Model Transformations to Share Rules between SWRL and R2ML

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

[1] FON-School of Business Administration, University of Belgrade, Serbia
milan@milanovic.org, devedzic@etf.bg.ac.yu
[2] School of Interactive Arts and Technology, Simon Fraser University Surrey, Canada
dgasevic@sfu.ca
[3] Institute of Informatics, Brandenburg Technical University at Cottbus, Germany
Giurca@tu-cottbus.de, G.Wagner@tu-cottbus.de

Abstract. Currently, there is no generally adopted standard for a Semantic Web rule language, but there are several important evolving proposals such as RuleML, Semantic Web Rule Language (SWRL), and REWERSE Rule Markup Language (R2ML). Having that in mind, one may expect that various systems (e.g. Web services) will use different rule languages, and thus introduce problems in sharing rules. In this paper, we show how model-driven engineering techniques can be used to enable sharing rules between SWRL and R2ML. The main benefit of this approach is that the transformations between languages are completely based on the languages' abstract syntax (i.e., metamodels). The main benefit of this approach is that it keeps the focus on the language concepts rather than on technical issues caused by different concrete syntax. Yet, we also provide transformations that bridge between both languages' concrete (XML) and abstract (MOF) syntax.

1. Introduction

The Semantic Web is based on the use of ontologies that should provide an explicit definition of the domain conceptualization. Employing the rich AI research experience and being driven by practical needs for the use on the Web, the W3C has adopted the Web Ontology Language (OWL) as a standard ontology language [2]. Although the adoption of OWL means that Semantic Web applications can exchange their ontologies and tool vendors can develop reasoners and query languages over OWL, there is also a need to have some other mechanisms for defining knowledge. This is mainly manifested through advanced mechanisms for enriching ontologies by using rules. Thus, we should also define a standardized Semantic Web rule language that will be based on OWL to provide an additional reasoning layer on top of OWL. On the other hand, there are many Semantic Web applications that might use (OWL) ontologies whose business logic is implemented by using various rule languages (e.g., F-Logic, Jess, and Prolog) [29]. In this case, the primary goal is to have a rule exchange language for sharing rules, and hence enabling reusability of their business logics.

The above arguments motivated the research in the (Semantic) Web community to look at their different aspects. The most important proposal for the first group of rule language is Semantic Web Rule Language (SWRL) [11] that tends to be a standardized reasoning layer built on top of OWL. However, this is just one submission to such a language, while the research in Semantic Web services (e.g., WSMO and SWSL) introduces/relies on other rule languages besides SWRL such as SWSL-Rules or F-Logic [29]. In fact, this can be addressed by the second group of research efforts for Semantic Web rules manifested in the Rule Interchange Format (RIF) initiative [8], which tries to define a standard for sharing rules. That is, RIF should be expressive enough, so that it can represent concepts of various rule languages. Besides RIF, one should also develop a (two-way) transformation between RIF and any rule language that should be shared by using RIF. Currently, there is no official submission to RIF, but RuleML [9] and the REWERSE Rule Markup Language (R2ML) [30] are two well-known RIF proposals.

In this paper, we propose transformations between R2ML and SWRL to enable interchanging SWRL rules with various other rule languages (e.g., OCL) via R2ML. However, we want our solution to be completely based on the abstract syntax of both languages, unlike other similar approaches that mainly focus on a concrete syntax (mainly XML-Schema-based) without efficient mechanisms to check whether the implemented transformations are valid w.r.t. the abstract syntax. The problem of sharing rules is further hampered by the lack of a transformation language for the Semantic Web. Motivated by our positive experience with the recent OMG standard of the Ontology Definition Metamodel (ODM) [7] [23], we propose using Model-Driven Engineering (MDE) principles and model transformations to address this issue. Consequently, we define the abstract syntax of R2ML and SWRL by means of metamodels. While R2ML is actually fully built by using metamodeling principles, SWRL is not, but there is already a comprehensive metamodel for SWRL, the Rule Definition Metamodel (RDM) proposed in [5]. This means that our proposed model transformations between R2ML and RDM will be based on the abstract syntax of both languages. Our solution also covers mappings between the abstract syntax of both languages and their XML-based concrete syntax, but this is completely decoupled from the transformations between R2ML and RDM.

2. Motivation

In order to motivate sharing rules expressed in SWRL and R2ML, consider the following rule example: *Each person that is a qualified driver can be added to a car rental as additional driver*. The rule representation in the SWRL XML-based concrete syntax is shown in Fig. 1. Since SWRL defines a rule language on top of OWL, this rule presumes that there is an ontology defining *Rental*, *Person*, and *QualifiedDriver* classes and the property *additionalDriver*. Inheriting the parts of the RuleML syntax, this SWRL rule defines the antecedent of the rule by using the *ruleml:body* element, while the consequent is defined by using *ruleml:head*.

```
<ruleml:Implies
    xmlns:ruleml="http://www.ruleml.org/0.9/xsd"
    xmlns:owlx="http://www.w3.org/2003/05/owl-xml"
    xmlns:swrlx="http://www.w3.org/2003/11/swrlx"
    xmlns:srv="http://www.eurobizrules.org/ebrc2005/eurentcs">
     <ruleml:body>
       <swrlx:classAtom>
         <owlx:Class owlx:name="srv:Rental"/>
         <ruleml:var>rental</ruleml:var>
       </swrlx:classAtom>
       <swrlx:classAtom>
         <owlx:Class owlx:name="srv:Person"/>
         <ruleml:var>person</ruleml:var>
       </swrlx:classAtom>
       <swrlx:classAtom>
         <owlx:Class owlx:name="srv:QualifiedDriver"/>
         <ruleml:var>person</ruleml:var>
       </swrlx:classAtom>
     </ruleml:body>
     <ruleml:head>
       <swrlx:individualPropertyAtom
        swrlx:property="srv:additionalDriver">
         <ruleml:var>rental</ruleml:var>
         <ruleml:var>person</ruleml:var>
       </swrlx:individualPropertyAtom>
     </ruleml:head>
</ruleml:Implies>
```

Fig. 1. A SWRL rule: Each additional driver of a car rental must be a qualified driver

Given the great diversity of rule concepts and existing rule languages, the R2ML language supports the following types of rules: integrity, derivation, reaction, and production rules. This means, we first have to decide to what type of R2ML rules we should transform the above SWRL rule. Having in mind the nature of the SWRL rule above, which defines that something must hold under given conditions, we actually should transform the above rule onto an R2ML integrity rule, or more specifically alethic integrity rule [30]. In Fig. 2, we show the SWRL rule from Fig. 1 in the R2ML XML-based concrete syntax. This R2ML alethic rule has a universally quantified formula as its constraint, while this universally quantified formula is an implication whose antecedent is obtained from SWRL *ruleml:body* and consequent from SWRL *ruleml:head*.

Once we transform the SWRL rule into R2ML, we can further transform it onto all other rule languages supporting integrity rules by exploiting the existing transformations for R2ML [26] (e.g., OCL invariants [20]). In a similar way, we may translate SWRL derivation rules into their R2ML counterparts. Moreover, we can visualize the SWRL rule by using the UML-based Rule Modeling Language (URML) [17], since R2ML is employed for serialization of URML rules.

```
<r2ml:AlethicIntegrityRule xmlns:srv="http://www.eurobizrules.org/ebrc2005/eurentcs">
  <r2ml:constraint>
    <r2ml:UniversallyQuantifiedFormula>
      <r2ml:ObjectVariable r2ml:name="person" r2ml:classID="srv:Person"/>
      <r2ml:Implication>
        <r2ml:antecedent>
          <r2ml:Conjunction>
            <r2ml:ObjectClassificationAtom r2ml:classID="srv:Rental">
              <r2ml:ObjectVariable r2ml:name="rental"/>
            </r2ml:ObjectClassificationAtom>
            <r2ml:ObjectClassificationAtom r2ml:classID="srv:QualifiedDriver">
              <r2ml:ObjectVariable r2ml:name="person"/>
            </r2ml:ObjectClassificationAtom>
          </r2ml:Conjunction>
         </r2ml:antecedent>
         <r2ml:consequent>
            <r2ml:ReferencePropertyAtom
             r2ml:referencePropertyID="srv:additionalDriver">
              <r2ml:subject>
                <r2ml:ObjectVariable r2ml:name="person" r2ml:classID="srv:Person"/>
              </r2ml:subject>
              <r2ml:object>
                <r2ml:ObjectVariable r2ml:name="rental"/>
              </r2ml:object>
            </r2ml:ReferencePropertyAtom>
         </r2ml:consequent>
      </r2ml:Implication>
    </r2ml:UniversallyQuantifiedFormula>
  </r2ml:constraint>
</r2ml:AlethicIntegrityRule>
```

Fig. 2. An R2ML (alethic) integrity rule equivalent to the SWRL rule from Fig. 1

From the above example, it is obvious that in both cases we have been using XML-based concrete syntax. However, the language definition is done by using abstract syntax, while concrete (visual or textual) syntax is employed to represent physically rules. Thus, defining and implementing mappings between languages should be done on the level of their abstract syntax, as this actually allows us to focus on mappings between language constructs, rather than on the implementation details of their concrete syntax. We should mention that the Semantic Web community has already recognized the importance of using the software engineering metamodeling-based standards for defining the abstract syntax of ontology [23] and rule languages [5][31]. Being driven by this approach, in the rest of the paper, we describe mappings between R2ML and SWRL on the level of their abstract syntax, and yet bridge the gap between R2ML and SWRL's abstract and concrete syntaxes by using MDE principles.

3. Model Transformations for Semantic Web Rules

In this section, we describe the basic principles of MDE (e.g., metamodeling, MDA, and model transformations), how MDE standards are used for defining Semantic Web rule languages, and our solution for bridging between R2ML and SWRL.

3.1 Model Driven Engineering: Basics

Model Driven Engineering is a new software engineering discipline in which the process heavily relies on the use of models [4]. Models are the central MDE concepts and they are specified by using modeling languages (e.g., UML or ODM), while modeling languages are 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 [28]. The OMG's Model Driven Architecture (MDA) is a possible architecture for MDE [19]. MDA consists of three layers, namely: M1 (model) layer for defining models of systems under study; M2 (metamodel) layer for defining model languages (e.g., ODM is defined on this layer); and M3 (metametamodel) layer where only one metamodeling language is defined (i.e. MOF) [22]. The relations between different MDA layers can be considered as instance-of or conformant-to, which means that a model is an instance of a metamodel, and a metamodel is an instance of a metametamodel. The MDA architecture uses XML Metadata Interchange (XMI), the OMG's standard that defines mappings of MDA-based metametamodels, metamodels, and models onto XML documents and XML Schemas [25]. Another MDE architecture is Eclipse Modeling Framework (EMF), which is different from MDA just in using Ecore on the M3 layer instead of MOF.

Model transformations play an important role and represent the central operation for handling models in the MDA [19]. Model transformations are the process of producing one model from another model of the same system [19]. The OMG adopted the MOF2 Query View Transformation (QVT) specification [24] to address this need. In our research, we have decided to use ATLAS Transformation Language (ATL) [1] as the primary language and tool for model transformations, as it is one of the most important contributions to QVT, is the official Eclipse recommendation for model-to-model transformations, and yet is an open-source solution. However, the actual ATL implementation is different from the QVT standard. ATL integrates an important concept – technical spaces [16]. In our case, this is important when bridging between abstract and concrete syntax of the same rule language (e.g., between MOF-based based R2ML metamodel and R2ML XML-Schema). This practically means that ATL has tools for automatic injection/extraction of XML rules into/from the MOF representation. Therefore, we can use the same transformation language for bridging between a language's concrete and abstract syntax and between abstract syntax of different languages, which reduces the learning curve of technologies needed to provide a complete solution.

3.2 Ontology Definition Metamodel (ODM)

The OMG's ODM specification is closely related to our work [7] [23], as it specifies MOF-based metamodels for the Semantic Web ontology languages RDFS and OWL, i.e. it defines the abstract syntax of RDFS and OWL by using MOF. The ODM specification 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 models. However, all these transformations are defined at the level of metamodels, thus everything happens in the MOF technical space [7]. This means that, for example, there is a gap between the RDF/XML syntax [3] 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.

Following the ODM specification, Brockmans & Haase [5] proposed a Rule Definition Metamodel (RDM), based on ODM (see [6]), as an abstract syntax for SWRL. To the best of our knowledge, they have not provided mappings between RDM and SWRL XML-based concrete syntax or mappings between RDM and other rule languages. Here we address both of these issues.

3.3 Proposed Metamodel-based Solution

Our solution consists of two transformation steps. The first one (see Fig. 3) is from SWRL rules represented in the SWRL XML format (SWRL.xsd and OWL.xsd) [10] to models compliant with RDM [5].

Since SWRL is actually based on OWL, we also consider transforming OWL ontologies along with SWRL rules, that is, transforming the OWL XML-based concrete syntax onto ODM. This is done by using the XML injection (ATL feature) of OWL/SWRL documents by instantiating the MOF-based metamodel of XML. Then, such MOF-based SWRL rules (XML models) are transformed to models compliant with RDM. This transformation between the XML metamodel and RDM is implemented in both ways, and it is a bridge between SWRL concrete and abstract syntax.

Second, RDM-based models obtained in the previous step are transformed into R2ML models, which are compliant to the R2ML metamodel (see Fig. 4). This is actually a (two-way) transformation between the SWRL and R2ML abstract syntax and the core of our solution. R2ML models can later be serialized into R2ML XML-based concrete syntax or we can import R2ML XML-based syntax into the representation compliant with the R2ML metamodel. This has also been implemented and described elsewhere [21]. Having in mind all the above transformations, we have the core of the solution that is based on the abstract syntax, but we actually can transform between SWRL and R2ML XML-based rules.

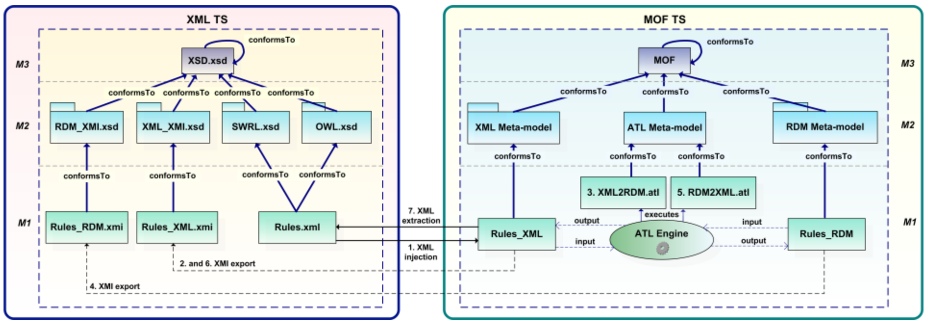

Fig. 3. First step in the transformation scenario: the OWL/SWRL XML format into the instances of the RDM metamodel

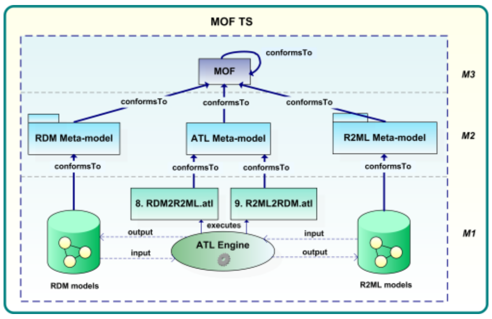

Fig. 4. Second step in the transformation scenario: the transformation of the models compliant to the RDM metamodel into the models compliant to the R2ML metamodel

4. Mappings between R2ML and SWRL

In this section, we first describe the parts of the R2ML abstract syntax relevant for representing SWRL rules, and consider how the R2ML language constructs correspond to elements of SWRL and OWL. We then describe mappings between these two languages in detail..

4.1 Definition of Abstract Syntax

The R2ML metamodel is defined by using the MOF metamodeling language. R2ML supports four kinds of rules, namely, integrity rules, derivation rules, production rules, and reaction rules. R2ML covers almost all of the use case requirements of the W3C

RIF WG [8]. Since SWRL rules can represent both integrity rules and derivation rules, we just describe R2ML integrity rules here. An *integrity rule*, also known as *(integrity) constraint*, consists of a constraint assertion, which is a sentence in a logical language such as first-order predicate logic or OCL (see Fig. 5). The R2ML framework supports two kinds of integrity rules: the *alethic* and *deontic* ones. An alethic integrity rule can be expressed by a phrase, such as "*it is necessarily the case that*" and a deontic one can be expressed by phrases, such as "*it is obligatory that*" or "*it should be the case that*."

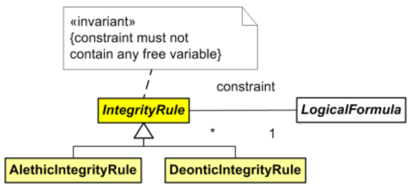

Fig. 5. The metamodel of integrity rules

The corresponding *LogicalFormula* must have no free variables, that is, all the variables from this formula must be quantified. R2ML defines the general concept of *LogicalFormula* (see Fig. 6) that can be *Conjunction*, *Disjunction*, *NegationAsFailure*, *StrongNegation*, and *Implication*. The concept of a *QuantifiedFormula* is essential for R2ML integrity rules, and it subsumes existentially quantified formulas and universally quantified formulas. Fig. 6 also contains elements such as *AtLeastQuantifiedFormula*, *AtMostQuantifiedFormula*, and *AtLeastAndAtMostQuantifiedFormula* for defining cardinality constraints with R2ML rules.

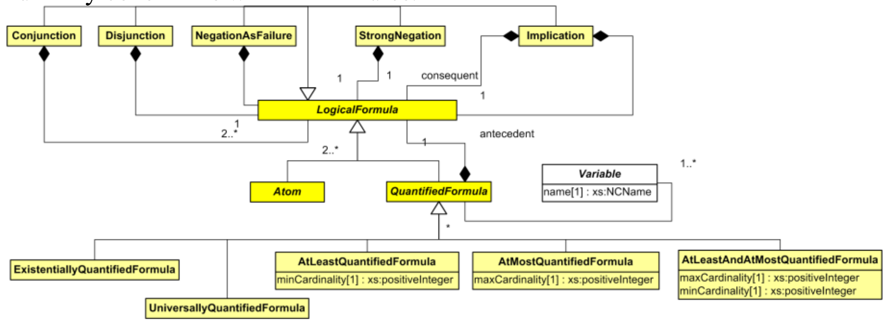
Conjunction
Disjunction
NegationAsFailure
StrongNegation
Implication
1
1
consequent
1
LogicalFormula
2..*
1
2..*
1
antecedent
1..*
Variable
name[1] : xs:NCName
Atom
QuantifiedFormula
*
ExistentiallyQuantifiedFormula
UniversallyQuantifiedFormula
AtLeastQuantifiedFormula
minCardinality[1] : xs:positiveInteger
AtMostQuantifiedFormula
maxCardinality[1] : xs:positiveInteger
AtLeastAndAtMostQuantifiedFormula
maxCardinality[1] : xs:positiveInteger
minCardinality[1] : xs:positiveInteger

Fig. 6. The concept of a logical formula in R2ML

Atoms are basic constituents of formulas in R2ML. Atoms are compatible with all important concepts of OWL/SWRL. R2ML distinguishes object atoms, data atoms, and generic atoms. Here we present just R2ML atoms necessary for our goal, that is, atoms used in SWRL – object and data (see [26] for a complete description and use of all supported atoms). An R2ML *ObjectClassificationAtom* refers to a class and consists of an object term (see Fig. 7). Its role is for object classification, i.e., an *ObjectTerm* is an instance of the referred class.

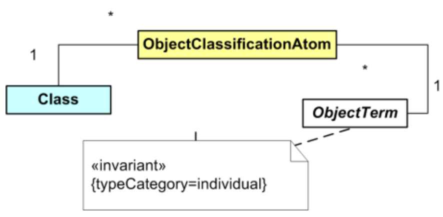

Fig. 7. R2ML ObjectClassificationAtom

A *ReferencePropertyAtom* associates an object term as *“subject”* with another object term as “*object*” (see Fig. 8). It corresponds to the UML and OWL concept of an object-valued property.

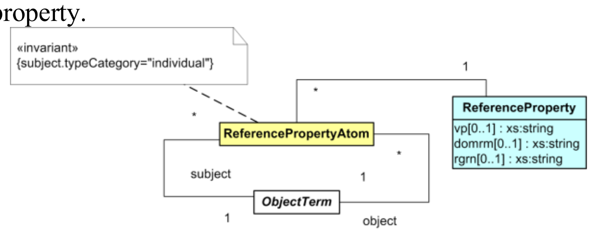

Fig. 8. R2ML ReferencePropertyAtom

Terms are the basic constituents of atoms. Similarly 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* (see Fig. 9).

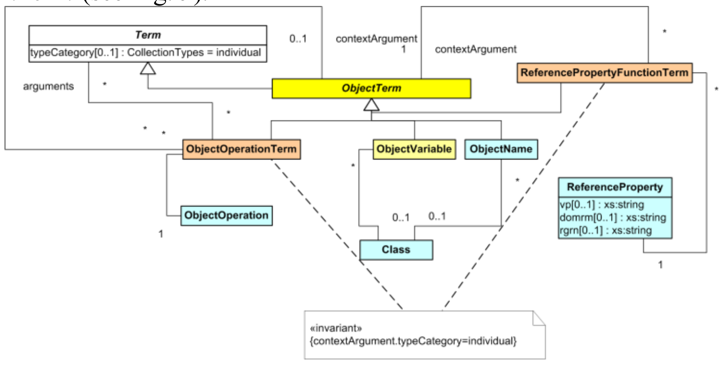

Fig. 9. R2ML Object Terms

An *ObjectOperationTerm* is formed with the help of a *contextArgument*, a user-defined operation, and an ordered collection of arguments. The *RoleFunctionTerm* corresponds to a functional association end (of a binary association) in a UML class model. *ObjectNames* in R2ML are the same artifacts like *Object* in UML. They also correspond to the *Individual* concept of OWL. *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).

The concept of data value in R2ML is related to the RDF concept of data literal. Following OWL, R2ML distinguishes between plain and typed literals. A *DataTerm* is a *DataVariable*, a *DataLiteral*, or *DataFunctionTerm*, which can be of three different types: *DataOperationTerm*, *AttributeFunctionTerm*, and *DatatypeFunctionTerm*.

We have already mentioned that we use RDM as an abstract syntax for SWRL language, but instead of using it with the ODM proposed in [6], we have adapted it to rely on the standard OMG ODM [23]. Due to space constraints for the paper, we do not explain RDM here in detail, but refer readers to the complete reference given in [5]. We just explain how RDM defines rules. The RDM *Rule* concept is a subclass of *OntologyElement*, while *OntologyElement* is defined in the ODM metamodel [23] as an *element* of the ODM *Ontology* class. An RDM (as well as SWRL) rule consists of an *antecedent* and a *consequent*, also referred to as the body and head of the rule, respectively. Both the RDM antecedent and consequent consist of a set of atoms which can possibly be empty. In the rest of the section, we define mappings between RDM and R2ML.

4.2 Conceptual mappings between SWRL and R2ML

In order to share rules between SWRL and R2ML, we define mappings between the constructs of SWRL and R2ML on the level of their abstract syntax. Every SWRL (i.e., RDM) rule (i.e., *Implies* element) is mapped to an R2ML *AlethicIntegrityRule* whose *constraint* is a *UniversallyQuantifiedFormula* and that formula is an *Implication.* In Tables 1, 2 and 3, we show mappings between SWRL and R2ML atoms in detail. As OWL (as well as SWRL/RDM) and R2ML distinguish between data values and objects, we accordingly divided mappings in the first two tables. In all the mappings shown in the tables, CD represents Class Description from [10], an expression ***T*** is a translation operator of a SWRL element to an R2ML element, and t is a variable. We should notice that SWRL rules semantics in these mappings is completely preserved.

Table 1. Mappings of SWRL Classification Atoms to R2ML Atoms

SWRL expression	R2ML expression
ClassAtom(classID, t)	ObjectClassificationAtom(classID, t)
ClassAtom(UnionOf(CD1, CD2), t)	Disjunction(***T***(ClassAtom(CD1, t)), ***T***(ClassAtom(CD2, t)))
ClassAtom(IntersectionOf(CD1, CD2), t)	Conjuction(***T***(ClassAtom(CD1, t)), ***T***(ClassAtom(CD2, t)))
ClassAtom(ComplementOf(CD), t)	StrongNegation(***T***(ClassAtom(CD, t)))
ClassAtom(OneOf({objID1,...,objIDn}), t)	Disjunction(EqualityAtom(objID1, t),..., EqualityAtom(objIDn ,t))
ClassAtom(ObjectRestriction(objPropID, allValuesFrom(CD)), t)	UniversallyQuantifiedFormula(x, Implication(ReferencePropertyAtom(objPropID, t, x), ***T***(ClassAtom(CD, t)))
ClassAtom(ObjectRestriction(objPropID, someValuesFrom(CD)), t)	ExistentiallyQuantifiedFormula(x, Conjuction(***T***(ClassAtom(CD, t)), ReferencePropertyAtom(objPropID, t, x)))
ClassAtom(ObjectRestriction(objPropID, hasValue(objID)), t)	ReferencePropertyAtom(objPropID, t, objID)
ClassAtom(ObjectRestriction(objPropID, mincardinality(n)), t)	AtLeastQuantifiedFormula(n, x, ReferencePropertyAtom(objPropID, t, x))
ClassAtom(ObjectRestriction(objPropID, maxcardinality(n)), t)	AtMostQuantifiedFormula(n,x, ReferencePropertyAtom(objPropID, t, x))
ClassAtom(ObjectRestriction(objPropID, mincardinality(m), maxcardinality(n)), t)	AtLeastAndAtMostQuantifiedFormula(m,n,x, ReferencePropertyAtom(objPropID, t, x))
ClassAtom(ObjectRestriction(objPropID, cardinality(n)), t)	AtLeastAndAtMostQuantifiedFormula(n,n,x, ReferencePropertyAtom(objPropID, t, x))

Table 2. Mappings of SWRL Datarange Atoms to R2ML Atoms

SWRL expression	R2ML expression
DatarangeAtom(datatypeID, t)	DataClassificationAtom(datatypeID, t)
DatarangeAtom(OneOf({objID1,...,objIDn}), t)	Disjunction(DatatypePredicateAtom(swrlb:equal, objID1, t), ... DatatypePredicateAtom(swrlb:equal,objIDn,t))
DatarangeAtom(DataRestriction(dataPropID, allValuesFrom(dataTypeID)), t)	UniversallyQuantifiedFormula(x, Implication(AttributionAtom(dataPropID,t,x), ***T***(DatarangeAtom(datatypeID, t)))
DatarangeAtom(DataRestriction(dataPropID, someValuesFrom(datatypeID), t)	ExistentiallyQuantifiedFormula(x, Conjuction(***T***(DatarangeAtom(datatypeID,t)), AttributionAtom(dataPropID, t, x))
DatarangeAtom(DataRestriction(dataPropID, allValuesFrom(OneOf({dataLiteral}))), t)	UniversallyQuantifiedFormula(x, Implication(AttributionAtom(dataPropID,t,x), ***T***(DatarangeAtom(OneOf({objID1, ..., objIDn}), t))
DatarangeAtom(DataRestriction(dataPropID, someValuesFrom(OneOf({dataLiteral}))), t)	ExistentiallyQuantifiedFormula(x, Conjuction(***T***(DatarangeAtom(OneOf({objID1, ..., objIDn}), t)), AttributionAtom(dataPropID, t, x))
DatarangeAtom(DataRestriction(dataPropID, hasValue(dataLiteral), t)	AttributionAtom(dataPropID, t, dataLiteral)
DatarangeAtom(DataRestriction(dataPropID, mincardinality(n)), t)	AtLeastQuantifiedFormula(n, x, AttributionAtom(dataPropID, t, x))
DatarangeAtom(DataRestriction(dataPropID, maxcardinality(n)), t)	AtMostQuantifiedFormula(n, x, AttributionAtom(dataPropID, t, x))
DatarangeAtom(DataRestriction(dataPropID, mincardinality(m), maxcardinality(n)), t)	AtLeastAndAtMostQuantifiedFormula(m, n, x, AttributionAtom(dataPropID, t, x))

Table 3. Mappings of other SWRL Atoms to R2ML Atoms

SWRL expression	R2ML expression
IndividualvaluedPropertyAtom(objectID1, objectID2)	ReferencePropertyAtom(individualvaluedPropertyID, objectID1, objectID2)
DatavaluedPropertyAtom(objectID, dataLiteral)	AttributionAtom(datavaluedPropertyID, objectID, dataLiteral)
SameAs(objectID1, objectID2)	EqualityAtom(objectID1, objectID2)
DifferentFrom(objectID1, objectID2)	InequalityAtom(objectID1, objectID2)
BuiltIn(builtinID, t)	DatatypePredicateAtom(builtinID, t)

Using mappings between SWRL and R2ML shown in the tables above, we now illustrate the transformation process with the example SWRL rule from Fig. 1 and its corresponding R2ML rule from Fig. 2. The SWRL *Implies* element is transformed to an R2ML *AlethicIntegrityRule* with *UniversallyQuantifiedFormula* element as its constraint, where *UniversallyQuantifiedFormula* has an *Implication* for its formula. The SWRL atoms from the *body* element of the *Implies* element are transformed to a *Conjunction* of the R2ML atoms in the *antecedent* part of the R2ML *Implication* element, and the atom from the *head* part of the SWRL *Implies* element is transformed to the R2ML atom in the *consequent* part of the R2ML *Implication*. As it is shown in Table 1, SWRL *ClassAtom*s with *Class* as their predicate symbol are transformed to R2ML *ObjectClassificationAtoms*. In this case, it is important to point out that variables used in the SWRL *ClassAtoms* are transformed to R2ML *ObjectVariable* of the R2ML *ObjectClassificationAtom*. In a similar way, we transform other two

ClassAtoms from the SWRL rule shown in Fig. 1. The SWRL *individualPropertyAtom* is transformed to R2ML *ReferencePropertyAtom*, as it is shown in Table 3. Variables used in the SWRL *individualPropertyAtom* are transformed to *ObjectVariables*, as the *subject* and *object* of the R2ML *ReferenceProprетyAtom*.

5. Implementation Experience

In this section, we explain the transformation steps undertaken to transform between SWRL rules and R2ML. This is a full implementation of the mappings defined in the previous section. Here we refer to Fig. 3 and Fig. 4 from Section 3.3 in order to position each specific transformation/step in this process of transformation. As we have already said in Section 3.3, the transformation process between R2ML and SWRL is split into two major steps.

In the first step, we bridge between the SWRL XML concrete syntax and the SWRL abstract syntax (i.e., RDM). To do this, we first use the XML injector, (see Fig. 3, step 1: XML injection), a part of ATL that automatically transforms SWRL XML documents like the one given in Fig. 1 (without any manually written transformation) into the models conforming to the MOF-based XML metamodel that defines XML (e.g., Node, Element, and Attribute). Once we inject SWRL XML rules into a MOF-based representation (*Rules_XML* in Fig. 3), 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. 3, step 2: XMI export). This is again an integrated ATL feature that requires no manual work. Now we transform between XML models (Rules_XML from Fig. 3) and RDM-compliant models (Rules_RDM from Fig. 3). This actually requires writing two ATL transformations (Fig. 3, step 3: XML2RDM.alt and step 5: RDM2XML.atl), and hence this is the *bridge* between the SWRL XML-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 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 be later exported into SWRL XML concrete syntax (Fig. 3, step 6: XML export) and obtain the rules in the from given in Fig. 1, which is important when transforming R2ML rules into the SWRL XML concrete syntax. Note also that once we transform SWRL rules into the RDM representation, we can also export SWRL rules in the RDM XMI format (Fig. 7, step 7: XMI export), and thus we can share SWRL rules with any MOF-compliant repository. This is another important contribution to the RDM metamodel itself [5] that improves its practical value to be used by other MOF-based tools. Finally, say that we have also changed the RDM metamodel [5], so that it is now based on the standard ODM metamodel [23] instead of being based on the one defined in [6].

The second step is the core of our transformation between the SWRL abstract syntax (i.e., RDM) and the R2ML abstract syntax (Fig. 4, steps 8 and 9). This transformation step is fully based on the conceptual mappings between the elements of the RDM and R2ML metamodel described in Section 4. The transformations between the RDM metamodel and the R2ML metamodel are defined as a sequence of rules in the ATL language (Fig. 4, steps 8 and 9: RDM2R2ML.atl and R2ML2RDM.atl). In these ATL transformations, we use ATL constructs such as matched rules, (unique) lazy rules, and helpers. In order to illustrate a part of these transformations, let us consider a matched rule. It basically matches a given type of a source model element, and generates one or more kinds of target model elements. Fig. 10 gives an example of a matched rule, which is, in fact, an excerpt of the *RDM2R2ML.atl* transformation for RDM *individualPropertyAtom*s that are transformed into R2ML *ReferencePropertyAtom*s. Note also that an additional step can be performed in order to transform rules between the R2ML abstract syntax and the R2ML XML concrete syntax. We previously implemented this bridge [21] in a similar way like we have done it for SWRL, in this paper. This means that we can transform all rules between the R2ML concrete (Fig. 2) syntax and the R2ML abstract syntax. Hence, we provide the whole chain of transformations bridging between R2ML XML rules and SWRL XML rules, but the core of this bridge is done on the level of the abstract syntax of two rule languages making sure that all rules being shared are valid w.r.t. their abstract syntax.

```
rule IndividualPropertyAtom2ReferencePropertyAtom{
      from i : RDM!Atom (
                        i.name = 'IndividualPropertyAtom'
               )
      to refpropat : R2ML!ReferencePropertyAtom (
                        isNegated <- false,
                        referenceProperty <- i.hasPredicateSymbol,
                        subject <- thisModule.IndividualVariable2ObjectVariable( i.terms->last() ),
                        object <- thisModule.IndividualVariable2ObjectVariable( i.terms->first() )
                     )
}
```

Fig. 10. An excerpt of the ATL transformation: A matched rule that transforms an RDM *IndividualPropertyAtom* to an R2ML *ReferencePropertyAtom*

All the transformations mentioned are available at [27] and [32], while we have also implemented a Java API, so that one can use the transformations in any Java based applications.

6. Related work and Conclusion

The current transformation between the R2ML and SWRL abstract syntax fully captures the definition of SWRL, so that all SWRL constructs can be translated onto their counterparts in R2ML and they then can also be transformed back from R2ML to SWRL. However, we have yet not finalized the implementation of all OWL (i.e., ODM) constructs to their R2ML equivalents (basically this is just for classes and properties, while restrictions have already been covered). This means that all OWL constructs used in the SWRL can be transformed in the R2ML, but separate OWL ontology definition has not supported yet. Having in mind the nature of open-world inference that OWL is based on, this is also allowed in SWRL as well as in R2ML. Nevertheless, this may have consequences if we want to map such SWRL rules from R2ML, for example, into OCL constraints for which we strictly have to define all elements of the underlying vocabulary. For instance, let us take a look at the SWRL rule from Fig. 1. In that rule, we do not have any information about the first *individua-*

lPropertyAtom's property (i.e., *additionaDriver*), that is, there is no explicitly defined domain and range for this property. During the implementation of the transformations between R2ML and OCL, we realized that the obtained R2ML rule from Fig. 2 can not be translated into a valid OCL, since we can not determine the context of the OCL invariant. Once we define ontology and all the properties referred to in the original SWRL rules and transform them to elements of the R2ML Vocabulary, we can obtain correct OCL invariants such as the following one:

```
context Rental
    inv: self.additionalDriver->notEmpty() implies
            self.additionalDriver->forall(d | d.oclIsTypeOf(QualifiedDriver)
```

Currently, we are working on supporting mappings between R2ML Vocabulary and the complete definition of OWL and between R2ML Vocabulary and UML elements related to classes. Once we complete these transformations, we will be able to evaluate to what extent we can share the rules between OCL and SWRL via R2ML.

We have also mentioned in Section 4, that in the current implementation of transformations between the SWRL and R2ML languages (as well as OCL constraints), we transform all SWRL rules into corresponding R2ML integrity rules. However, some SWRL rules may be intended to represent explicit definitions of concepts, so they should be transformed into R2ML derivation rules. While a derivation rule represents an explicit constructive definition, an integrity rule rather complements a definition by defining the admissible knowledge states with respect to the concepts constrained by it. While the 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 without *NAF* (Negation as Failure). Generally, supporting the transformation of SWRL rules into R2ML derivation rules will only require using a different type of logical formulas, but much of the current transformation will be reused. Once we support derivation rules, it will be possible to translate SWRL rules into F-Logic, Jess, RuleML, since the present R2ML translators support transformation of derivation rules [27]. Nevertheless, there is an open issue how to determine automatically whether we should translate a SWRL rule into an integrity rule or into a derivation rule. This basically requires analyzing the context in which the rule is defined (i.e., based on the notion of ontology elements that the formulas of rules are based on).

To the best of our knowledge, there is no available solution to transforming rule languages on the level of their abstract syntax and by using model transformation languages. The main benefit of our solution is that the mappings between the abstract syntax of rule languages (e.g., R2ML and SWRL in our case) are completely independent of their concrete syntax. Thus, we do not have to reconsider the mappings between two different languages when supporting various concrete syntax of the languages under study. Since the mappings between the same language's abstract and concrete syntax are straightforward, the effort and the price of support for concrete syntax are lower. For example, in the case of SWRL, we may use two concrete syntax, namely, the RDF/XML concrete syntax [3] and the OWL XML presentation syntax [10]. Currently, we only support the first one, but when we implement the support for the second one, we will not reconsider the mappings between R2ML and SWRL, but only between SWRL RDF concrete syntax and RDM, i.e., SWRL abstract syntax.

A similar approach to ours is applied in the ODM specification [23] where the (model) transformations between OWL and the languages such as UML, Topic Maps, and ER models are defined at the level of their abstract syntax (i.e., metamodels). Our solution goes one step further and demonstrates how to bridge between concrete and abstract syntax of Semantic Web languages. Besides the obvious benefit of developing transformations between rule languages on the level of abstract syntax, the use of model transformations and languages such as ATL is more suitable than XSLT. Although, in principle, we could use XSLT to map between abstract syntax thanks to XMI in which all MOF-based metamodels can be stored, the available 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 [12]. 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 [18]. In the case of using model transformations and engines such as ATL, we also provide a convenient support for transforming semantic Web rules in the XMI format, and thus interoperability with MOF-based tools and integration with the current trends in software engineering.

In future research we plan to use the proposed approach to provide mappings between R2ML and OMG's initiatives for Semantics of Business Vocabulary and Business Rules (SBVR) and Production rules. This also nicely fits into the OMG's initiative for Business Rule Management (http://www.omg.org/busrulesmgmtrfi) by enabling the use of Semantic Web rules, which will be another important aspect of our future research. Note that the proposed solution also complements our efforts for sharing Semantic Web policies by using R2ML [13]. Providing the transformations between R2ML and the policy languages KAoS and Rei, we can further transform policies into metamodeling-based approaches to trust and security [15].

7. References

1. (2006). ATLAS Transformation Language (ATL). http://www.sciences.univ-nantes.fr/lina/atl.
2. Bechhofer, S. et al. (2004). "OWL Web Ontology Language Reference," *W3C Recommendation*, http://www.w3.org/TR/owl-ref/.
3. Beckett, D., Ed. (2004). "RDF/XML Syntax Specification (Revised)," *W3C Recommendation*, http://www.w3.org/TR/rdf-syntax-grammar/.
4. Bézivin, J. (2005). "On the unification power of models," *Software and System Modeling*, vol. 4, no. 2, pp. 171-188.
5. Brockmans, S. & Haase, P. (2006). "A Metamodel and UML Profile for Rule-extended OWL DL Ontologies - A Complete Reference," *Universität Karlsruhe (TH) - Technical Report*.
6. Brockmans, S., et al. (2004). "Visual Modeling of OWL DL Ontologies Using UML," *In Proc. of the 3rd Int'l Semantic Web Conference*, Hiroshima, Japan, 2004, pp. 198-213.
7. Gašević, D., et al., (2006). *Model Driven Architecture for Ontology Development*, Springer, Berlin-Heidelberg.
8. Ginsberg, A. (2006). "RIF Use Cases and Requirements," *W3C Working Draft*, http://www.w3.org/TR/rif-ucr/.

9. Hirtle, D., et al. (2006). "Schema Specification of RuleML 0.91," http://www.ruleml.org/spec/.
10. Hori, M., Euzenat, J., Patel-Schneider, F. P. (2003). "OWL Web Ontology Language, XML Presentation Syntax," *W3C Note*, http://www.w3.org/TR/owl-xmlsyntax/.
11. Horrocks, I., et al. (2004). "SWRL: A Semantic Web Rule Language Combining OWL and RuleML," *W3C Member Submission*, http://www.w3.org/Submission/SWRL.
12. Jovanović, J. & Gašević, D. (2005). "XML/XSLT-Based Knowledge Sharing," *Expert Systems with Applications*, vol. 29, no. 3, 2005, pp. 535-553.
13. Kaviani, N., Gašević, D., Hatala, M., Clement, D., Wagner, G. (2006). "Towards Unifying Rules and Policies for Semantic Web Services," *In Proceedings of the 3rd Annual LORNET Conf. on Intelligent, Interactive, Learning Object Repositories Network*, Montreal, Canada.
14. Klyne, G., Carroll J., Eds. (2004). "Resource Description Framework (RDF): Concepts and Abstract Syntax," *W3C Recommendation*, http://www.w3.org/TR/rdf-concepts/.
15. Koch, M. & Parisi-Presicce, F. (2006). "UML specification of access control policies and their formal verification," *Software and Systems Modeling*, vol. 5, no. 4, pp. 429-447.
16. Kurtev, I., Bézivin, J., & Aksit, M. (2002). "Technological Spaces: an Initial Appraisal," *In Proceedings of the CoopIS, DOA'2002 Federated Conferences*, Industrial track, Irvine, USA.
17. Lukichev, S. & Wagner, G. (2005). "Visual Rules Modeling," *In Proce. of the 6th International Andrei Ershov Mem. Conf. Perspectives of System Informatics*, Novosibirsk, Russia.
18. Matheus, C.J. (2004). "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.
19. Miller, J. & Mukerji, J., Eds. (2003). "MDA Guide Version 1.0.1," *OMG Doc. omg/03-06-01*, http://www.omg.org/cgi-bin/doc?omg/03-06-01.
20. Milanović, M., et al. (2006). "On Interchanging between OWL/SWRL and UML/OCL," *In Proceedings of the 6th Workshop. on OCL for (Meta-)Models in Multiple Application Domains (OCLApps)*, Genoa, Italy, pp. 81-95.
21. Milanović, M., et al. (2006). "Model transformations to bridge concrete and abstract syntax of Web rule languages: The R2ML experience," *International Journal of World Wide Web-Internet and Web Information Systems*, (submitted).
22. (2006). Meta Object Facility (MOF) Core, v2.0. *OMG Document formal/06-01-01*, http://www.omg.org/cgi-bin/doc?formal/2006-01-01.
23. (2006). OMG Ontology Definition Metamodel (ODM), Sixth Revised Submission. *OMG Document ad/2006-05-01*, http://www.omg.org/docs/ad/06-05-01.pdf.
24. (2005). MOF QVT Final Adopted Specification. *OMG document 05-11-01*, http://www.omg.org/docs/ptc/05-11-01.pdf.
25. (2005). 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.
26. (2006). REWERSE I1 Rule Markup Language (R2ML). http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/6.
27. (2006). R2ML Translators. http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/15.
28. Seidewitz, E. (2003). "What Models Mean," *IEEE Software*, vol., 20, no.5, pp. 26-32.
29. Sheth, A. et al. (2006). "Semantics to energize the full services spectrum," *Comm. of the ACM*, vol. 49, no. 7, pp. 55-61.
30. Wagner, G. et al. (2006). "A Usable Interchange Format for Rich Syntax Rules Integrating OCL, RuleML and SWRL," *In Proceedings of the Workshop of Reasoning on the Web*, Edinburgh, UK.
31. Wagner, G., et al. (2004). "The Abstract Syntax of RuleML - Towards a General Web Rule Language Framework," *In Proceedings of the IEEE/WIC/ACM International Conference on Web intelligence*, pp. 628-631.
32. http://www.eclipse.org/m2m/atl/atlTransformations/#R2ML2SWRL