariant constructs that are not supported in the standard OCL metamodel (more details are given later in this subsection).

An overview of the inheritance relationships between all classes defined in the package is shown in Fig. 5. The basic structure of the package consists of the OCL metamodel's classes such as *OclExpression* that is an abstract superclass for all OCL expressions; and *FeatureCallExp* that is superclass for the *OperationCallExp* and *PropertyCallExp* classes. *OperationCallExp* represents an operation defined on a UML *Classifier* (such as UML *Class*), while *PropertyCallExp* models a reference to an *Attribute* of a *Classifier* defined in a UML model.

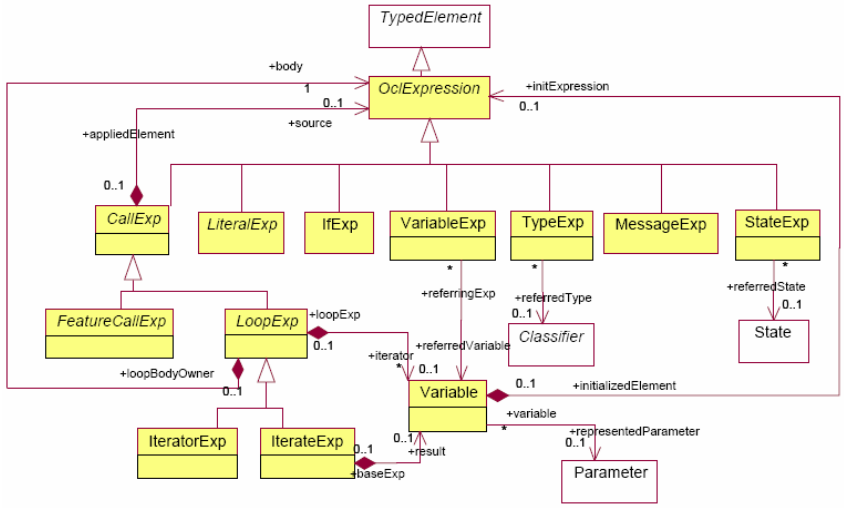

Fig. 5. The basic structure of expressions in the OCL metamodel

Since the standard specification of the OCL metamodel [3] does not contain support for OCL invariants (constraints that always must hold), we had to introduce the *EnhancedOCL* package. Thus, we just improved the model of the OCL language (i.e., OCL metamodel) to more precisely reflect the constructs that can be represented in the OCL language. Otherwise, we would not be able to provide mappings between OCL abstract and concrete syntax as explained in Section 4.3, given that we found out that OCL metamodel

could not support all OCL expressions[3]. This package contains the *Invariant* class, as a subclass of the *OclModuleElement* class (see Fig. 6). The white classes are from the UML metamodel, light-gray (or yellow) colored ones are from the standard OCL metamodel, and dark gray (green) are classes that we defined.

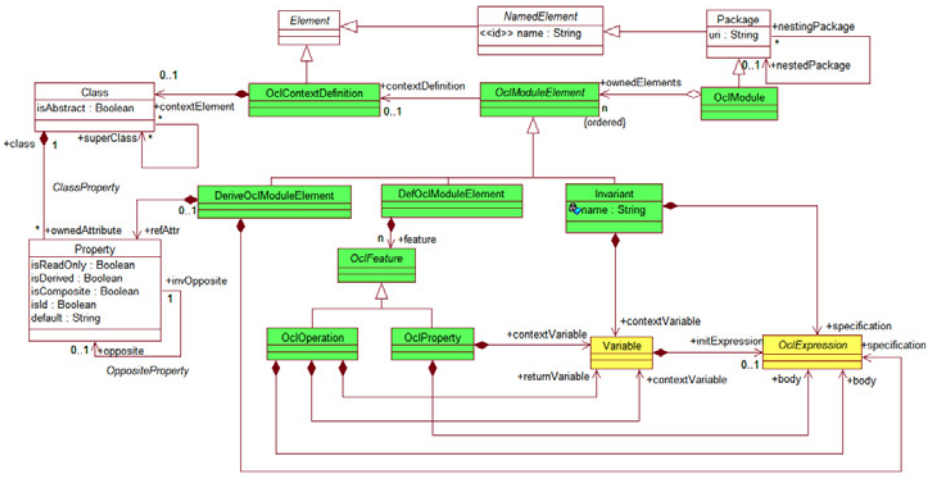

Fig. 6. Elements of the *EnhancedOCL* package in the OCL metamodel

Our *OclModuleElement* class represents a superclass for following elements:

- OCL invariant elements (represented with the *Invariant* class);
- OCL operations and properties, that is, “def” elements (represented with the abstract class *OclFeature*) that are represented with classes *OclOperation* and *OclProperty*, respectively; and
- OCL derivation rules, i.e., "derive" elements represented with class *DeriveOclModuleElement*.

3.2.2. The OCL Concrete Syntax

The concrete syntax of the OCL language is defined in a form of a full attribute grammar, where each production in an attribute grammar may have synthesized attributes attached to it [44]. The value of synthesized attributes of elements on the left hand side of a production rule is always derived from attributes of elements at the right hand side of that production rule. Each production may also have inherited attributes attached to it. In the attribute

[3] We are very grateful to Mariano Belaunde, who is involved in the OCL and QVT standardization process, for his generous help in defining the *EnhancedOCL package*.

grammar that specifies the concrete syntax, every production rule is denoted using the EBNF formalism and annotated with synthesized and inherited attributes, and disambiguating rules. Each production rule has one synthesized attribute called *ast* (short for abstract syntax tree), that holds the instance of the OCL Abstract Syntax that is returned by the rule. Each production rule also has one inherited attribute called *env* (short for environment), that holds a list of names that are visible from the expression. All names are references to elements in the model. In fact, *env* is a name space environment for the expression or expression part denoted according to the production rule [44].

The mapping from concrete to abstract syntax is described as part of the grammar in [44]. It is described by adding a synthesized attribute *ast* to each production which has the corresponding metaclass from the abstract syntax as its type. This allows the mapping to be fully formalized within the attribute grammar formalism. We show in Fig. 7 an example of production rules for *OclModuleElement*.

```
[A] OclModuleElementCS ::= DeriveOclModuleElementCS
[B] OclModuleElementCS ::= DefOclModuleElementCS
[C] OclModuleElementCS ::= InvariantCS
```

Fig. 7. Production rules for *OCLModuleElement*

The Abstract syntax mapping for *OclModuleElement* is defined in Fig. 8.

```
OclModuleElementCS.ast : OclModuleElement
```

Fig. 8. Abstract syntax mapping of *OclModuleElement*

Mapping from abstract to concrete syntax can be defined by applying the production rules from left to right, as shown in Fig. 7. In Section 3.1.1, we have shown some examples of invariants in the OCL textual concrete syntax. The full definition of the OCL concrete syntax with well-formedness rules can be found in [3].

3.3. SWRL Metamodel and SWRL Concrete Syntax

In this section, we present the SWRL language [3] by explaining its SWRL abstract syntax as well as its XML-based and RDF-based concrete syntax. Because mappings of SWRL OWL/XML Schema and its metamodel follow the same principles as for R2ML XML Schema and R2ML metamodel (in Section 3.2.2), we will here describe only parts of the SWRL relevant to our discussion, that is, SWRL's abstract syntax (metamodel) and it's two concrete syntaxes. SWRL that tends to be a standardized reasoning layer built on top of Web Ontology Language (OWL). OWL is a standard ontology language that Semantic Web applications use to exchange their ontologies.

3.3.1. The SWRL Abstract Syntax: RDM Metamodel

In the official W3C's submission of the SWRL language, there is no metamodel defined. In [3], authors, inspired by OMG's Ontology Definition Metamodel (ODM) [45] (a standardized MOF-based metamodel for OWL), proposed a Rule Definition Metamodel (RDM) – a MOF-based metamodel for SWRL [3]. As SWRL includes OWL constructs, RDM represents an extension of ODM [45]. ODM defines a metamodel for ontologies, by using the MOF metamodeling language.

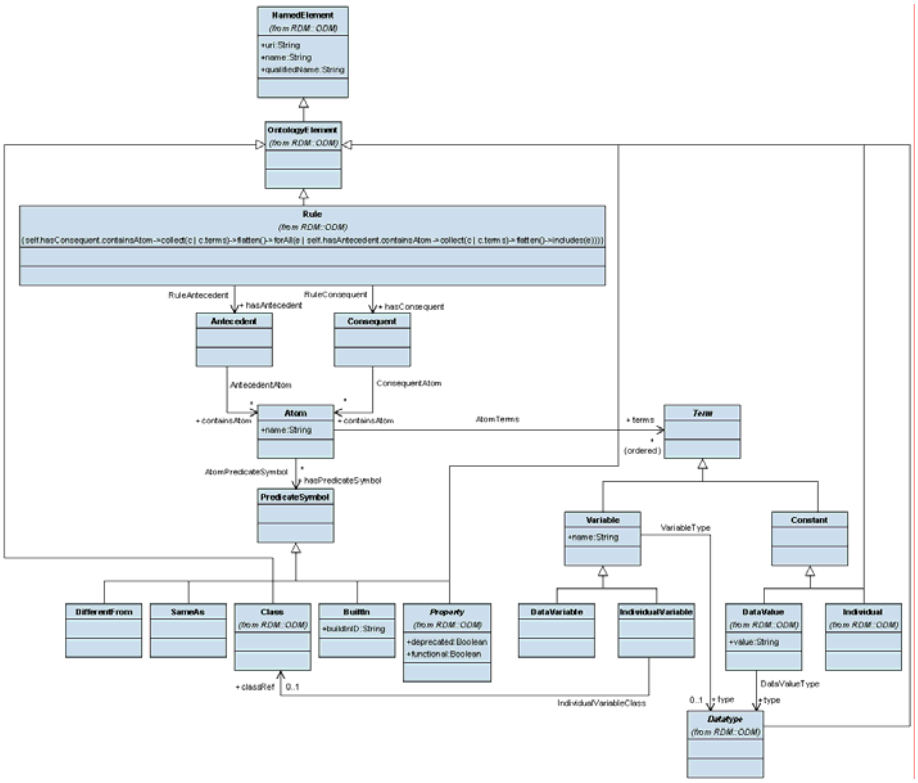

Fig. 9. Rule Definition Metamodel (adapted from [0])

SWRL defines rules as part of ontology. To reflect this, the RDM metamodel defines the *Rule* class as a subclass of ODM's *OntologyElement*. A SWRL rule (*Rule* class) consists of an antecedent (body) and a consequent (head). Both the antecedent and consequent consist of a set of atoms which can possibly be empty, as depicted by the multiplicity in Fig. 9[4]. Every SWRL rule is an implication, which means that *if* all atoms of the antecedent hold, *then* the consequent holds. The same antecedent or consequent can be used in several rules, as indicated in the meta-model by the multiplicity of the asso-

[4] Remark: some associations are not shown for the sake of better readability.

ciation between the *Rule* class, on the one hand, and the *Antecedent* class or the *Consequent* class on the other.

3.3.2. The SWRL Concrete Syntax

The SWRL language has two concrete syntax, namely, the RDF/XML concrete syntax [5] and OWL/XML concrete syntax [28]. We will give a short description of both in the following two subsections.

3.3.2.1. RDF/XML concrete syntax

As SWRL is defined on top of OWL, while OWL is based on RDF [35] (framework created to standardize defining and using metadata, i.e., resource descriptions on the Web), SWRL has an RDF-based concrete syntax. *RDF Schema* (*RDFS*) [13] is used to define vocabulary for RDF documents, and thus specify object types to which a certain property can be applied. This means that RDFS provides a basic typing mechanism for RDF models. The full definition of the RDF/XML schema of SWRL can be found in [29].

We mapped the SWRL RDF schema to the RDM metamodel by using the following mapping rules defined that we defined:

1. Every metamodel class is represented by an RDFS *Class*. The name of the RDFS *Class* is the same as the class name. The corresponding RDFS *Class* contains super-classes for each super-class from the model class.
2. A MOF association of RDM is mapped to an RDF *Property*. If a name of an association end is defined, then this name is used as a name of the RDF *Property*. If the association end name is not defined, then the referenced class name is used as an RDF *Property* name. Such an RDF *Property* has a *domain* attribute, which represents an RDF *Class* to which the property is attributed, and a *range* attribute which defines the scope of the property (the type speaking in terms of programming languages).

3.3.2.2. OWL/XML concrete syntax

The SWRL OWL/XML Concrete Syntax is a combination of the OWL Web Ontology Language XML Presentation Syntax[5] [28] with the RuleML XML syntax [0]. The mapping rules between the SWRL OWL/XML Schema and RDM metamodel follow the similar rules as we defined between R2ML XML Schema and R2ML metamodel defined in Section 3.1.2.

[5] XML Presentation Syntax is actually a concrete syntax defined by using XML Schema.

4. An implementation example: an ATL/QVT based approach

In this section, we describe implementation details of transformations between rule languages abstract and concrete syntax. We first show how we bridge between the XML Schema concrete syntax and the MOF-based abstract syntax of R2ML. Then, we describe specificities of bridging between SWRL's RDF concrete syntax (because for the SWRL OWL/XML Schema concrete syntax is the same as for R2ML). In the last part of this subsection, we describe how we bridge between the OCL EBNF-based concrete syntax and its abstract syntax (MOF-based metamodel). Using these bridges between the abstract and concrete syntax of rule languages, we cover most of the available rule languages relevant for the web rule community.

We would also mention that we do not describe the transformation between the SWRL OWL/XML Schema and RDM metamodel here, because this transformation follow the similar principles as the transformation between R2ML XML Schema and R2ML metamodel also described in this subsection.

4.1. Transformations between R2ML XML Schema and R2ML Metamodel

In this subsection, we explain the transformation steps undertaken to transform R2ML XML documents into the models compliant to the R2ML metamodel. We have already explained that the R2ML concrete syntax is an XML-based syntax and conforms to the R2ML XML Schema explained in Section 3.1.2. This syntax is located in the XML TS. However, the R2ML metamodel is defined in MOF, that is, it is in the MDE TS. To develop transformations between these two representations of R2ML, we should put them into the same technical space. We decided to implement these transformations in the MDE TS by using QVT and its implementations such as ATL. This approach enables easier maintaining of transformations, it has a better tool for managing MOF-based models, and also has one more important benefit, that is, MOF-based models can automatically be transformed into XMI [48]. This means, that we can produce the R2ML XMI documents by using general-purpose tools for transforming MOF-based models into XMI [48] implemented as a part of MOF-based repositories. We base our solution on the second alternative, i.e., in the MDE TS by using ATL. The overall organization of the transformation process is shown in Fig. 10. It is obvious that the 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 TS to the MDE TS); and 2. From the R2ML XML schema to the R2ML metamodel. In the rest of the section we explain both of these transformations.

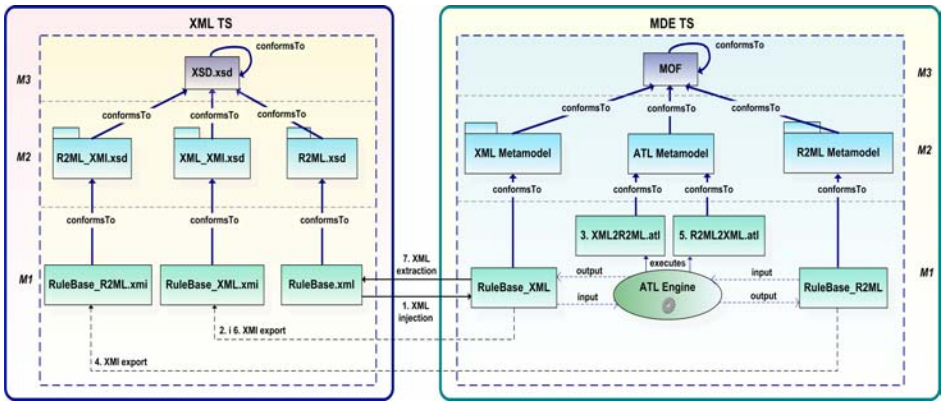

Fig. 10. The transformation scenario: R2ML XML format into the R2ML metamodel and vice versa

4.1.1. Transforming R2ML XML Schema into R2ML MOF-based Metamodel

The transformation process consists of two primary steps as follows.

Step 1. XML injection from the XML TS to the MDE TS. This means that we have to represent R2ML XML documents (RuleBase.xml from Fig. 10) into the form compliant to MOF. We use the XML injector that transforms R2ML XML documents (written w.r.t. the R2ML XML Schema, R2ML.xsd from Fig. 10) into the models conforming to the MOF-based XML metamodel (step 1 in Fig. 10) that defines XML elements such as XML *Node*, *Element*, and *Attribute*. 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. 10), created by the XML injector, is located on the M1 layer of the MDE TS. This means that the XML injector instantiates the MOF-based XML metamodel. 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. 10). This XMI format can be regarded as an implicitly defined XML schema (XML_XMI.xsd) compliant to the XML metamodel. Since ATL can use MDR as model handler, we can employ MDR's tool for exporting XMI documents (e.g., RuleBase_XML.xmi). For example, R2ML rules from Examples 1 and 2 (see Section 3.1.1) are represented in the XMI document in Fig. 11.

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. 10). 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 metamo-

metamodel. The XML2R2ML.atl transformation is performed on the M1 level (i.e., the model level) of the MDE TS. 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 check is automatically done by ATL's transformation engine during at run-time as this is the fundamental requirement of metamodeling). This principle is well-known as metamodel-driven model transformations [8]. In Table 1, we give an excerpt of mappings between the R2ML XML Schema, XML metamodel, and R2ML metamodel (where names of XML Schema and metamodel elements represent constraints on their instances). 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). The ATL transformation is done for classes, attributes, and references.

```
<XML.Root xmi.id = 'a1' name = 'r2ml:RuleBase' value = ''>
  <XML.Element.children>
    <XML.Element xmi.id = 'a6' name = 'r2ml:IntegrityRuleSet'
     value = ''>
     <XML.Element.children>
       <XML.Element xmi.id = 'a1'
        name = 'r2ml:AlethicIntegrityRule' value = ''>
         <XML.Element.children>
           <XML.Attribute xmi.id = 'a2' name = 'r2ml:id'
            value = 'IR001'/>
           <XML.Element xmi.id = 'a3'
            name = 'r2ml:constraint' value = ''>
            <XML.Element.children>
              <XML.Element xmi.id = 'a4'
               name = 'r2ml:UniversallyQuantifiedFormula'
               value = ''>
               <XML.Element.children>
                 <XML.Element xmi.id = 'a24'
                  name = 'r2ml:ObjectVariable' value = ''>
                  <XML.Element.children>
                    <XML.Attribute xmi.id = 'a12'
                     name = 'r2ml:name' value = 'r1'/>
                    <XML.Attribute xmi.id = 'a13'
                     name = 'r2ml:classID' value = 'Rental'/>
                  </XML.Element.children>
                 </XML.Element>
               <!--..-->
              </XML.Element>
       <!--..-->
     </XML.Element.children>
   </XML.Element>
  </XML.Element.children>
</XML.Root>
```

a)

```
<XML.Root xmi.id = 'a1' name = 'r2ml:RuleBase' value = ''>
  <XML.Element.children>
    <XML.Element xmi.id = 'a6' name = 'r2ml:IntegrityRuleSet'
     value = ''>
     <XML.Element.children>
       <XML.Element xmi.id = 'a15' name = 'r2ml:DerivationRule'
        value = ''>
        <XML.Element.children>
          <XML.Attribute xmi.id = 'a22' name = 'r2ml:id'
           value = 'DR001'/>
          <XML.Element xmi.id = 'a23' name = 'r2ml:conditions'
           value = ''>
           <XML.Element.children>
             <XML.Element xmi.id = 'a4'
              name = 'r2ml:ObjectClassificationAtom'
              value = ''>
              <XML.Element.children>
                <XML.Element xmi.id = 'a24'
                 name = 'r2ml:ObjectVariable' value = ''>
                 <XML.Element.children>
                   <XML.Attribute xmi.id = 'a32'
                    name = 'r2ml:classID'
                    value = 'PremiumCustomer'/>
                   <XML.Element xmi.id = 'a33'
                    name = 'r2ml:ObjectVariable' value = ''/>
                    <XML.Element.children>
                      <XML.Attribute xmi.id = 'a42'
                       name = 'r2ml:classID'
                       value = 'Customer'/>
                      <XML.Attribute xmi.id = 'a42'
                       name = 'r2ml:nane'
                       value = 'customer'/>
                    </XML.Element.children>
                 </XML.Element>
                </XML.Element.children>
              </XML.Element>
             <!--..-->
           </XML.Element>
       <!--..-->
     </XML.Element.children>
   </XML.Element>
  </XML.Element.children>
</XML.Root>
```

b)

Fig. 11. The integrity and derivation rules from Example 1 and Example 2 in Section 3.1.1. represented in the XML XMI format

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

R2ML schema	XML metamodel	R2ML metamodel	Description
RuleBase	Root name = 'r2ml:RuleBase'	RuleBase	Captures a collection of rules.
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'	basCont- Voc.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 (similar to functions in programming languages) in defining mappings. Each rule in the ATL has one input element (i.e., an instance of a meta-class 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. This actually means that we instantiate the R2ML metamodel (M2 level), i.e., create R2ML models (M1 level). In our ATL transformation, we mainly use so-called ATL matched rules. A matched rule matches a given type of source model element, and generates one or more kinds of target model elements. These rules are activated by the ATL rule engine for each element of a given type of the source model. Fig. 12 gives an example of a matched rule which is in fact an excerpt of the X2ML2R2ML.atl transformation for the *Root* class (XML!Root) of the XML metamodel. The ATL rule RuleBase transforms the Root of the XML model (RuleBase_XML) into the *RuleBase* element (i.e., R2ML!RuleBase in Fig. 12) of the R2ML metamodel. The *RuleBase* element (the Root of an XML tree) captures a collection of different *RuleSet*s (i.e., Derivation, Integrity, Reaction or Production). Each *RuleSet* includes rules of their own kind (e.g., *IntegrityRuleSet* includes only *IntegrityRule*-s). From the ATL rule shown in Fig. 12, we invoke rules for nested elements (e.g., *DerivationRuleSet* and *IntegrityRuleSet*) as well as for the ruleBaseID attribute.

```
module XML2R2ML;
create OUT : R2ML from IN : XML;

rule RuleBase {
  from
     i : XML!Root
  to o : R2ML!RuleBase (
      ruleBaseID <- i.getAttrVal('xmlns:r2ml'),
      rules <- XML!Element.allInstances()->select(e |
                                          e.name = 'r2ml:DerivationRuleSet'
                                          or e.name = 'r2ml:IntegrityRuleSet')
    )
```

Fig. 12. An excerpt of the transformation: A matched rule

However, this type of rules is not suitable for all input elements of the XML metamodel that we transformed into the R2ML metamodel. This is typical for a situation when several input elements represent the same entity, while in the target model we have to have only one unique definition of that element and other should only refer to that one. For example, the *ObjectVariable* XML element in the input rules might be used several times (as this is a tree) and the only way to know that all of them refer to the same variable is by the value of its name attribute (e.g., customer). However, in the target R2ML metamodel, an *ObjectVariable* should only be defined once (as it is a graph), while all other parts of the model can only refer to that definition. Fig. 4a shows an R2ML *IntegrityRule*, while Fig. 13 gives the same rule in a form of a UML object diagram representing instances of the R2ML MOF-based metamodel. In Fig. 4a, we can see that the *r1 ObjectVariable* element is a unique for the whole rule, and it appears at 3 different (marked) places in the rule. This means that an XML tree contains three different nodes referring to the same object, i.e., the *r1 ObjectVariable*. On the other hand, the instance of the R2ML MOF-based metamodel contains one and only one node referring to that object (r1:*ObjectVariable*), as it is shown in Fig. 13. Three links to that object represent three different appearances of that *ObjectVariable* in the whole rule.

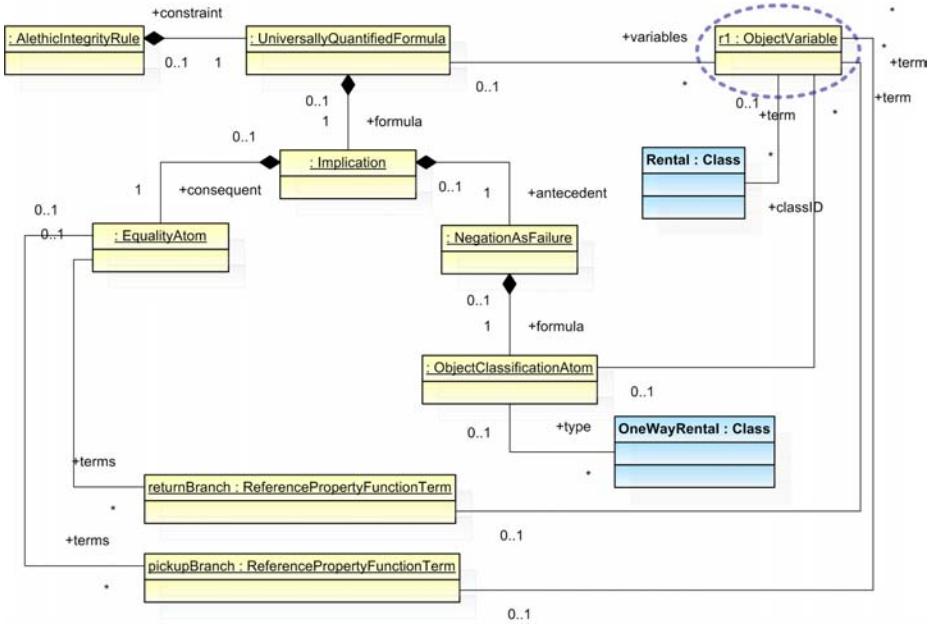

Fig. 13. A UML object diagram that represents the integrity rule shown in Fig. 4 in the R2ML XML format

It is obvious that matched rules are unsuitable for this type of transformation. As a solution to this issue, we employed ATL's "unique lazy" rules. They

are “lazy”, since we have to invoke them explicitly from some other rule. They are ”unique”, since for a specific element of the source model they only create an element of the target model once, while all other executions of the same rule for that specific input element create references to the only target definition. This type of rules resembles called templates in XSLT with the main distinction that XSLT does not possess any mechanism for unique element definitions, which is very hard to implement in XSLT [15]. The unique lazy ATL rule that transforms *ObjectVariables* from the XML metamodel into the R2ML metamodel is shown in Fig. 14. It transforms an Element of the XML metamodel (XML!Element) whose attribute name has the value “r2ml:ObjectVariable”. This element is transformed into the *ObjectVariable* element of the R2ML metamodel (R2ML!ObjectVariable) following all conditions given above.

```
unique lazy rule ObjectVariable {
  from i : XML!Element ( i.name = 'r2ml:ObjectVariable' )
  to ov : R2ML!ObjectVariable (
          type <- i.children->select(c | c.oclIsKindOf(XML!Attribute) and
                                         c.name = 'r2ml:classID')
                                             ->collect(e | thisModule.ClassRule(e))
                                                ->first(),
          name <- i.getAttrVal('r2ml:name')
    )
}
```

Fig. 14. An excerpt of the transformation from the R2ML XML format to the R2ML metamodel: A unique lazy rule

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. 10). For example, for rules shown in Fig. 11 (i.e., in the XMI format of the XML metamodel), we get an R2ML XMI document in Fig. 15. Fig. 15 shows a RuleBase that contains IntegrityRulesSet with one AlethicIntegrityRule. AlethicIntegrityRule then is constrained by a UniversallyQuantifiedFormula, which contains variable declarations. The current version of the transformation of the R2ML XML format into the R2ML metamodel covers derivation, integrity, reaction rules and production rules.

```
<R2ML.RuleBase xmi.id="a1">
  <R2ML.RuleBase.rules>
    <R2ML.IntegrityRuleSet xmi.id="a2">
      <R2ML.IntegrityRuleSet.rules>
        <R2ML.Rules.AlethicIntegrityRule xmi.id="a3">
          <R2ML.Rules.IntegrityRule.constraint>
            <R2ML.Formulas.UniversallyQuantifiedFormula
              xmi.id="a4">
              <R2ML.Formulas.QuantifiedFormula.variables>
                <R2ML.Terms.TerBasic.Variables.ObjectVariable
                  xmi.idref="a5"/>
              </R2ML.Formulas.QuantifiedFormula.variables>
              <R2ML.Formulas.QuantifiedFormula.formula>
                <R2ML.Formulas.Implication xmi.id="a11">
                  <R2ML.Formulas.Implication.consequent>
                    <R2ML.Atoms.EqualityAtom
                      xmi.id="a12" isNegated="false">
                      <R2ML.Atoms. EqualityAtom.terms>
                        <R2ML.ReferencePropertyFunctionTerm
                          xmi.idref="a7"/>
                        <R2ML. ReferencePropertyFunctionTerm
                          xmi.idref="a9"/>
                      </R2ML.Atoms.AtBasic.EqualityAtom.terms>
                    </R2ML.Atoms.AtBasic.EqualityAtom>
                  </R2ML.Formulas.Implication.consequent>
                  <!--...-->
                </R2ML.Formulas.Implication>
    <!--...-->
    </R2ML.IntegrityRuleSet>
  </R2ML.RuleBase.rules>
</R2ML.RuleBase>
```

a)

```
<R2ML.RuleBase xmi.id="a1">
  <R2ML.RuleBase.rules>
    <R2ML.DerivationRuleSet xmi.id = 'a2'>
      <R2ML.DerivationRuleSet.rules>
        <R2ML.Rules.DerivationRule xmi.id = 'a3'>
          <R2ML.Rules.DerivationRule.conclusions>
            <R2ML.Formulas.qf.LiteralConjunction xmi.id = 'a4'>
              <R2ML.Formulas.qf.LiteralConjunction.atoms>
                <R2ML.Atoms.AttributionAtom xmi.id = 'a5'
                  isNegated = 'false'>
                  <R2ML.Atoms.AttributionAtom.dataValue>
                    <R2ML.Vocabulary.TypedLiteral xmi.idref = 'a6'/>
                  </R2ML.Atoms.AttributionAtom.dataValue>
                  <R2ML.Atoms.AttributionAtom.attribute>
                    <R2ML.Vocabulary.Attribute xmi.idref = 'a7'/>
                  </R2ML.Atoms.AttributionAtom.attribute>
                  <R2ML.Atoms.AttributionAtom.subject>
                    <R2ML.Terms.Variables.ObjectVariable
                      xmi.idref = 'a8'/>
                  </R2ML.Atoms.AttributionAtom.subject>
                </R2ML.Atoms.AttributionAtom>
              </R2ML.Formulas.qf.LiteralConjunction.atoms>
            </R2ML.Formulas.qf.LiteralConjunction>
          </R2ML.Rules.DerivationRule.conclusions>
          <!--...-->
        </R2ML.Rules.DerivationRule>
      </R2ML.DerivationRuleSet.rules>
    </R2ML.DerivationRuleSet>
  </R2ML.RuleBase.rules>
</R2ML.RuleBase>
```

b)

Fig. 15. The R2ML XMI representation of the R2ML integrity and derivation rules shown in Fig. 4 (R2ML XML) and Fig. 11 (XML XMI)

4.1.2. Transforming R2ML MOF-based Metamodel and 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. 10) into an XML model (RuleBase_XML) by using an ATL transformation named R2ML2XML.atl (step 5 in Fig. 10). After applying this transformation to the input R2ML models, XML models (RuleBase_XML) are stored in the model repository (RuleBase_XML.xmi in Fig. 10). The output XML model conforms to the XML metamodel. The R2ML2XML.atl transformation transforms elements of the source R2ML metamodel to elements of the target XML metamodel. This transformation is also executed on the M1 level, as well as the XML2R2ML.atl transformation. Mappings from Table 1 between the R2ML XML Schema, XML metamodel, and R2ML metamodel apply here with no changes. So, for the R2ML rules given the R2ML XMI format in Fig. 15, we get an XML model which can be serialized back into the XML XMI format (step 6 in Fig. 10), as it is shown in Fig. 11.

Step 2. The XML extraction from the MDE TS to the XML TS. In this step, we transform the XML model (RuleBase_XML in Fig. 10) which conforms to

MOF-based XML metamodel and is generated in step 1 above, to the Rule-Base.xml document (Step 7 in Fig. 10). 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 the checking the negation of Atoms. All the ATL matched transformation rules are defined straightforward similar as in the XML2R2ML transformation, except for unique elements (like ObjectVariable). For every R2ML metamodel element, we create a necessary number of XML meta-model elements (*Attribute*, *Element*), which corresponds to the R2ML XML Schema. Since XML metamodel Element's children association end is defined as a composition [39], we can just not use one *ObjectVariable* and then reference to it from different *Element*s. To implement the support for transforming these elements, we used ATL lazy rules, which when called create a new element with the same content from the same input element. For example, for a unique lazy rule, which created a unique *ObjectVariable* element (from Fig. 14), we define a corresponding lazy rule like the one shown in Fig. 16.

```
lazy rule ObjectVariable {
  from i : R2ML!ObjectVariable
  to o : XML!Element (
            name <- 'r2ml:ObjectVariable',
            children <- Sequence { attrName,
                                   if not i.classRef.oclIsUndefined() then
                                       thisModule.ClassRule(i.classRef)
                                   else  OclUndefined
                                   endif
                     }
     ),
     attrName : XML!Attribute (
        name <- 'r2ml:name',
        value <- i.name
    )
}
```

Fig. 16. An excerpt of the ATL transformation from the R2ML metamodel to the R2ML XML: A lazy rule

This rule creates an output XML Element with the name "r2ml:ObjectVariable" for every R2ML *ObjectVariable* element on which it is applied. In Fig. 16, we also give an example of invocation of another lazy rule (i.e., *ClassRule*) for the R2ML *Class* elements that are also unique.

4.2. Transformations between SWRL RDF/XML Schema and RDM Metamodel

Because the transformation between the SWRL OWL/XML Schema and the RDM metamodel follows very similar principles as the transformation between the R2ML XML Schema and the R2ML metamodel, described in Section

4.1.1, we will describe here in detail only differences which are important for the bi-directional transformation between the SWRL RDF/XML Schema and the RDM metamodel.

Regarding both transformations, to define them we need to define two sets of transformations. The first set of transformations for bridging between the SWRL OWL/XML Schema and the RDM metamodel, and the second set for bridging between the SWRL RDF/XML Schema and the RDM metamodel. For bridging between the SWRL OWL/XML Schema and the RDM metamodel, we need to define two transformations, namely, a transformation from the SWRL OWL/XML Schema to RDM metamodel (which includes the XML injection and XML2RDM.atl transformation form Fig. 17) and another transformation from the RDM metamodel to the SWRL OWL/XML Schema (which includes RDM2XML.atl transformation and XML extraction in Fig. 17). For bridging between the SWRL RDF/XML Schema and the RDM metamodel, we also need to define two transformations, namely, a transformation from the SWRL RDF/XML Schema and the RDM metamodel (which includes the XML injection and the RDF2RDM.atl transformation from Fig. 17) and a transformation from the RDM metamodel to the SWRL RDF/XML Schema (which includes the RDM2RDF.atl transformation and the XML extraction from Fig. 17) The overall organization of the transformation process is shown in Fig. 17.

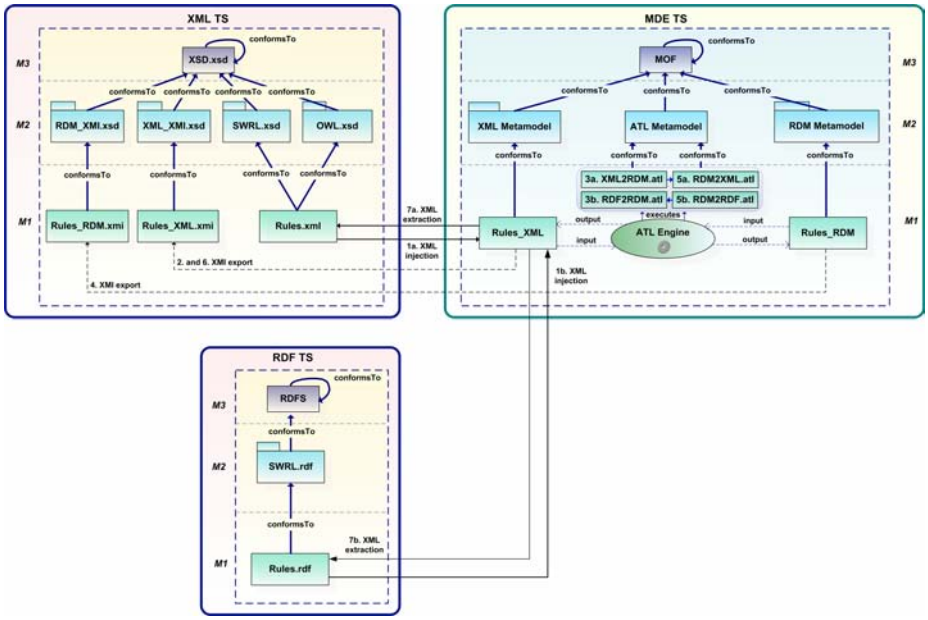

Fig. 17. The transformation scenario: SWRL XML and RDF format into the RDM metamodel and vice versa

To bridge between the SWRL RDF concrete syntax and the SWRL abstract syntax (i.e., RDM), we first use the XML injector, (see Fig. 17, 1b. for RDF TS: XML injection), a part of ATL that automatically transforms SWRL RDF docu-

ments (for RDF syntax) into the models conforming to the MOF-based XML metamodel that defines XML. Once we inject SWRL RDF rules into a MOF-based representation (*Rules.xml* in Fig. 17), we can manipulate with them like with any other type of MOF-based models. Thus, such XML models can be represented in the XML XMI format (in Fig. 17, step 2: XMI export). Now we transform between XML models (Rules_XML from Fig. 17) and RDM-compliant models (Rules_RDM from Fig. 17). This actually requires writing two ATL transformations (Fig. 17, step 3b: RDF2RDM.alt and step 5b: RDM2RDF.atl), and hence this is the *bridge* between both the SWRL XML-based and RDF-based concrete syntax and the SWRL abstract syntax. Both transformations are executed on the M1 level, but they require the input and output models to be compliant to the input and output metamodels (i.e., XML and RDM), respectively. This way we check validity of all input SWRL XML-based and RDF-based rules w.r.t. the RDM metamodel. Since we have implemented transformations in both directions, we can transform RDM rules into the XML models, that can later be exported into SWRL RDF concrete syntax (Fig. 17, step 6: XML export) to obtain the rule in the RDF/XML syntax-form. Note also that once we transform SWRL rules into the RDM representation, we can also export SWRL rules into the RDM XMI format (Fig. 17, step 7b: XMI export in RDF TS), and thus we can share SWRL rules with any MOF-compliant repository. This is another important contribution to the RDM metamodel itself [14] that improves its practical value to be used by other MOF-based tools.

As we have mentioned in the beginning of this section, the transformation between the SWRL OWL/XML Schema and the RDM metamodel follow the same principles as transformation between the R2ML XML Schema and the R2ML metamodel. We should note differences for this transformation in relation to the transformation between the SWRL RDF/XML Schema and the RDM metamodel. The first step (1a. XML injection from Fig. 17) and the last (7a. XML extraction from Fig. 17) are different, because we inject and extract SWRL OWL/XML files to the XML TS and not to RDF TS. For the transformation between the XML metamodel and the RDM metamodel, we defined two transformations (3a. XML2RDM.atl and 5a. RDM2XML.atl from Fig. 17) that transforms XML instances of SWRL OWL/XML rules to and from RDM metamodel, respectively.

4.3. Transformations between OCL concrete syntax and OCL Metamodel

This transformation includes bridging between the OCL (EBNF-based) concrete syntax and the OCL abstract syntax (i.e., OCL metamodel). Because the OCL textual concrete syntax is located in the EBNF TS, we need to create an instance of the OCL metamodel (abstract syntax) in the MDE TS. To do this, we first use the EBNF injector, (see Fig. 18, step 1: EBNF injection), a part of the ATL toolkit, and the OCL Lexer and Parser. We generated the OCL Parser and Lexer by using the ATL TCS (Textual Concrete Syntax) tools.

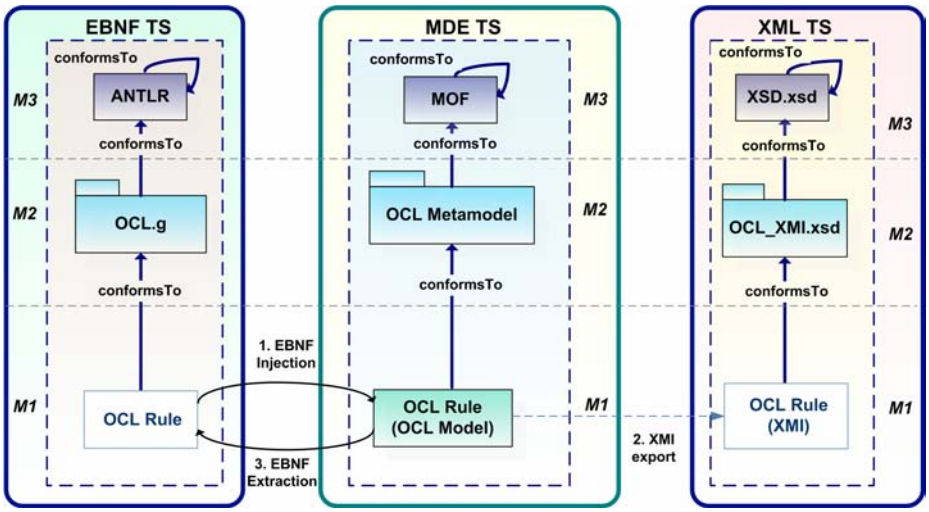

Fig. 18. The transformation scenario between OCL concrete and abstract syntax (metamodel)

The OCL Parser automatically transforms OCL invariants into the models conforming to the MOF-based OCL metamodel. Once we created the OCL TCS and generated OCL Parser and Lexer based on it, the EBNF injector takes for input the OCL metamodel, OCL code that we want to parse (as the *.ocl* textual file), generated OCL Lexer and Parser, and it returns a MOF-based OCL model as output. Once we inject OCL invariants into a MOF-based representation (OCL Rule in Fig. 18), we can manipulate with them like with any other MOF-based model (and export it into OCL XMI, step 2 in Fig. 18). Fig. 19 shows an excerpt of the mappings between the OCL meta-model (in the KM3 format[6], Fig. 19a) and its textual concrete syntax defined in TCS (Fig. 19b).

Fig. 19 shows the corresponding part of the OCL metamodel, because TCS works in a way that, for each element of a metamodel, it defines a description in the textual concrete syntax, by using the constructs of the TCS language (e.g., *template*). The figure shows the following elements:

- Class *OclModule* is represented with *ownedElement*s (of the type *OclModuleElement*) and is a construct which is first created (denoted with *main*).
- Class *Invariant* is represented with: the context definition (i.e., the name of the appropriate UML or MOF class), the keyword "inv", the name of this constraint if it is defined, the symbol ":", and the specification (body) of this constraint. The elements <newline> and <tab> are used in serialization of

[6] KM3 is a domain specific language (DSL) for defining metamodels as well as MOF or Ecore. Its syntax is very similar to the one of Java [3].

the OCL model to the OCL code, and they represent a new line feed, and text bias, respectively.

- Class *OclContextDefinition* is represented with the keyword "context" and *contextElement* element.

```
class OclModule extends Package {
        reference ownedElements[*] ordered container : OclModuleElement;
    }

class Invariant extends OclModuleElement {
        attribute name : String;
        reference specification container : OclExpression;
    }

class OclContextDefinition extends Element {
        reference contextElement container : Class;
    }
```

a) OCL meta-model

```
template OclModule main
        : ownedElements
        ;

template Invariant context
        :  (isDefined(contextDefinition) ? contextDefinition)
           <newline> <tab> "inv" (isDefined(name) ? name)
           <no_space> ":" [ specification ] { endNL = false}
        ;

template OclContextDefinition
        : "context" contextElement
        ;
```

b) OCL textual concrete syntax

Fig. 19. An excerpt of transforming the OCL meta-model elements into the OCL textual concrete syntax (TCS)

TCS elements are associated to their corresponding elements in the metamodel by their names. For example, TCS template *OclModule* corresponds to the KM3 class *OclModule*, while *ownedElement* corresponds to the KM3 *OclModuleElement* class (and all its subclasses).

5. Experiences

The transformation is tested on a set of real world rules collected by the REWERSE Working Group I1 at Brandenburg University of Technology at Cottbus, from different sources such as Warmer and Kleppe' book [60] and the SWRL specification [29]. In this section, we report on some lessons learned when developing and applying the transformations. These lessons also helped us validate the R2ML, RDM and OCL MOF-based metamodels as well as to propose some changes to the R2ML, RDM and OCL metamodels. It is important to notice that while we here report on some of the experiences with the use of R2ML, the same can be applied to SWRL, as that was the language for which there was not any solution available to bridging its abstract and concrete syntax. We have already indicated some our experience with the OCL metamodel, and what extensions to that model we had to do in Section 3.2.

Making Unique Mappings for Bridging Abstract and Concrete Syntax. In the case we have defined abstract and concrete syntax in different technical spaces, we need to define conceptual mappings between corresponding elements of that syntax (such as in the case of R2ML as shown in Section 3.1). When such mappings are defined, we need to define rules for checking consistency between abstract and concrete syntax (these rules represent well-formedness rules shown, e.g., for R2ML in Section 3.1.1). Based on the defined mappings, we can later define two unidirectional transformations (for both directions) or one bidirectional transformation to bridge concrete models into one or another technical space (such as for R2ML in Section 4.1.1.), con-

forming to the rule language' metamodel, or concrete syntax definition, respectively

Transformations based on Mappings for Bridging Abstract and Concrete Syntax. Once a technical space for the representation of a rule language's abstract syntax is chosen (e.g., MOF or EBNF), and when such an abstract syntax is defined, we then need to choose a suitable solution to developing the transformation. This solution should be based on criteria such as potential to define needed transformations, flexibility to change those transformations, and compatibility to work in needed technical spaces. Once the abstract or concrete syntax of the rule language is changed, we need to change mappings between their elements and implemented transformations, too.

XML/RDF Schema and Metamodel Mappings. XML Schema elements and metamodel elements are mapped in the following way: every metamodel class is mapped to an XML element and a complexType in the XML schema (or to RDF class if we map RDF-based syntax). The names of the XML element and complex type are the same as the class's name. If the class is abstract, the corresponding element is also abstract (this holds only for XML, not for RDF which does not have a mechanism for defining abstract classes). The corresponding XML element contains XML attributes for each data type attributes from the model class. A MOF association is mapped to an XML attribute (or to an RDF *Property* in the case of RDF-based syntax). Such an XML attribute is a part of the content model of the XML complexType mapped from the class referencing the association that is being mapped. Composite and n-ary MOF associations are always mapped using XML elements. A composite association is mapped to an XML element that is a part of the content model of the XML complexType generated from the class referencing this association. In the case of RDF, a MOF composite association is mapped to an RDF Property whose domain is a translated class referencing the association and whose range is the translated class referenced by the association.

Transformations of Rule Language Variables. As model transformation languages transform every element of the input model into the corresponding element of the output model, this can be problematic if we want to have only one output element for multiple input elements (i.e., one-to-many transformation) such as the case for rule language variables as described in section 4.1. In order to define these types of transformation rules, we need to search through the input XML model graph to find all occurrences of a variable (by using helper functions described in section 5.2.1.2) and to create a unique element with the same name in the output model. In the opposite direction the situation is a bit simpler. In the case we need to create multiple elements in the output model from one unique element of the input model, we propose using transformation rules/functions, which when called create a new element with the same content from the same input element (see Fig. 16).

Abstract classes. Originally, some classes of the R2ML metamodel were defined as abstract classes (e.g., Disjunction, Conjunction, and Implication) [58]. When we attempted to transform rules form the R2ML XML format into the R2ML metamodel, we faced the problem that the ATL engine refused executing the ATL transformations. The problem was that some classes

should actually have not been 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. We should note that we did not face this problem with the RDM and OCL metamodels.

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 only be 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 cannot be composed by two other instances, regardless of the fact that one of them is an instance of the class A and another one of the class C [10]. Since ATL uses the MDR model repository for storing input and output models of transformations, 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 in the R2ML and RDM metamodels 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).

Transformations summary. During the implementation of the transformations, we created a certain number of different ATL rules and helpers for transformations. Table 2, shows the number of different ATL rules for the transformations that bridge between the R2ML XML Schema and R2ML metamodel.

Table 2. Number of different ATL rules in XML2R2ML and R2ML2XML transformations

Transformation/rules	Matched rules	Lazy rules	Unique lazy rules	Helpers
R2ML XML schema to R2ML metamodel	43	0	9	28
R2ML metamodel to R2ML XML schema	43	12	0	1

The number of matched rules is the same for both transformations. This is obvious because we transform R2ML elements straightforwardly to their R2ML XML Schema representation and vice versa. As described in section 4.1.1, for every element that must be defined as unique in the MDE TS (i.e., the R2ML metamodel), we used unique lazy rules for its creation. In the opposite direction, we have only used lazy rules, since we wanted to create more than one output elements with same contents from one input element. The number of helpers is much higher in the XML2R2ML transformation, and this is due to the need to walk through the input XML model and find occurrences of the same *ObjectVariable*, as described in sections 5.2.1.2 and 5.2.1.4.

For transformations between the SWRL OWL/XML Schema and RDM metamodel, a summary of the used types of transformation rules is shown in Table 3.

Table 3. Number of different ATL rules in the XML2RDM and RDM2XML transformations

Transformation/rules	Matched rules	Lazy rules	Unique lazy rules	Helpers
SWRL OWL/XML Schema to RDM meta-model	35	0	3	23
RDM meta-model to SWRL OWL/XML Schema	35	3	0	1

As with the transformation of the R2ML metamodel into the R2ML XML Schema, the number of the matched rules is the same for both transformations. As already described for R2ML, for every element which must be defined as unique in the MDE TS (i.e., RDM metamodel), we used unique lazy rules for their creation. In the opposite direction, we used only lazy rules, because we had to create more than one output element with the same contents from the same input element (see Section 5.2.1.4). The number of helper operations is larger in the XML2RDM transformation, because we need to search through the input XML model to find instances of the same *IndividualVariable* elements.

Regarding transformations between the SWRL RDF/XML Schema and RDM metamodel, a summary of the used types of transformation rules is shown in Table 4.

Table 4. Number of different ATL rules in the RDF2RDM and RDM2RDF transformations

Transformation/rules	Matched rules	Lazy rules	Unique lazy rules	Helpers
SWRL RDF/XML Schema to RDM meta-model	35	0	3	23
RDM meta-model to SWRL RDF/XML Schema	35	3	0	1

As with the transformation between SWRL OWL/XML Schema and RDM metamodel, the number of transformation rules and helpers is the same, because the SWRL OWL/XML Schema and RDF Schema have the same number of elements.

For the OCL metamodel and OCL concrete syntax bidirectional transformation, we defined a number of transformation rules (called templates in TCS), and a summary is shown in Table 5.

In this case from Table 5, we can see that we have 51 templates, that is, one simple TCS template for every non-abstract metamodel class whose corresponding concrete syntax element does not have operator associated with it. We have 6 primitiveTemplates for every primitive type (such as integer, boolean), one enumerationTemplate which corresponds to the OCL metamodels

CollectionKind enumeration, and we also have six operatorTemplates used to describe those OCL abstract syntax element that have operators associated with them (as described in Section 5.2.2.3).

Table 5. Number of different ATL/TCS rules in the OCL metamodel to and from OCL concrete syntax

Transformation/rules	primitiveTemplate	template	operatorTemplate	enumerationTemplate
OCL metamodel to/from OCL concrete syntax	6	51	6	1

6. Related work

XML Interchange Metadata (XMI) is one of the most related works to our approach. It is the OMG's specification providing a set of rules for mapping between MOF-based models, metamodels, and metametamodels and XML. Although XMI allows for sharing MOF-based artifacts between various applications, it is rather a verbose solution that produces many XML elements and attributes and without ways for developers to define and use their own XML-based concrete syntax suitable for specific domains such as Web rule languages. In this paper, we have shown how model transformation languages can be used to bridge this gap between concrete and abstract syntax of R2ML, SWRL and OCL, and thus a way for approaching various Web rule definitions. A special advantage of this approach is that once we have mappings between an arbitrary R2ML, SWRL and OCL concrete syntax and their corresponding abstract syntax, we also have mappings between that arbitrary R2ML, SWRL and OCL concrete syntax and the R2ML, SWRL and OCL XMI concrete syntax, respectively.

The proposed solution will even be more suitable if we only need to provide one bi-directional transformation between any pair of concrete and abstract syntax. This problem has already been addressed by several researchers, where they have proposed different solutions for bridging MOF-based abstract syntax (e.g., metamodels) and textual [30] [41] and graphical [19] concrete syntax of languages. Given that all current solution are looking at this problem at a more general level, i.e., mappings between MOF-based metamodels and EBNF grammars, we think that it would be useful to have their specialization dealing only with mappings between XML (as a special type of EBNF grammars) and MOF. This is actually very similar to transformations between XML schemas and Semantic Web ontologies (i.e., so-called lifting and lowering) [2]. One another interesting initiative in this area is related to model weaving [17]. The use of matching transformations and model weaving enabled to semi-automate the production of model transformations. Matching transformations are a special kind of transformations that implement heuristics and algorithms (such as element similarity and best links [17]) to create weaving models, while weaving models are models that capture different kinds of relation-

ships between models in a weaving metamodel[7]. The weaving model is translated into a model transformation language (such as ATL). This approach could enable, to some extent (we can say that variables cannot be supported in this way), bridging XML/RDF based concrete syntax with their abstract syntax (i.e., metamodels) by defining weaving models between the XML metamodel and the rule language's corresponding metamodel elements.

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) [25] [4] [20]. 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 TS. Examples of transformations for the RuleML language, which are done by using the XSLT approach, are the following: XSLT translator [56] between the Horn-logic subsets of RuleML and Relational-Functional Markup Language (RFML) [51], XSLT translator from RuleML to Jess [55], translators between Positional RuleML to Object-Oriented RuleML [49], and translator from RFML to RuleML. Note also that there are some translators for R2ML developed by using XSLT [50] such as translators from R2ML to F-Logic, between the F-Logic XML format and R2ML, from R2ML to Jess, R2ML to RuleML, R2ML to the rules of Jena2 – a most commonly used Semantic Web framework [37], and from R2ML to JBoss rules.

Although the use of XML is very suitable, the previous analysis of the use of XSLT for sharing knowledge indicates that XSLT is hard to maintain where modifications of input and output formats can completely invalidate previous versions of XSLTs [31]. Even some recent experiences in transforming rule languages (SWRLp) report on constraints of XSLT (e.g., to transform unique symbols) that can only be overcome by XSLT extensions implemented in other languages such as Java and Jess [36]. An important drawback of the XSLT approach is that XSLT does not have any language to check validity of the XML documents regarding XML Schema to which it conforms to during the execution of the transformation and it does not have a constraint language such as OCL in MDE TS. In our case, we used the ATL language whose main benefits are a good support different technical spaces through XML and EBNF injection and extraction and advanced features for creating and using a richer set of transformation rules (matched, called, lazy, and unique).

The OMG's Ontology Definition Metamodel (ODM) specification is closely related to our work [45] [21], as it specifies MOF-based metamodels for Semantic Web ontology languages Resource Description Framework Schema and OWL, i.e., it defines the abstract syntax of RDFS and ODM by using MOF. The ODM speciation also defines QVT-based mappings between the ODM and RDFS metamodels with metamodels of languages such as UML, Common Logics, Topic Maps, and Entity-Relationship model. However, all these transformations are defined on the level of metamodels, thus everything

[7] The weaving metamodel is a metamodel capable of representing correspondences and links between model elements [0].

happens in the MDE TS [22]. This means that, for example, there is a gap between the RDF/XML syntax [5] usually supported by OWL tools (as a concrete syntax of the OWL language) and the OWL metamodel (i.e., an abstract syntax of OWL). Our approach shows how this can be overcome, so that one can achieve the full compatibility between abstract and concrete syntax of a Web language for knowledge representation. We hope that our experience will be used in developing the future Web rule interchange standard.

7. Conclusion

In this paper, we have demonstrated potentials of model transformations for transforming Web rule languages. First, the use of model transformation languages forces us to use valid source and target models. This means that the transformation cannot be executed properly if either of rule models is not fully conformant to its metamodel. In our case, the source XML rule models have to be conformant to the XML metamodel, while R2ML, RDM and OCL models have to be conformant to the R2ML, RDM and OCL metamodels, respectively. 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 rule language metamodels, i.e. their abstract syntax. This helped us detect some issues in the rule language metamodels such as conflicting compositions and inappropriate abstract classes in the R2ML metamodel. Third, instances of rule metamodels are stored into MOF-based repositories such as MDR. Since model repositories have generic tools for exporting/importing models and metamodels in the XMI format, we employ them to export instances of the R2ML, RDM and OCL metamodels in the XMI format, and thus share R2ML, RDM and OCL models with other MOF-compliant applications. Finally, the use of ATL is more appropriate than XSLT when transforming rules between the XML and MDE technical spaces, since ATL supports advanced features (e.g., lazy and unique lazy rules) for transforming between languages based on metamodels (i.e., graphs) and XML-based concrete syntax (i.e., trees).

Looking from the perspective of general applicability of the proposed approach to other Web rule languages, let us refer to F-Logic and Web Service Modeling Language (WSML). For example, F-Logic's mainly used concrete syntax is textual, while the EU-funded NeON project defined a metamodel for F-Logic [26]. This basically means that our ATL/TCS approach to bridging between an EBNF-defined concrete and MOF-based abstract syntax of a Web rule language can directly be applied to F-Logic. Of course, the case of F-Logic is even more interesting, as F-Logic has an XML-based concrete syntax, which means that our approach to mapping between XML-based concrete syntax and MOF-based abstract syntax of rule languages can directly be applied to F-Logic as well. Furthermore, once F-Logic is placed in the MDE TS (via its MOF-based metamodel), we can then provide transformations

between F-Logic and other rule languages (e.g., R2ML), without a need to pay attention to different concrete syntax of F-Logic. Similarly, WSML has EBNF- and XML-based textual concrete syntax, which indicates that for most of current and forthcoming Web rule languages our solution can be used. This is even more significant, if we are aiming at further integration of Web rule languages into the software development process, where MDE can play a vital role.

In the future work, we will use real-world rules we have transformed into the R2ML metamodel for implementing 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 (using its RDM metamodel). In this way, we will be able to validate the potentials of the R2ML metamodel to integrate various rule languages as well as to (re)use rules from different origins in MOF-based applications. Of course, in this research, we will 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 RDM) [38]. In this way, we will be able to validate the potentials of the R2ML metamodel to integrate various rule languages as well as to (re)use rules from different origins in MOF-based applications.

8. Acknowledgements

The research of Athabasca University has in part been support by Canada's NSERC-funded LORNET Research network (http://www.lornet.org/). The research of the Brandenburg University of Technology at Cottbus has partially been funded by the European Commission and by the Swiss State Secretariat for Education and Research within the 6th Framework Programme project REWERSE number 506779 (cf. http://rewerse.net).

9. References

1. Akehurst, D., H., Kent, S., "A relational approach to defining transformations in a metamodel", *In Proceedings of the 5th International Conference on The Unified Modeling Language, Dresden, Germany*, pp. 243-258, 2002.
2. An, Y., Borgida, A., Mylopoulos, J., "Constructing complex semantic mappings between XML data and ontologies", *In Proceedings of the 4th International Semantic Web Conference, LNCS 3729*, Galway, Ireland, pp. 6-20, 2005.
3. ATLAS Transformation Language (ATL), http://www.sciences.univ-nantes.fr/lina/atl.
4. Ball, M., Boley, B., Hirtle, D., Mei, J., Spencer, B, "Implementing RuleML Using Schemas, Translators, and Bidirectional Interpreters", *In Proc. of the W3C Workshop on Rule Languages for Interoperability*, Washington, D.C., 2005.

5. Beckett, D. (Ed.), RDF/XML Syntax Specification (Revised), W3C Recommendation, http://www.w3.org/TR/rdf-syntax-grammar/, 2004.
6. Bézivin, J., Kurtev, I., "Model-based Technology Integration with the Technical Space Concept", *In Proceedings of the Metainformatics Symposium*, 2006.
7. Bézivin, J., Büttner, F., Gogolla, M., Jouault, F. Kurtev, I., and Lindow, A., "Model Transformations? Transformation Models!", *In Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems, LNCS 4199*, Genoa, Italy, 2006.
8. 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.
9. Bézivin, J., "On the unification power of models", *Software and System Modeling*, vol. 4, no. 2, pp. 171–188, 2005.
10. Bock, C., "UML 2 Composition Model", *Journal of Object Technology*, Vol. 3, No. 10, pp. 47-73, 2004.
11. Boley, H., "The Rule Markup Language: RDF-XML Data Model, XML Schema Hierarchy, and XSL Transformations", Invited Talk, INAP2001, Tokyo, Springer-Verlag, LNCS 2543, pp. 5-22, 2001.
12. Boley, H., Wagner, G., Tabet, S., Antoniou, G., "The Abstract Syntax of RuleML: Towards a General Web Rule Language Framework", Rule Markup Initiative (RuleML), Proc. of the 2004. IEEE/WIC/ACM Int. Conf. on Web Intelligence (WI'04). Beijing, pp. 628-631, 2004.
13. Brickley, D., Guha, R., V., "RDF Vocabulary Description Language 1.0: RDF Schema", W3C Working Draft. [Online]. Available: http://www.w3.org/TR/2003/WD-rdf-schema-20031010/.
14. Brockmans, S., Haase, P., "A Metamodel and UML Profile for Rule-extended OWL DL Ontologies - A Complete Reference", *Universität Karlsruhe (TH) - Technical Report*, 2006.
15. Czarnecki, K., Helsen, S., "Feature-based survey of model transformation approaches", in IBM Syst. J., Vol. 45, No. 3. (July 2006), pp. 621-645.
16. Duddy, K., Gerber, A., Lawley, M., Raymond, K., Steel, J., "Model transformation: A declarative, reusable patterns approach", *In Proceedings of the. 7th IEEE International Enterprise Distributed Object Computing Conference*, pp. 174-195, 2003.
17. Del Fabro, M., D., Valduirez, P., "Semi-automatic Model Integration using Matching Transformations and Weaving Models", In *Proceedings of the 2007 ACM Symposium on Applied Computing* (Seoul, Korea, March 11 - 15, 2007). SAC '07. ACM, New York, NY, 963-970..
18. Favre, J., M., "Towards a Basic Theory to Model Model Driven Engineering", *In Proc. of the UML2004 Int. Workshop on Software Model Engineering (WISME 2004)*, Lisbon, Portugal, 2004.
19. Fondement, F., Baar, B., "Making Metamodels Aware of Concrete Syntax", In Proceedings of the 1st European Conference on Model Driven Architecture (ECMDA): Fundamentals and Applications, LNCS 3748, Nuernberg, Germany, pp. 190-204, 2005.
20. 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.
21. Gašević, D., Djurić, D., Devedžić, V., "Model Driven Architecture and Ontology Development", Springer, Heidelberg, 2006.

22. Gašević, D., Djurić, D., Devedžić, V. “Bridging MDA and OWL ontologies”, *Journal of Web Engineering*, vol. 4, no. 2, pp. 119-134, 2005.
23. Ginsberg, A., “RIF Use Cases and Requirements”, *W3C Working Draft*, http://www.w3.org/TR/rif-ucr/, 2006.
24. Gómez, J., Cachero, C., Pastor, O., “LES Objects: A Model-based Code Generation Environment for Object-Oriented Conceptual Modeling of Web Application Interfaces”, *In Proceedings of the 16th European Conference on Object-Oriented Programming*, Málaga, Spain, 2002.
25. Grosof, B., N., Gandhe, M., D., Finin, T., W., “SweetJess: Translating DAMLRuleML to JESS”, *In Proceedings of the 1st International Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, Sardinia, Italy, 2002.
26. Haase, P, Brockmans, S., Palma, R., Euzenat, Z., d'Aquin, M., "D1.1.2 Updated Version of the Networked Ontology Model, NeOn Project Deliverable D1.1.2, August 2007, http://www.neon-project.org/web-content/index.php?option= com_ weblinks&task=view&catid=17&id=56.
27. Hirtle, D., Boley, H., Grosof, B., Kifer, M., Sintek, M., Tabet, S., Wagner, G., “*Schema Specification of RuleML 0.91*”, http://www.ruleml.org/spec/, 2006.
28. Hori, M., Euzenat, J., Patel-Schneider, F., P., “*OWL Web Ontology Language XML Presentation Syntax*”, W3C Note, 2003.
29. 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.
30. Jouault, F., Bézivin, J., Kurtev, I., “TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering”, *In Proceedings of the 5th International Conference on Generative programming and Component Engineering*, Portland, USA (in press), 2006.
31. Jovanović, J., Gašević, D., “XML/XSLT-Based Knowledge Sharing”, *Expert Systems with Applications*, Vol. 29, No. 3, pp. 535-553, 2005.
32. Java Metadata Interface (JMI) Specification, Sun Microsystems, JSR-000040, http://jcp.org/aboutJava/communityprocess/final/jsr040/index.html.
33. Kifer, M., Lausen, G., Wu, “J. Logical foundations of object oriented and frame-based languages”, *in Journal of the ACM 42*, 741–843, 1995.
34. Kurtev, I., Bézivin, J., Aksit, M., "Technological Spaces: an Initial Appraisal", CoopIS, DOA'2002, Industrial track, 2002.
35. Lassila, O., Swick, R., R., “Resource Description Framework (RDF) Model and Syntax Specification”, W3C Recommendation. [Online]. Available: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.
36. Matheus, C., J., "SWRLp: An XML-Based SWRL Presentation Syntax", In Proceedings of the 3rd International Workshop on Rules and Rule Markup Languages for the Semantic Web, Hiroshima, Japan, pp. 194-199, 2004.
37. McBride, B., “Jena: A Semantic Web Toolkit”, *IEEE Internet Computing*, vol. 6, no. 6, pp. 55-59, 2002.
38. 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.
39. Milanović, M., “Modeling Rules on the Semantic Web“, Master thesis, University of Belgrade, 2007.
40. Miller, J., Mukerji, J., (eds.) "MDA Guide Version 1.0.1", OMG, 2003.

41. Muller, P., A., Fleurey, F., Fondement, F., Hassenforder, M., Schneckenburger, R., Gérard, S., Jézéquel, J., M., "Model-Driven Analysis and Synthesis of Concrete Syntax", *In Proceedings of the ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems, LNCS 4199*, Genoa, Italy, pp. 98-110, 2006.
42. 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.
43. OMG Business Process Definition MetaModel (BPDM), Beta 2, OMG Adopted Specification, OMG Document Number: dtc/07-12-04, http://www.omg.org/cgi-bin/doc?dtc/2007-12-04, 2007.
44. OMG Object Constraint Language, *OMG Specification, Version 2.0, formal/06-05-01*, http://www.omg.org/docs/formal/06-05-01.pdf, 2006.
45. OMG Ontology Definition Metamodel (ODM), Sixth Revised Submission, OMG Document ad/2006-05-01, http://www.omg.org/docs/ad/06-05-01.pdf, 2006.
46. OMG Unified Modeling Language (UML) 2.0, Docs. formal/05-07-04 & formal/05-07-05, 2005.
47. OMG MOF QVT Final Adopted Specification, OMG document 05-11-01, 2005.
48. OMG 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.
49. Object-Oriented to Positional RuleML Translators http://www.ruleml.org/ooruleml-xslt/oo2prml.html, 2006.
50. *REWERSE I1 Rule Markup Language (R2ML)*, http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/6, 2006.
51. Relational-Functional Markup Language (RFML), http://www.relfun.org/rfml/, 2006.
52. Sendall, S., Kozaczynski, W., "Model Transformation: The Heart and Soul of Model-Driven Software Development", *IEEE Software 20*, no. 5, 42-45, 2003.
53. Seidewitz, E., "What Models Mean", IEEE Software, pp. 26-32, 2003.
54. Sintek, M., Decker, S., "TRIPLE - A Query, Inference, and Transformation Language for the Semantic Web", *in Proceedings of International Semantic Web Conference (ISWC)*, Sardinia, June 2002.
55. Translator from RuleML to Jess, http://www.ruleml.org/jess/, 2006.
56. Translators between RuleML and RFML, http://www.relfun.org/ruleml/rfml-ruleml.html, 2006.
57. Wagner, G., Giurca, A., Lukichev, S., "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.
58. Wagner, G., Damasio, C., V., Antioniou, G., "Towards a general web rule language", *International Journal of Web Engineering and Technology*, Vol. 2, Nos. 2/3, pp.181–206, 2005.
59. Wagner, G., Giurca, A., Lukichev, S., Antoniou G., Damasio C., V., Fuchs N., E., "Language Improvements and Extensions", *REWERSE I1-D8 deliverable*, http://rewerse.net/deliverables.html, 2006.
60. Warmer, J., Kleppe, A., The Object Constraint Language: Getting Your Models Ready for MDA, Second Edition, Addison Wesley, 2003.

Milan Milanović is a PhD candidate at University of Belgrade. His main research interests are rules, modeling and metamodeling, Service Oriented Architectures and Enterprise systems. His Internet address is http://milan.milanovic.org.

Dragan Gašević is a Canada Research Chair in semantic technologies and an Assistant Professor at the School of Computing and Information Sys-tems, Athabasca University, Canada. His current research interests are in the area of semantic tech-nologies, software language engineering and ser-vice oriented architectures. He can be reached at http://dgasevic.athabascau.ca

Adrian Giurca is a senior researcher at the Institute for Informatics of the Brandenburg University of Technology, Germany.He received in 2004 a doctoral degree from the University of Bucharest. He has been or is involved in research projects founded by the European Commission (FP6). He has been member of the Network of Excellence REWERSE (http://rewerse.net 2004-2008) of the 6th Framework Program of the European Commission. His main research interests are: Knowledge Bases, Logic Programming and Uncertainty, Rule Markup Languages and the Semantic Web (FLogic, R2ML, RIF, RDF, OWL) and UML. He can be reached at http://www.informatik.tu-cottbus.de/~agiurca/.

Sergey Lukichev is a Research Assistant in Institute of Informatics, Brandenburg University of Technology at Cottbus, Germany. His main research interests are: A UML-based rule modeling, Rule markup languages for the Semantic Web, Rule interchange and Rule verification. He holds a M.Sc. degree at University of Amsterdam. He can be reached at http://oxygen.informatik.tu-cottbus.de/~lukichev.

Gerd Wagner is a Professor of Internet Technolo-gy at the Brandenburg University of Technology at Cottbus, Germany. His research interests in-clude agent-oriented modeling and agent-based simulation, foundational ontologies, (business) rule technologies and the Semantic Web. He can be reached at http://www.informatik.tu-cottbus.de/~gwagner/.

Vladan Devedžić is a professor of computer science at the University of Belgrade, Serbia. His main research interests include software engineer-ing, intelligent systems, knowledge representation, ontologies, Semantic Web, intelligent reasoning, and applications of artificial intelligence tech-niques to education and medicine. He can be reached at http://devedzic.fon.rs.

Received: January 22, 2009; Accepted: August 26, 2009.