A Usable Interchange Format for Rich Syntax Rules Integrating OCL, RuleML and SWRL

Gerd Wagner
Institute of Informatics
Brandenburg University of Technology at Cottbus
03046 Walther Pauer Str.2, Cottbus, Germany
G.Wagner@tu-cottbus.de

Adrian Giurca
Institute of Informatics
Brandenburg University of Technology at Cottbus
03046 Walther Pauer Str.2, Cottbus, Germany
Giurca@tu-cottbus.de

Sergey Lukichev
Institute of Informatics
Brandenburg University of Technology at Cottbus
03046 Walther Pauer Str.2, Cottbus, Germany
Lukichev@tu-cottbus.de

ABSTRACT

Rules are becoming increasingly important in business modeling and requirements engineering, and as a high level programming paradigm. In the area of rule modeling there are different developer communities like UML modelers and ontology architects. The former use rules in business modeling and in software development, while the latter use rules in collaborative Web applications. Each of them is using different rule languages and tools. Since a business rule is the same rule no matter in which language it is formalized, it is important to support the interchange of rules between different systems and tools.

This paper presents an interchange format for rules integrating the *Rule Markup Language* (RuleML), the *Semantic Web Rule Language* (SWRL) and the *Object Constraint Language* (OCL). These languages provide a *rich syntax* for expressing rules. This means that they support conceptual distinctions, such as distinguishing different types of terms and different types of atoms, which are not present in standard predicate logic. The interchange format is *usable* in the sense that it allows structure-preserving markup of all constructs of these different languages and does not force users to translate their rule expressions to a completely different language paradigm such as having to transform a function into a functional predicate. In the case of OCL rules this is achieved by developing a logical reconstruction of the 'unlogical' original OCL metamodel.

We consider three kinds of rules in this paper: integrity rules, derivation rules and production rules. We define rule concepts with the help of MOF/UML, a subset of the UML class modeling language proposed by the Object Management Group (OMG) for the purpose of 'meta-modeling', i.e. for defining languages conceptually on the level of an abstract (semi-visual) syntax.

Keywords

Rules, rule markup languages, integrity rules derivation rules, production rules, rule metamodels, OCL, RuleML, SWRL

1. INTRODUCTION

Rule markup languages will be the vehicle for using rules on the Web and in other distributed systems. They allow deploying, executing, publishing and communicating rules in a network. They may also play the role of a lingua franca for exchanging rules between different systems and tools. In a narrow sense, a rule markup language is a concrete (XML-based) rule syntax for the Web. In a broader sense, it should have an abstract syntax as a common basis for defining various concrete languages serving different purposes. The main purposes of a rule markup language is to permit reuse, interchange and publication of rules.

We adopt the Model Driven Architecture (MDA,[8]), which is a framework for distinguishing different abstraction levels defined by the Object Management Group (OMG), [11]. As illustrated in Figure 1, we consider rules at the three abstraction levels defined be the MDA:

At the **('computation-independent') business domain level** (called CIM in OMG's MDA), rules are statements that express (certain parts of) a business/domain policy (e.g., defining terms of the domain language or defining/constraining domain operations) in a declarative manner, typically using a natural language or a visual language. Examples of such rules are:

- *"The driver of a rental car must be at least 25 years old"*
- *"A gold customer is a customer with more than $1Million on deposit"*
- *"An investment is exempt from tax on profit if the stocks have been bought more than a year ago"*
- *"When a share price drops by more than 5% and the investment is exempt from tax on profit, then sell it"*

At the **platform-independent operational design level** (called PIM in OMG's MDA), rules are formal statements, expressed in some formalism or computational paradigm, which can be directly mapped to executable statements of a software system. Examples of rule languages at this level are SQL:1999 [15], OCL 2.0 [10] and DOM Level 3 Event Listeners [6]. Remarkably, SQL provides operational constructs for all three business rule categories mentioned above: checks and assertions operationalize a notion of integrity rules, views operationalize a notion of derivation rules, and triggers operationalize a notion of reaction rules.

Copyright is held by the author/owner(s).
WWW2006, May 22–26, 2006, Edinburgh, UK.
.

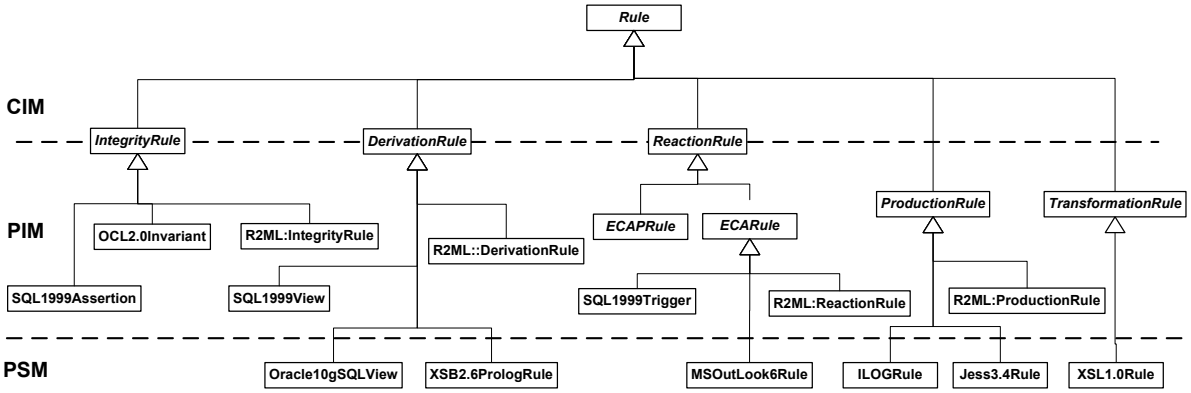

Figure 1: Rule concepts at three different abstraction levels: computation-independent (CIM), platform-independent (PIM) and platform-specific (PSM) modeling.

At the **platform-specific implementation level** (called PSM in OMG's MDA), rules are statements in a language of a specific execution environment, such as Oracle 10g views [12], Jess 3.4 [7], XSB 2.6 Prolog [18] or the Microsoft Outlook 6 Rule Wizard [9].

Rule interchange will be important both at the PIM and the PSM level. So, there are four interchange types:

PIM to PIM examples: OCL to SQL, SQL to ISO Prolog.

PIM to/from PSM examples: OCL to/from Java, SQL to/from Oracle 10g.

PSM to PSM examples: XSB Prolog to SWI Prolog, ILOG to ILOG, ILOG to Jess.

General purpose rule interchange formats, such as RuleML and R2ML, address the PIM level. They support a PSM to PSM interchange via the PIM level. Since there will be several rule interchange formats, there is also the issue of mapping them on each other.

In general, a rule interchange will not be loss-free. For instance, since RuleML cannot represent several linguistic distinctions made in OCL and SWRL, an OCL to SWRL interchange is not well supported by RuleML, while R2ML allows a loss-free interchange.

We assume that Web rule languages do not directly follow the tradition of predicate-logic-style rule languages such as Prolog, but rather follow the recent developments of Web knowledge representation languages such as RDF [13] and OWL [16]. This requires that they accommodate:

- *Web naming concepts*, such as URIs/IRIs and XML namespaces,
- *The ontological distinction between objects and data values*,
- *The datatype concepts of RDF.*

As a working name for our interchange format, we use the acronym R2ML standing for *REWERSE Rule Markup Language.*

2. CONTENT LANGUAGE

Logical formulas and rules are expressed with the help of logical connectives on the basis of a *content language* consisting of a *datatype language* and a user-defined *content vocabulary.*

2.1 Datatype Language

The datatype language consists of a set of predefined datatype names, including the name `rdfs:Literal` standing for the generic datatype of all Unicode strings. Each predefined datatype name is associated with:

- a set of **data literals**, which are Unicode strings;
- a set of **datatype function names**;
- a set of **datatype predicate names**.

2.2 Content Vocabulary Constructs

The user-defined content vocabulary includes

- user-defined **object names**;
- user-defined **object function names** comprising role function names and object property names;
- user-defined **data function names** comprising attributes and data operations;
- user-defined **noun concept names** standing for general noun concepts, among which we distinguish *object types* ('classes') and *datatypes*;
- user-defined **verb concept names**, called 'predicate symbols' in traditional logic, standing for general verb concepts, or *predicates*, among which we distinguish *properties* and *associations*; properties are either *attributes*, if they are data-valued, or *reference properties*, if they are object-valued.

In Web languages such as RDF and OWL, all these names are globally unique standard identifiers in the form of URI references. One of the goals of R2ML is to comply with important Semantic Web standards like RDF(S) and OWL. In particular, R2ML accommodates the data type concept of RDF.

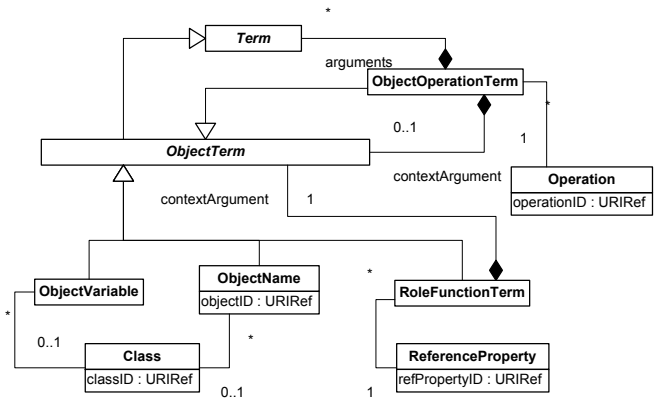

Figure 2: Object terms

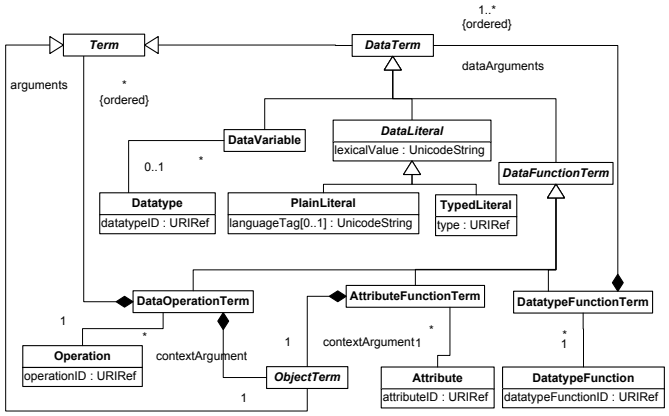

Figure 3: Data terms

2.2.1 User-Defined Object Functions

User-defined object functions are either *role functions* or *object operations*. They are used in *object terms* as shown in Figure 2.

A **role function** corresponds to a functional association end (of a binary association) in a UML class model. For example, in Figure 4, both association ends `pickupBranch` and `returnBranch` define role functions.

An **object operation** is a special type of user-defined function that corresponds to an object-valued non-state-changing (i.e. side-effect-free) operation in a UML class model. For example, in Figure 4, the operation `getLastRental()` defines an object operation.

2.2.2 User-Defined Data Functions

User-defined data functions are either *attributes* or *data operations*. They are used in *data terms* as shown in Figure 3.

An **attribute** is a special type of user-defined function that corresponds to a data-valued property in a UML class model. In Figure 4, `reservationDate` is an attribute of the class Rental.

A **data operation** is a special type of user-defined function that corresponds to a data-valued operation in a UML class model. In Figure 4, `age()` defines a data operation.

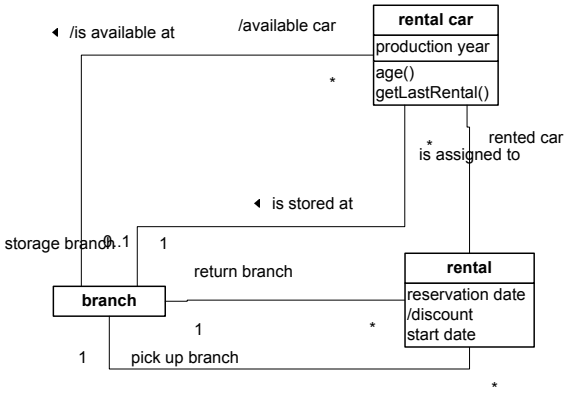

Figure 4: Sample vocabulary

2.2.3 User-Defined Noun Concepts

User-defined noun concepts comprise **classes** (or *object types*) and user-defined **datatypes** including enumerations.

2.2.4 User-Defined Verb Concepts

User-defined verb concepts comprise **properties** and **associations**. Properties are either *attributes*, if they are data-valued, or *reference properties*, if they are object-valued.

Notice that we use an attribute name both as the name of a function and as the name of the corresponding functional predicate. Likewise, we use a reference property name both as the name of a property predicate and as the name of the corresponding role function. This kind of naming liberty, which is supported by RDF and Common Logic, helps to switch between functional and relational languages.

3. TERMS

Terms are either *object terms* standing for *objects*, or *data terms* standing for *data values*. The concrete syntax of first-order non-Boolean OCL expressions can be directly mapped to our abstract concepts of *ObjectTerm* and *DataTerm*, depicted in Figures 2 and 3, which can be viewed as a predicate-logic-based reconstruction of the standard OCL abstract syntax.

3.1 Object Terms

An object term, as shown in Figure 2, is either an *object variable*, an *object name* (also called 'constant symbol' in traditional predicate logic and 'object identifier' in object-oriented programming), a *role function term* or an *object operation term*.

An *ObjectOperationTerm* refers to an object operation and has an object term as an optional context argument and zero or more object terms and data terms as arguments.

A *RoleFunctionTerm* refers to a role (reference property), and takes an object term as context argumnet.

Example 1 (ObjectOperationTerm). *The expression* `x.getLastRental()`*, which returns the last rental of a rental car, is an* ObjectOperationTerm*,* `getLastRental()` *denotes an operation and* `x` *is the context argument (see Figure 4).*

Example 2 (RoleFunctionTerm).
The expression `x.pickupBranch`*, where* pickupBranch *is a*

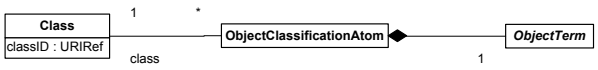

Figure 6: Object classification atoms

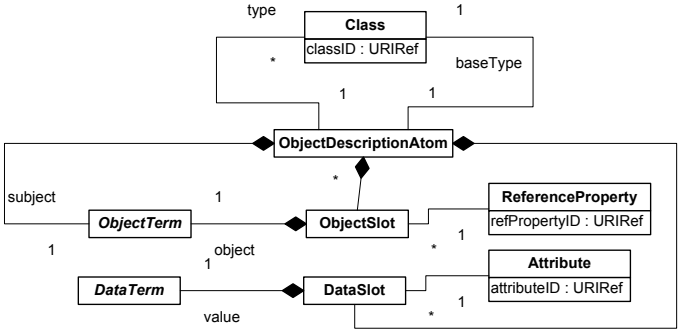

Figure 7: Object description atoms

role name of a Branch in association between classes Rental and Branch (Fig. 4), is a RoleFunctionTerm*, where* `x` *is an object term and* `pickupBranch` *is a reference property.*

3.2 Data Terms

A data term (cf. Figure 3) is either a data variable, a data literal, or a *data function term*, which can be of three different types:

1. A *datatype function term* formed with the help of a datatype function that comes with the corresponding datatype.
2. An *attribute function term* formed with the help of a user-defined attribute.
3. A *data operation term* formed with the help of a user-defined non-state-changing data operation.

4. ATOMS

The basic logical constituents of a rule are *atomic formulas*, called 'atoms'. In R2ML we define 9 types of atoms, including two convenience constructs:

-*Object description atom*, representing a conjunctive aggregate of a number of simpler atoms (like an RDF description).

-*Inequality atom*, representing a negation of an equality atom. All atoms from our framework are presented in Figure 5.

An *object classification atom* (Figure 6) refers to a class and consists of an object term.

Following RDF [13] and OWL [16], we adopt the concept of an *object description atom*. An *object description atom* (Figure 7) refers to a class as a base type and to zero or more classes as categories, and consists of a number of property/term pairs (attribute data term pairs and reference property object term pairs). Any instance of such atom refers to one particular object, that is referenced by an objectID, if it is not anonymous.

An *attribution atom* (Figure 8) consists of an object term as "subject", and a data term as "value".

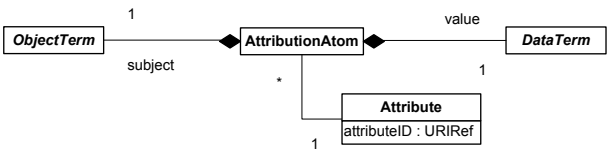

Figure 8: Attribution atoms

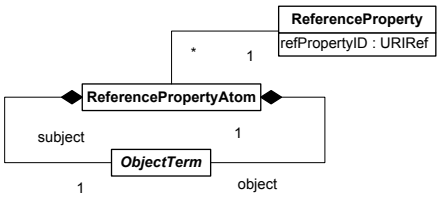

Figure 9: Reference property atoms

A *reference property atom* (Figure 9) associates object terms as "subjects" with other object terms as "objects".

In order to support common fact types of natural language directly, it is important to have n-ary predicates (for $n > 2$).

An *association atom* (Figure 10) is constructed using an n-ary predicate as association predicate, a collection of data terms as "data arguments" and a collection of object terms as "object arguments".

An *equality atom* or *inequality atom*, see Figure 11, is composed of two or more object terms.

A *data classification atom* (Figure 12) consists of a data term and refers to a data type.

A *datatype predicate atom* (Figure 13) refers to a datatype predicate, and consists of a number of data terms.

5. FORMULAS

R2ML provides two abstract concepts for formulas: the concept of AndOrNafNegFormula (see Figure 14), which represents the most general quantifier free logical formula with weak and strong negations, and the concept of LogicalFormula (see Figure 15), which corresponds to a general first order formula.

The distinction between strong negation and negation-as-failure is used in several computational languages: it is presented in explicit form in extended logic programs [3], only implicitly in SQL [15] and OCL [10], as was shown in [5]. Intuitively speaking, weak negation captures the absence of positive information, while strong negation captures the presence of explicit negative information (in the sense of

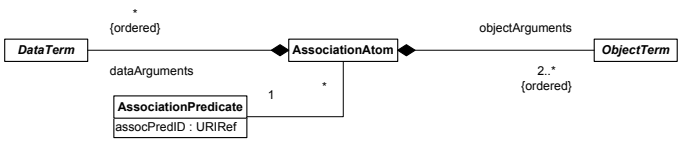

Figure 10: An association atom can express an n-ary association between classes and datatypes.

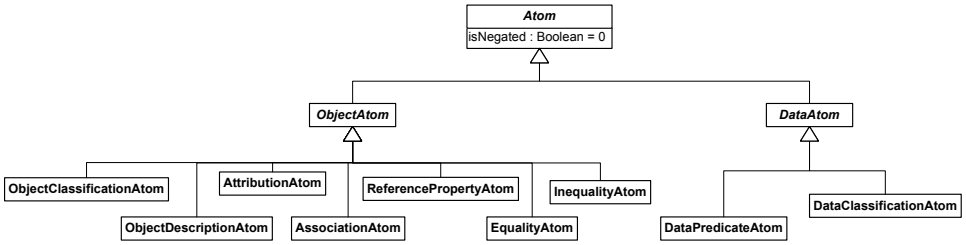

Figure 5: Atoms types

Figure 11: Equality and inequality atoms

Figure 12: Data classification atoms

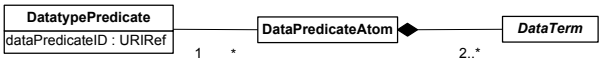

Figure 13: Datatype predicate atoms

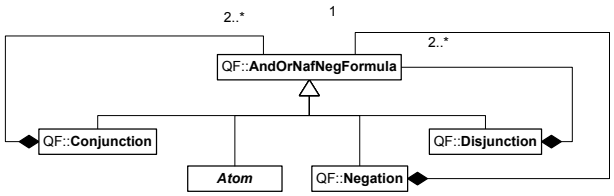

Figure 14: AndOrNafNegFormula

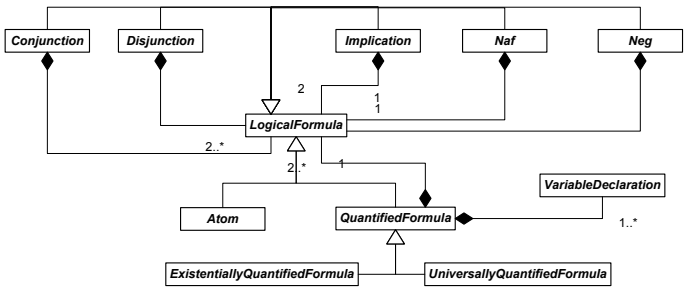

Figure 15: Logical formula

Kleene's 3-valued logic). Under the minimal/stable model semantics [2] weak negation captures the computational concept of negation-as-failure (or closed-world negation) [1].

6. ACTIONS

The REWERSE I1 Rule Markup Language (R2ML) offers support both for production rules and reaction rules. With this respect it defines the concept of an *action*. Following the OMG Production Rule Representation submission, an action (Fig. 16) is either an *InvokeActionExpression* or an *AssignActionExpression* or a *CreateActionExpression* or a *DeleteActionExpression*. R2ML also provides message actions in the form of a concrete SOAPAction.

All actions refer to a *context* which is an R2ML object term.

InvokeActionExpression models the receipt by an object of a request invoking a call of an operation. It refers to an R2ML *Operation* and contains an ordered, possible empty list of arguments represented as R2ML terms. The execution of this action is done by the corresponding operation-call.

AssignActionExpression refers to an UML *Property* and contains a DataTerm as a *value*. This action assigns a value to a property.

CreateAction refers to an R2ML *Class* and contains a list of slots (object slots and/or data slots). The execution of this action is a constructor call for creation of a new object in the system.

DeleteActionExpression refers to an UML *Class* and contains an *ObjectTerm*. This action removes an instance of the *Class*.

7. RULES

7.1 Integrity Rules

Integrity rules, also known as (integrity) constraints, consist of a constraint assertion, which is a sentence in a logical language such as first-order predicate logic or OCL (see Figure 17). R2ML framework supports two kinds of integrity rules: the *alethic* and the *deontic* ones. The alethic integrity rule can be expressed by a phrase, such as "it is necessarily the case that" and the deontic one can be expressed by phrases, such as "it is obligatory that" or "it should be the case that". A *LogicalStatement* is a *LogicalFormula* that has no free variables i.e. all the variables from this formula are quantified.

7.2 Derivation Rules

R2ML derivation rules have "conditions" and "conclusions" (see Figure 18). Conditions of a derivation rule are

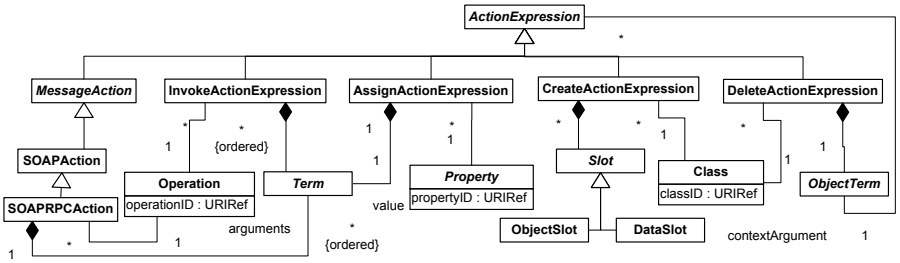

Figure 16: Actions

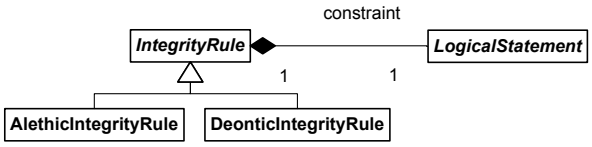

Figure 17: Integrity rule metamodel

Figure 18: Derivation rule metamodel

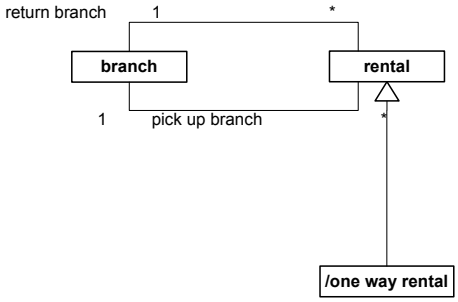

Figure 20: Business vocabulary

AndOrNafNegFormula, as defined in Figure 14. Conclusions are restricted to DNF conjuncts (disjunction of literal conjunctions).

7.3 Production Rules

Production rules have "conditions", "post-conditions" and a "produced action" (see Figure 19). Conditions and post-conditions of a production rule are *LogicalFormula*, as defined in Figure 15. Production rule may execute an *Action*. The actions are defined in Figure 16.

8. RULEML, SWRL AND OCL

The terminological correspondence between SWRL, RuleML, OCL and R2ML is presented in Tables 1 and 2.

9. RULE EXAMPLES

EXAMPLE 3. *The following integrity rule is based on the business vocabulary, depicted on Fig. 20: "If rental is not a one way rental then return branch of rental must be the same as pick-up branch of rental."*

```
<r2ml:AlethicIntegrityRule r2ml:id="AIT1001">
```

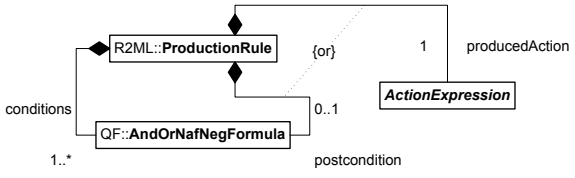

Figure 19: Production rule metamodel

```
<r2ml:constraint>
 <r2ml:Implication>
  <r2ml:Conjunction>
   <r2ml:ObjectClassificationAtom
       r2ml:classID="srv:Rental">
    <r2ml:ObjectVariable r2ml:name="rental1"/>
   </r2ml:ObjectClassificationAtom>
   <r2ml:ObjectClassificationAtom
       r2ml:classID="srv:OneWayRental"
       r2ml:isNegated="true">
    <r2ml:ObjectVariable r2ml:name="rental1"/>
   </r2ml:ObjectClassificationAtom>
  </r2ml:Conjunction>
  <r2ml:Conjunction>
   <r2ml:ReferencePropertyAtom
       r2ml:refPropertyID="srv:returnBranch">
    <r2ml:subject>
     <r2ml:ObjectVariable r2ml:name="rental1"/>
    </r2ml:subject>
    <r2ml:object>
     <r2ml:ObjectVariable r2ml:name="returnBranch"/>
    </r2ml:object>
   </r2ml:ReferencePropertyAtom>
   <r2ml:ReferencePropertyAtom
       r2ml:refPropertyID="srv:pickupBranch">
    <r2ml:subject>
     <r2ml:ObjectVariable r2ml:name="rental1"/>
    </r2ml:subject>
    <r2ml:object>
     <r2ml:ObjectVariable r2ml:name="pickupBranch"/>
    </r2ml:object>
   </r2ml:ReferencePropertyAtom>
   <r2ml:EqualityAtom>
    <r2ml:ObjectVariable r2ml:name="returnBranch"/>
    <r2ml:ObjectVariable r2ml:name="pickupBranch"/>
   </r2ml:EqualityAtom>
  </r2ml:Conjunction>
 </r2ml:Implication>
</r2ml:constraint>
```

Table 1: Basic language constructs for vocabularies

R2ML	RuleML	SWRL	UML/OCL
data variable	variable	data variable	Variable
object variable	variable	individual variable	Variable
data term	term	data term	Literal
object term	term	object term	Object
association predicate	n/a	n/a	Association
class	n/a	owl description	Class
attribute	n/a	iObject	Property
data predicate	rel	n/a	Property

Table 2: Accommodating different atoms concepts

R2ML	RuleML	SWRL	UML/OCL
ObjectDescriptionAtom	SlotAtom	n/a	n/a
ObjectClassificationAtom	PositionalAtom	classAtom	≪ instanceOf ≫
AssociationAtom	PositionalAtom	n/a	link
AttributionAtom	PositionalAtom	datavaluedPropertyAtom	property instance
ReferencePropertyAtom	PositionalAtom	individualPropertyAtom	link
BuiltinPredicate	n/a	builtinAtom	Operation
DataClassificationAtom	n/a	dataRangeAtom	property instance
EqualityAtom	n/a	sameIndividualAtom	==
InequalityAtom	n/a	differentIndividualsAtom	!=

```
</r2ml:AlethicIntegrityRule>
```

Moreover, our language permits to rewrite the same constraint in a functional style:

```
<r2ml:AlethicIntegrityRule r2ml:id="AIT1001">
<r2ml:constraint>
 <r2ml:Implication>
  <r2ml:Conjunction>
   <r2ml:ObjectClassificationAtom
           r2ml:classID="srv:Rental">
    <r2ml:ObjectVariable r2ml:name="rental1"/>
   </r2ml:ObjectClassificationAtom>
   <r2ml:ObjectClassificationAtom
      r2ml:classID="srv:OneWayRental"
      r2ml:isNegated="true">
    <r2ml:ObjectVariable r2ml:name="rental1"/>
   </r2ml:ObjectClassificationAtom>
  </r2ml:Conjunction>
  <r2ml:Conjunction>
   <r2ml:EqualityAtom>
     <r2ml:RoleFunctionTerm
          r2ml:refPropertyID="returnBranch">
      <r2ml:argument>
       <r2ml:ObjectVariable r2ml:name="rental1"/>
      </r2ml:argument>
     </r2ml:RoleFunctionTerm>
     <r2ml:RoleFunctionTerm
            r2ml:refPropertyID="pickupBranch">
      <r2ml:argument>
       <r2ml:ObjectVariable r2ml:name="rental1"/>
      </r2ml:argument>
     </r2ml:RoleFunctionTerm>
    </r2ml:EqualityAtom>
   </r2ml:Conjunction>
  </r2ml:Implication>
 </r2ml:constraint>
</r2ml:AlethicIntegrityRule>
```

EXAMPLE 4. *The following derivation rule is based on the business vocabulary, depicted on Fig. 21:*

> *A car is available for rental if it is not assigned to any rental and is not scheduled for service.*

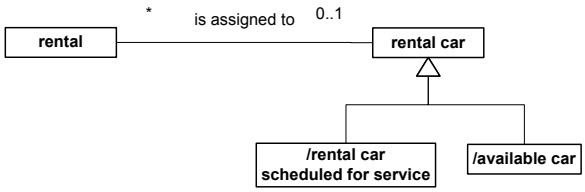

Figure 21: Business vocabulary

The first condition of this rule "a rental car is not assigned to a rental", corresponds to a negation-as-failure, which is expressed by the tag `<WeakNegation>`. *The second condition,* `not scheduled for service` *is a categorization and corresponds to a strong negation, because it requires the value to be explicitly false.*

```
<r2ml:DerivationRule
    r2ml:id="DR001"
    xmlns:srv="http://www.services.org/EU-Rent/">
 <r2ml:conditions>
  <r2ml:ObjectClassificationAtom
    r2ml:classID="srv:RentalCar">
   <r2ml:ObjectVariable r2ml:name="rentalCar1"/>
  </r2ml:ObjectClassificationAtom>
  <r2ml:ObjectClassificationAtom
    r2ml:classID="srv:RentalContract">
   <r2ml:ObjectVariable r2ml:name="rentalContract"/>
  </r2ml:ObjectClassificationAtom>
  <r2ml:qf.WeakNegation>
   <r2ml:ReferencePropertyAtom
        r2ml:refPropertyID="srv:isAssignedTo">
     <r2ml:subject>
       <r2ml:ObjectVariable r2ml:name="rentalCar1"/>
     </r2ml:subject>
     <r2ml:object>
       <r2ml:ObjectVariable r2ml:name="rentalContract"/>
     </r2ml:object>
   </r2ml:ReferencePropertyAtom>
   </r2ml:qf.WeakNegation>
```

```
   <r2ml:ObjectClassificationAtom
       r2ml:classID="srv:rentalCarScheduledForService"
       r2ml:isNegated="true">
     <r2ml:ObjectVariable r2ml:name="rentalCar1"/>
   </r2ml:ObjectClassificationAtom>
 </r2ml:conditions>
 <r2ml:conclusion>
  <r2ml:qf.LiteralConjunction>
   <r2ml:ObjectClassificationAtom
        r2ml:classID="srv:isAvailable">
    <r2ml:ObjectVariable r2ml:name="r1"/>
   </r2ml:ObjectClassificationAtom>
  </r2ml:qf.LiteralConjunction>
 </r2ml:conclusion>
</r2ml:DerivationRule>
```

EXAMPLE 5 (PRODUCTION RULE: ASSIGN DISCOUNT). *"If the order has value greater than 1000 and the order's customer type is not "gold" then assign a discount of 10%."*

```
<r2ml:ProductionRule
        xmlns="http://www.services.org/EU-Rent/">
 <r2ml:conditions>
  <r2ml:qf.Conjunction>
   <r2ml:DataPredicateAtom
        r2ml:dataPredicateID="swrlb:greaterThan">
     <r2ml:dataArguments>
      <r2ml:AttributeFunctionTerm
            r2ml:attributeID="orderValue">
       <r2ml:contextArgument>
        <r2ml:ObjectVariable
            r2ml:name="order"
            r2ml:classID="srv:Order"/>
       </r2ml:contextArgument>
      </r2ml:AttributeFunctionTerm>
      <r2ml:TypedLiteral
            r2ml:lexicalValue="1000"
            r2ml:typeLiteral="xs:positiveInteger"/>
     </r2ml:dataArguments>
   </r2ml:DataPredicateAtom>
   <r2ml:DataPredicateAtom
            r2ml:dataPredicateID="swrlb:equal"
            r2ml:isNegated="true">
    <r2ml:dataArguments>
     <r2ml:AttributeFunctionTerm
            r2ml:attributeID="customerRating">
      <r2ml:contextArgument>
       <r2ml:ObjectVariable
            r2ml:name="order"
            r2ml:classID="srv:Order"/>
      </r2ml:contextArgument>
     </r2ml:AttributeFunctionTerm>
     <r2ml:TypedLiteral
            r2ml:lexicalValue="gold"
            r2ml:typeLiteral="xs:string"/>
    </r2ml:dataArguments>
   </r2ml:DataPredicateAtom>
  </r2ml:qf.Conjunction>
 </r2ml:conditions>
 <r2ml:producedAction>
  <r2ml:AssignActionExpression
       r2ml:propertyID="srv:discount">
   <r2ml:contextArgument>
    <r2ml:ObjectVariable
            r2ml:name="order"
            r2ml:classID="srv:Order"/>
   </r2ml:contextArgument>
   <r2ml:TypedLiteral
            r2ml:lexicalValue="10"
            r2ml:typeLiteral="xs:positiveInteger"/>
  </r2ml:AssignActionExpression>
 </r2ml:producedAction>
</r2ml:ProductionRule>
```


Acknowledgment

This research has 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`).

10. CONCLUSION

In this paper, we have presented a usable interchange format that integrates the rule markup languages RuleML and SWRL with OCL.

The rich syntax of our interchange format allows to map many language constructs directly without translating them to different kinds of expressions. This increases the usability of the interchange format and also allows loss-free interchange in many cases.

11. REFERENCES

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

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

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

[4] Wagner, G., Antoniou, G., Tabet, S., and Boley, H.,The Abstract Syntax of RuleML - Towards a General Web Rule Language Framework, Rule Markup Initiative (RuleML), http://www.ruleml.org

[5] Wagner, G., Web Rules Need Two Kind of Negations, in Proc. of Principles and Practice of Semantic Web Reasoning, PPSWR 2003, pp.33-50.

[6] DOM Model, W3C Recommendation, http://www.w3.org/DOM/

[7] Jess, Sandia Lab., http://herzberg.ca.sandia.gov/jess/

[8] Model Driven Architecture (MDA), OMG, http://www.omg.org/cgi-bin/doc?mda-guide

[9] MS OutLook, Microsoft Corp., http://www.microsoft.com

[10] Object Constraint Language (OCL), v2.0, http://www.omg.org/docs/ptc/03-10-14.pdf

[11] Object Management Group (OMG), http://www.omg.org

[12] Oracle Views, Oracle Corp., http://oracle.com

[13] Resource Description Framework (RDF), W3 Recommendation, http://www.w3.org/RDF/

[14] Semantic Web Rule Language (SWRL), http://www.daml.org/swrl

[15] Standard Query Language (SQL1999),

[16] Web Ontology Language (OWL), W3 Recommendation, http://www.w3.org/2004/OWL/

[17] XML Metadata Interchange (XMI), http://www.omg.org/technology/documents/formal/xmi.htm

[18] XSB, http://xsb.sourceforge.net/

[19] W3C Workgroup on RIF Charter, http://www.w3.org/2005/rules/wg/charter