A General Markup Framework for Integrity and Derivation Rules

Gerd Wagner, Adrian Giurca, Sergey Lukichev
Brandenburg University of Technology at Cottbus,
{G.Wagner, Giurca, Lukichev}@tu-cottbus.de

This paper discusses the design of integrity and derivation rules on the basis of Rule Markup Language (RuleML) and Semantic Web Rule Language (SWRL). We propose a general markup framework for integrity and derivation rules (R2ML). Rule concepts are defined 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. From these MOF/UML language models we can obtain concrete markup syntax by applying a mapping procedure for generating corresponding languages from parameterized DTDs. **Keywords:***rule markup languages, integrity rules derivation rules, rule meta-models*

1 Introduction

Rule markup languages will be the vehicle for using rules on the Web and in distributed systems. They allow deploying, executing, publishing and communicating rules on the Web. They may also play the role of a lingua franca for exchanging rules between different systems and tools. It may be used: to express derivation rules for enriching Web ontologies by adding definitions of derived concepts or for defining data access permissions; to describe and publish the reactive behavior of a system in the form of reaction rules; and to provide a complete XML-based specification of a software agent. 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

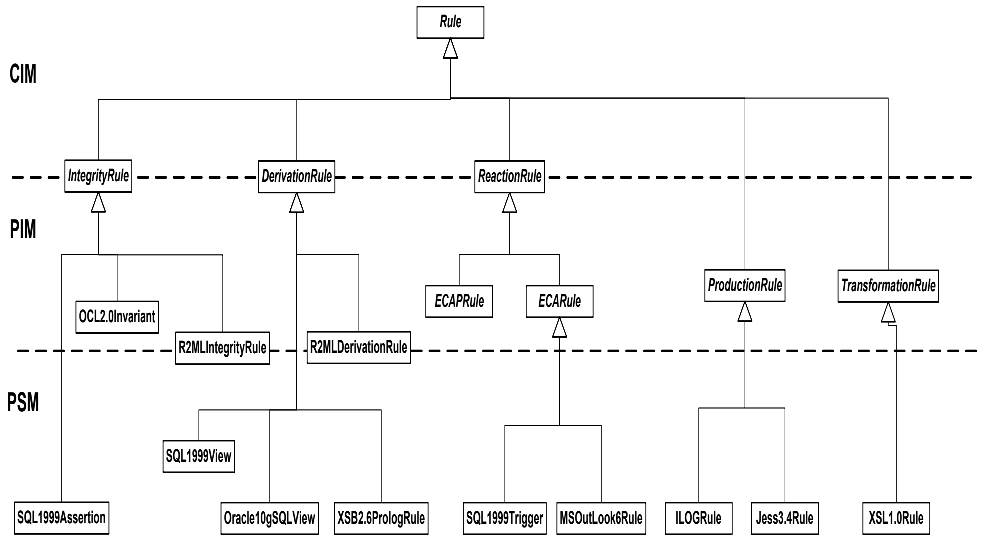

Figure 1: Rule concepts at three different abstraction levels: computation-independent (CIM), platform-independent (PIM) and platform-specific (PSM) modeling. Notice that integrity, derivation and reaction rules are both CIM and PIM concepts.

defining various concrete languages serving different purposes. The main purposes of a rule markup language is to permit reuse, interchange and publication of rules. Our goal is to define a family of rule languages capturing the most important types of rules. In this paper we start with integrity and derivation rules. While these languages should come with a recommended standard semantics, their rule expressions may, in addition, allow alternate semantics, which are also considered acceptable. This will accommodate various formalisms based on non-standard logics, supporting temporal, fuzzy, defeasible, and other forms of reasoning.
Our development approach is Model Driven Architecture (MDA,[9]) which is a framework for model-driven software development defined by the OMG, [12].

As illustrated in Figure 1, we consider rules at three different abstraction levels:
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 [16], OCL 2.0 [11] and DOM Level 3 Event Listeners [7]. Remarkably, SQL provides operational constructs for all three business rule categories mentioned above: checks/assertions operationalize a notion of integrity rules, views operationalize a notion of derivation rules, and triggers operationalize a notion of reaction rules.
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 [13], Jess 3.4 [8], XSB 2.6 Prolog [19] or the Microsoft Outlook 6 Rule Wizard [10].
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 [14] and OWL [17]. This requires that they accommodate:

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

SWRL [15] and the RuleML, [5] do not consider functions in atom construction. For simplicity, in R2ML we follow the same approach which corresponds of NafNegDatalog dialect. As a working name for our markup framework, we use the acronym R2ML standing for REWERSE Rule Markup Language.

This paper is focused on the development of suitable MOF/UML models for integrity and derivation rules, provides parameterized DTD's for the language constructs and gives rule examples, which illustrate the usage of the framework in modeling different kind of business rules.

2 The Rule Model

In this section, using the MOF/UML metamodeling, we define the vocabulary and all abstract constructs of R2ML framework . Such a metamodel can be transformed into a DTD and/or XML Schema definition by following a certain mapping procedure, like the one recommended by the XMI specification of the OMG. In this paper we present parameterized DTD's for all concepts depicted in the MOF/UML model of our framework. The main steps for obtaining a such DTD are:

1. mapping the segmentation defined by an abstract class to a corresponding DTD choice assigned to a parameter entity.
2. mapping each non-abstract class to a corresponding XML element by
   a) mapping the attributes of the class to corresponding attributes of the element, taking into consideration if they are mandatory or optional
   b) mapping the parts of the class to corresponding subelements, taking the multiplicity constraints of the aggregation into consideration (e.g. 1..* maps to +)

Logical formulas, logical statements and rules are expressed in various ways on the basis of a vocabulary, which includes

- definitions of proper names or globally unique object identifiers (such as URI references) standing for objects (or individuals);
- definitions of terms standing for general concepts, among which we distinguish entity types (classes) and datatypes;
- definitions of fact type expressions (or predicate symbols) standing for fact types (or predicates and properties);
- fact statements (expressing facts) instantiating fact type expressions, within the definition of individuals

In Web languages such as RDF and OWL, all these language elements do not have ordinary names. Instead, they have globally unique standard identifiers in the form of URI references.

2.1 Vocabulary Constructs

The R2ML framework has its own basic vocabulary which is defined to permit the basic constructs for rules:

- Vocabulary for classification: *Class*, *RDF:DataType*;
- Vocabulary for construction of data and object terms: *DataVariable*, *DataValue*, *OperationTerm*, *AttributeFunctionTerm*, *BuiltinFunctionTerm*, *ObjectConstant*, *RoleFunctionTerm*;
- Vocabulary for construction of atoms: *AssociationPredicate*, *DataPredicate*, *Attribute* and *ReferenceProperty.*

One of the goals of R2ML is to be compliant with the significant semantic web standards like RDF and OWL. R2ML accommodate data types from RDF and individuals from OWL. Below is the DTD for R2ML vocabulary:

```
<!-- vocabulary concepts used in terms-->
<!ELEMENT %Class; EMPTY>
<!ATTLIST %Class; %ID; (CDATA) #REQUIRED>

<!ELEMENT %RDFDataType; EMPTY>
<!ATTLIST %RDFDataType; %ID; (CDATA) #REQUIRED>

<!-- vocabulary concepts used in atoms-->
<!ELEMENT %AssociationPredicate; EMPTY>
<!ATTLIST %AssociationPredicate; %ID; (CDATA) #REQUIRED>

<!ELEMENT %DataPredicate; EMPTY>
<!ATTLIST %DataPredicate; %ID; (CDATA) #REQUIRED>

<!ELEMENT %Attribute; EMPTY>
<!ATTLIST %Attribute; %ID; (CDATA) #REQUIRED>

<!ELEMENT %ReferenceProperty; EMPTY>
<!ATTLIST %ReferenceProperty; %ID; (CDATA) #REQUIRED>
```

2.2 Objects, Data, Variables

The abstract concepts of *ObjectTerm* and *DataTerm* are depicted on the Figure 3 and 4.

Definition 1 (ObjectTerm, Figure 3) *The concept of* ObjectTerm *is used for modeling variables, that can be instantiated by object values and object constants. RoleFunctionTerm is an object term that is used to model an association end. For example, on the Figure 2, in the association* isMarriedTo *a* Person *plays a role of* wife *or* husband *and they are modeled as role function terms.*

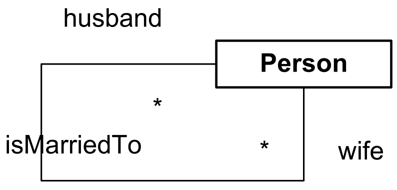

Figure 2: Role function terms `wife` and `husband`

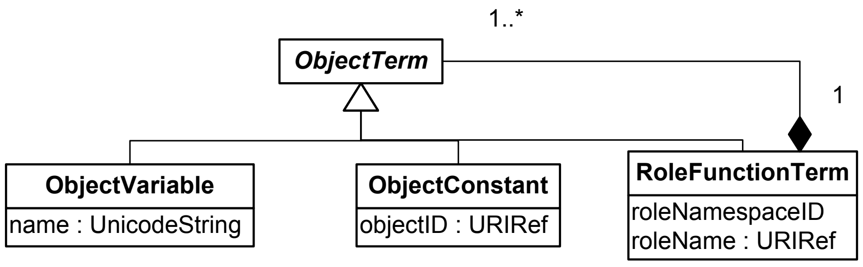

Figure 3: Object terms

Definition 2 (DataTerm, Figure 4) *DataTerm is used to represents primitive data types and data values. There are three types of data terms:* DataVariable*, which represents a variable,* DataValue*, which represents a value and* DataFunctionTerm. DataFunctionTerm *can be of three different types:*

1. BultinFunctionTerm *represents arithmetic built-ins.*
2. AttributeFunctionTerm *represents an attribute function (a function, which returns attribute value for an object).*

3. OperationTerm *represents user-defined function (method, for instance) and takes* DataTerm *or* ObjectTerm *as parameters.*

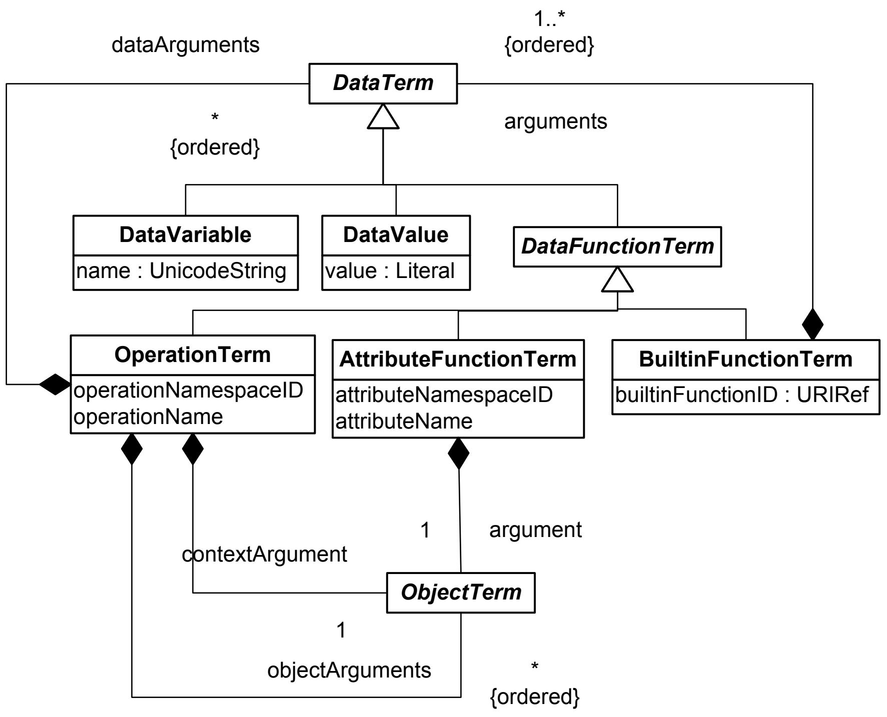

Figure 4: Data terms

Definition 3 (VariableDeclaration, Figure 5) *A variable declaration is a pair of a type and a set of variables, associated with this type.*

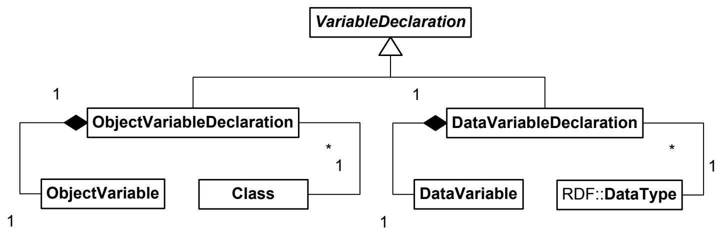

Figure 5: R2ML Variable Declaration

2.2.1 DTD module for objects terms, data terms

Below is the R2ML DTD module for objects terms, data terms and variables. ObjectTerm and DataTerm are abstract concepts and they are not used in the concrete markup of rules. In a concrete rule we use the derived concrete concepts i.e. ObjectVariable, ObjectConstant, DataVariable and RDFLiteral. The same remark holds also for the abstract concept of VariableDeclaration.

```
<!ELEMENT %ObjectTerm; (%ObjectVariable;|%ObjectConstant;
                      |%RoleFunctionTerm;)>

<!ELEMENT %ObjectVariable; EMPTY>
<!ATTLIST %ObjectVariable; %name; (CDATA) #REQUIRED>

<!ELEMENT %ObjectConstant; EMPTY>
<!ATTLIST %ObjectConstant; %ID; (CDATA) #REQUIRED>

<!ELEMENT %RoleFunctionTerm; (%objectArgument;)>
<!ATTLIST %RoleFunctionTerm; %namespaceID; (CDATA) #REQUIRED>
<!ATTLIST %RoleFunctionTerm; %name; (CDATA) #REQUIRED>

<!ELEMENT %DataTerm; (%DataVariable;|%DataValue;
             |%OperationTerm; |%AttributeFunctionTerm;
             |%BuiltinFunctionTerm;)>

<!ELEMENT %DataVariable; EMPTY>
<!ATTLIST %DataVariable; %name; (CDATA) #REQUIRED>

<!ELEMENT %DataValue; EMPTY>
<!ATTLIST %DataValue; %value; (CDATA) #REQUIRED>

<!ELEMENT %OperationTerm; (contextArgument,
                          objectArgument, dataArgument*)>
<!ATTLIST %OperationTerm; %namespaceID; (CDATA) #REQUIRED>
<!ATTLIST %OperationTerm; %name; (CDATA) #REQUIRED>

<!ELEMENT %AttributeFunctionTerm; (objectArgument)>
```

```
<!ATTLIST %AttributeFunctionTerm; %namespaceID; (CDATA) #REQUIRED>
<!ATTLIST %AttributeFunctionTerm; %name; (CDATA) #REQUIRED>

<!ELEMENT %BuiltinFunctionTerm; (dataArgument, dataArgument*)>
<!ATTLIST %BuiltinFunctionTerm; %ID; (CDATA) #REQUIRED>

<!ELEMENT %VariableDeclaration; (%ObjectVariableDeclaration;|
                                 %DataVariableDeclaration;)>

<!ELEMENT %ObjectVariableDeclaration; (%ObjectVariable;)>
<!ATTLIST %ObjectVariableDeclaration; %classRef; (CDATA) #REQUIRED>

<!ELEMENT %DataVariableDeclaration; (%DataVariable;)>
<!ATTLIST %DataVariableDeclaration; %RDFDataTypeRef; (CDATA) #REQUIRED>

<!ELEMENT objectArgument (%ObjectTerm;)>
<!ELEMENT dataArgument (%DataTerm;)>
<!ELEMENT contextArgument (%ObjectTerm;)>
```

2.3 Atoms

The basic constituent of a rule is the *atom*. In R2ML we define metamodels for atoms, which are compatible with all important concepts of OWL, SWRL and RuleML. All atoms from our framework are presented on the Figure 6.

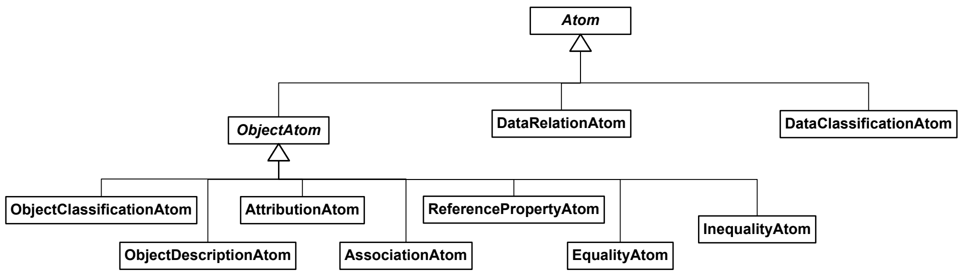

Figure 6: R2ML Atoms

Definition 4 (Object Classification Atom, Figure 7) *An* object classification atom *consists of a class type (as "base type") and an object term, variable, constant or function term.*

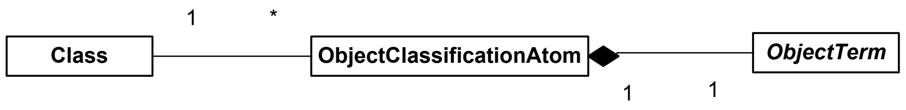

Figure 7: R2ML Object Classification Atom

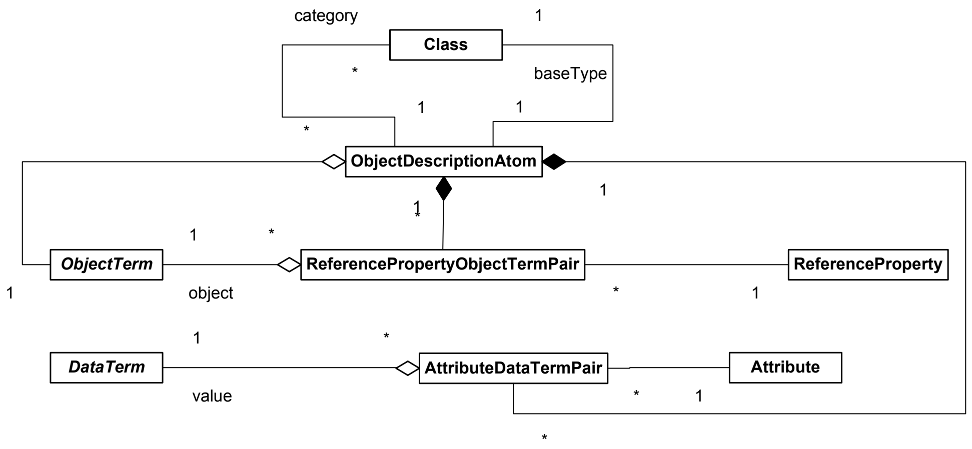

Figure 8: R2ML Object Description Atom is a shorthand from providing a set of descriptions for a particular object.

Following RDF [14] and OWL [17], we adopt the concept of *object description atom.*

Definition 5 (ObjectDescriptionAtom, Figure 8) *An* ObjectDescriptionAtom *is a collection of classes (categories and a base type), 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.*

Definition 6 (AttributionAtom, Figure 9) *An* attribution atom *in R2ML consists of a variable, constant or function (object terms) as "subject", and a data variable or an RDF literal (*data term*) as "value".*

Definition 7 (ReferencePropertyAtom, Figure 10) *A* reference property atom *associates variables or constants (*object terms*) as "subjects" with other object terms as "objects".*

Notice that, in the OWL approach, property axiom is mixed up with constraint assertions about it (for example, a property to be functional) but in the R2ML approach the definitions are separated from constraints. In order to support common fact types of natural language directly, it is important to have n-ary predicates (for $n > 2$).

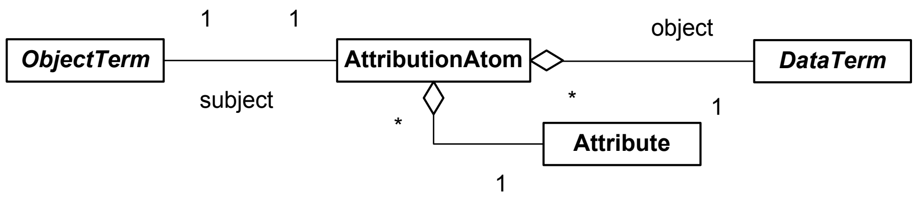

Figure 9: R2ML Attribution Atom

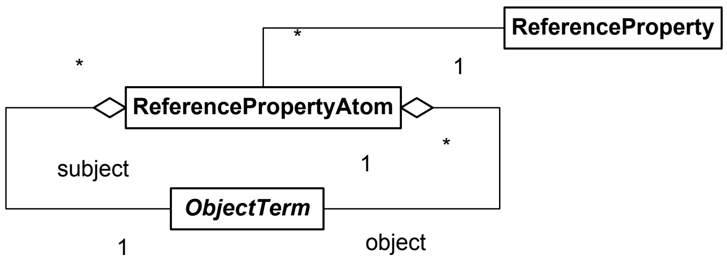

Figure 10: R2ML Reference Property Atom

Definition 8 (AssociationAtom) Association atom *is constructed using one n-ary predicate as association predicate, a collection of data terms as "data argument" and a collection of object terms as "object argument". The model is depicted on Figure 11.*

As in OWL, *equality* and *inequality* atoms must be defined.

Definition 9 (EqualityAtom, InequalityAtom, Figure 12) *An EqualityAtom (InequalityAtom) is a composition of two or more object terms.*

Definition 10 (DataClassificationAtom, Figure 13) *A* data classification atom *consists of an RDF data type and a data term, which is a data variable or an RDF literal.*

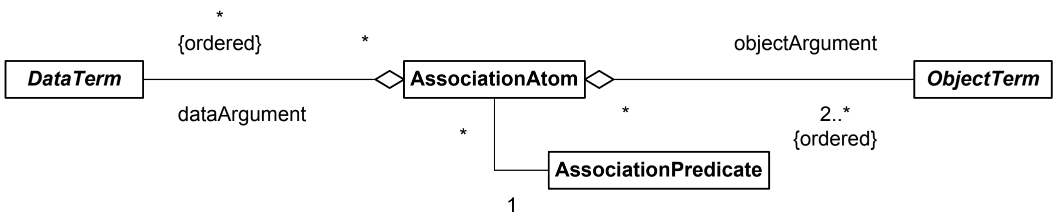

Figure 11: R2ML Association Atom can express an n-ary association between classes.

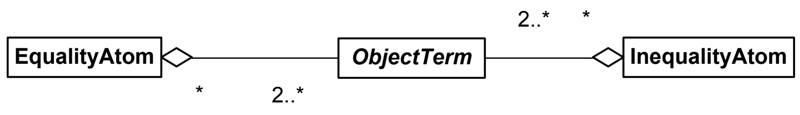

Figure 12: R2ML Equality and Inequality Atoms

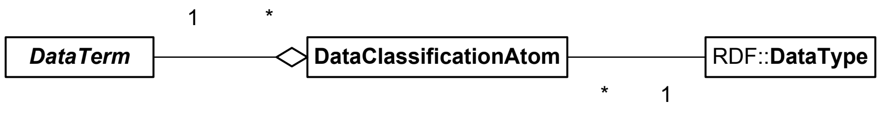

Figure 13: R2ML Data Classification Atom

Definition 11 (DataRelationAtom, Figure 14) *A* data relation atom *describes a relation between several data terms, using a data predicate.*

2.3.1 The DTD for Atoms.

Atom and *ObjectAtom* are abstract concepts and they are not used in a markup. For a markup we use derived concrete concepts, for example, *EqualityAtom*, *AssociationAtom* and so on. We keep all the abstract concepts in the DTD in order to use them in a future general XML Schema of the framework.

```
<!-- abstract atoms -->
<!ELEMENT %Atom (%ObjectAtom;|%DataRelationAtom;|
                %DataClassificationAtom;)>
<!ELEMENT %ObjectAtom; (%AssociationAtom;|%ObjectDescriptionAtom;|
            %ObjectClassificationAtom;|%AttributionAtom;|
            %ReferencePropertyAtom;|%EqualityAtom;|%InequalityAtom;)>
<!ELEMENT %AssociationAtom; (%DataTerm;*,%ObjectTerm;,%ObjectTerm;+)>
<!ATTLIST %AssociationAtom; %associationPredicateRef; (CDATA) #REQUIRED>
<!ELEMENT %ObjectDescriptionAtom; (%ReferencePropertyObjectTermPair;*,
                                %AttributeDataTermPair;*)>
```

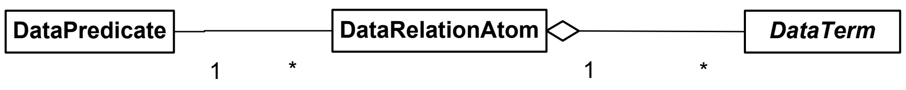

Figure 14: R2ML Data Relation Atom

```
<!ATTLIST %ObjectDescriptionAtom; category (CDATA) #IMPLIED>
<!ATTLIST %ObjectDescriptionAtom; baseType (CDATA) #REQIURED>
<!ELEMENT %ReferencePropertyObjectTermPair; (object)>
<!ATTLIST %ReferencePropertyObjectTermPair;
          %referencePropertyRef; (CDATA) #REQUIRED>
<!ELEMENT object (%ObjectTerm;*)>
<!ELEMENT %AttributeDataTermPair; (value)>
<!ATTLIST %AttributeDataTermPair; %attributeRef; (CDATA) #REQUIRED>
<!ELEMENT value (%DataTerm;*)>
<!ELEMENT %ObjectClassificationAtom; (%ObjectTerm;)>
<!ATTLIST %ObjectClassificationAtom; %classRef; (CDATA) #REQUIRED>
<!ELEMENT %AttributionAtom; (subject, value)>
<!ATTLIST %AttributionAtom; %attributeRef; (CDATA) #REQUIRED>
<!ELEMENT subject (%ObjectTerm;)>
<!ELEMENT %ReferencePropertyAtom; (subject, object)>
<!ATTLIST %ReferencePropertyAtom;
          %referencePropertyRef; (CDATA) #REQUIRED>
<!ELEMENT %DataClassificationAtom; (%DataTerm;)>
<!ATTLIST %DataClassificationAtom;
          %RDFdataTypeRef; (CDATA) #REQUIRED>
<!ELEMENT %DataRelationAtom; (%DataTerm;*)>
<!ATTLIST %DataRelationAtom; %dataPredicateRef; (CDATA) #REQUIRED>
<!ELEMENT %EqualityAtom; (%ObjectTerm;, %ObjectTerm;+)>
<!ELEMENT %InequalityAtom;(%ObjectTerm;, %ObjectTerm;+)>
```

2.4 Formulas

R2ML provides two abstract concepts for formulas: the concept of *AndOrNafNegFormula* (see Figure 15), which represents the most general quantifier free logical formula with weak and strong negations, and the concept of *LogicalFormula* (see Figure 16), which corresponds to a general first order formula.

2.4.1 R2ML uses two kinds of negations.

The distinction between weak and strong negation (marked up as `<naf>` and `<neg>` in RuleML and also in our markup proposal from the Section 4) is used in several computational languages: it is presented in explicit form in extended logic programs [3],

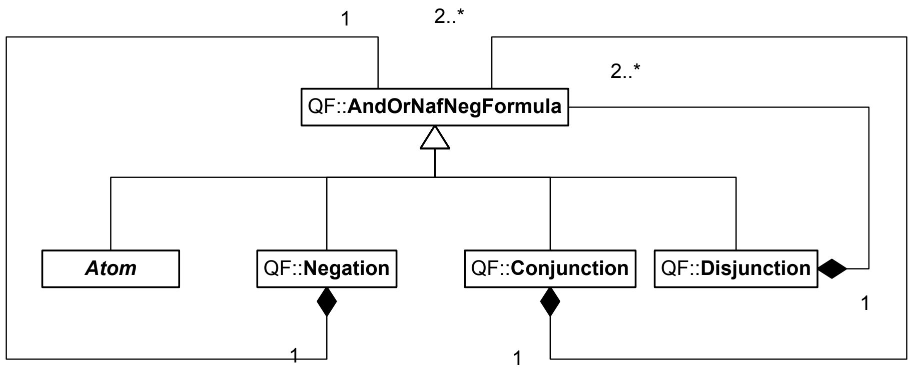

Figure 15: R2ML AndOrNafNegFormula

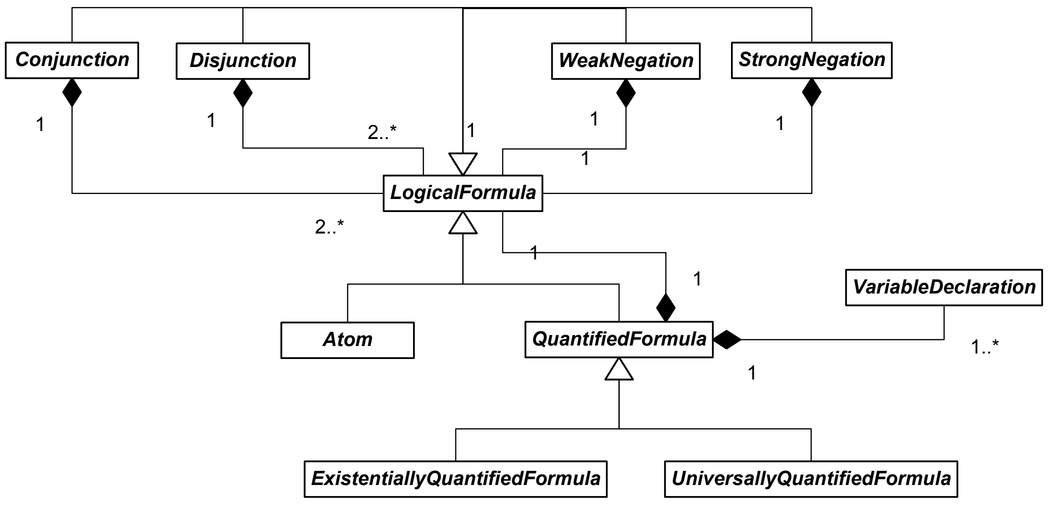

Figure 16: R2ML Logical Formula

only implicitly in SQL [16] and OCL [11], as was shown in [6]. Intuitively speaking, weak negation captures the absence of positive information, while strong negation captures the presence of explicit negative information (in the sense of Kleene's 3-valued logic). Under the minimal/stable models [2] weak negation captures the computational concept of negation-as-failure (or closed-world negation) [1]. The model is depicted on Figure 17.

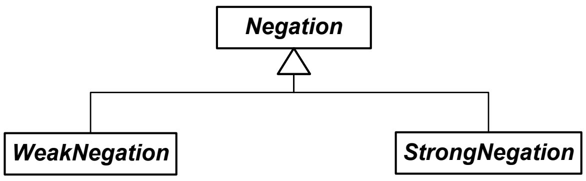

Figure 17: R2ML Negations

2.4.2 The DTD for formulas.

```
<!-- abstract formulas -->
<!ELEMENT %NegObjectFormula; (%ObjectAtom;|%NegObjectAtom;)>
<!ELEMENT %NegObjectAtom; (%ObjectAtom;)>

<!ELEMENT %AndOrNafNegFormula; (%Atom;?|
                           %QF-Conjunction;?|%QF-Disjunction;?|
                           %QF-WeakNegation;?|%QF-StrongNegation;?)>

<!ELEMENT %QF-StrongNegation; (%AndOrNafNegFormula;)>
<!ELEMENT %QF-WeakNegation; (%AndOrNafNegFormula;)>
<!ELEMENT %QF-Conjunction; (%AndOrNafNegFormula;,%AndOrNafNegFormula;+)>
<!ELEMENT %QF-Disjunction; (%AndOrNafNegFormula;,%AndOrNafNegFormula;+)>
```

2.5 Rules Constructs

2.5.1 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 18). 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

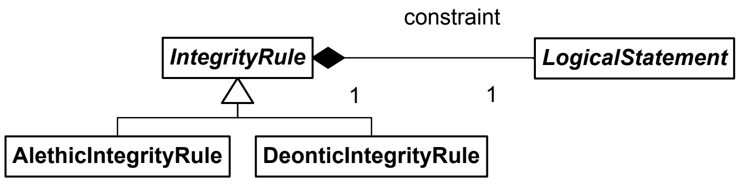

Figure 18: Integrity Rules in R2ML

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. The R2ML DTD for integrity rules is simple:

```
<!-- R2ML Integrity Rule -->
<!ELEMENT %AlethicIntegrityRule; (constraint)>
<!ELEMENT %DeonticIntegrityRule; (constraint)>
<!ELEMENT constraint (%LogicalStatement;)>
```

2.5.2 Derivation Rules

Derivation Rules, have "conditions" and "conclusions" (see Figure 19). In R2ML framework the conditions of a derivation rule are *AndOrNafNegFormula*, as defined on Figure 15. Conclusions are restricted to object atoms or strongly negated object atoms (*NegObjectAtom*), as depicted on Figure 20.

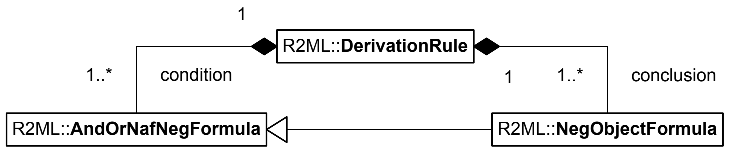

Figure 19: The R2ML Model of Derivation Rules

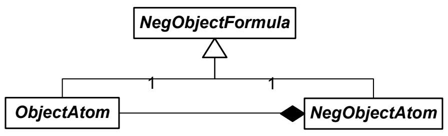

Figure 20: R2ML NegObjectFormula

Table 1: Basic language constructs for vocabularies

R2ML	RuleML	SWRL
data variable	variable	data variable
object variable	variable	individual variable
data term	term	data term
object term	term	object term
association predicate	n/a	n/a
class	n/a	OWL:Description
attribute	n/a	iObject
data predicate	rel	n/a

Table 2: Accommodating different atoms concepts

R2ML	RuleML	SWRL
ObjectDescriptionAtom	SlotAtom	n/a
ObjectClassificationAtom	PositionalAtom	classAtom
AssociationAtom	PositionalAtom	n/a
AttributionAtom	PositionalAtom	datavaluedPropertyAtom
ReferencePropertyAtom	PositionalAtom	individualPropertyAtom
DataRelationAtom	n/a	builtinAtom
DataClassificationAtom	n/a	datarangeAtom
EqualityAtom	n/a	sameIndividualAtom
InequalityAtom	n/a	differentIndividualsAtom

2.5.3 DTD for derivation rules

Below we present the R2ML DTD for derivation rules:

```
<!ELEMENT %DerivationRule (condition, conclusion)>
<!ELEMENT condition (%LogicalFormula+)>
<!ELEMENT conclusion (%LogicalFormula+)>
```

3 SWRL, RuleML and R2ML

In R2ML we accommodate all important language constructs from SWRL and RuleML. Because of space limitation we present the terminological correspondence between these languages and R2ML in Tables 1 and 2.

4 An R2ML Concrete Language Example

For the concrete definition of some abstract concepts that are needed for a derived rule markup language, we use short names for the elements and the same names for abstract types. We apply this rule for every abstract concept from the MOF/UML model that need a derived concrete concept. The markup of rules from our examples (Example 1 and Example 2) is based on the following terminology proposal:

```
<!--
We keep the name for every abstract concept because
these names will not be used in concrete instances of the rules.
The concrete rules will use short names for convenience.
-->
<!ENTITY % SWRL_PREFIX "swrlx">
<!ENTITY % RDF_PREFIX "rdf">
<!ENTITY % RuleML_PREFIX "ruleml">
<!ENTITY % COLON ":">
<!ENTITY % SWRL_NAMESPACE_DECLARATION "xmlns%COLON;%SWRL_PREFIX;">
<!ENTITY % RDF_NAMESPACE_DECLARATION "xmlns%COLON;%RDF_PREFIX;">
<!ENTITY % RuleML_NAMESPACE_DECLARATION "xmlns%COLON;%RuleML_PREFIX;">
<!ENTITY % R2ML_NAMESPACE_DECLARATION "xmlns%COLON;">

<!-- datatypes and literals from RDF -->
<!ENTITY % RDFDataType "%RDFPREFIX;%COLON;datatype">
<!ENTITY % RDFLiteral "%RDFPREFIX;%COLON;literal">

<!-- vocabulary names -->
<!ENTITY % Class "Class">
<!ENTITY % ID "id">
<!ENTITY % AssociationPredicate "AssocPred">
<!ENTITY % DataPredicate "DataPred">
<!ENTITY % Attribute "Attr">
<!ENTITY % ReferenceProperty "RefProp">

<!-- objects, data and variables -->
<!ENTITY % ObjectTerm "ObjectTerm">
<!ENTITY % ObjectVariable "OVar">
```

```
<!ENTITY % ObjectConstant "OConst">
<!ENTITY % RoleFunctionTerm "Role">

<!ENTITY % DataTerm "DataTerm">
<!ENTITY % DataVariable "DVar">
<!ENTITY % DataValue "DataValue">
<!ENTITY % OperationTerm "Operation">
<!ENTITY % AttributeFunctionTerm "Attribute">
<!ENTITY % BuiltinFunctionTerm "BuiltinFunction">

<!ENTITY % contextArgument "contextArgument">
<!ENTITY % objectArgument "objectArgument">
<!ENTITY % dataArgument "dataArgument">

<!ENTITY % VariableDeclaration "VariableDeclaration">
<!ENTITY % ObjectVariableDeclaration "OVarDecl">
<!ENTITY % DataVariableDeclaration "DVarDecl">

<!ENTITY % value "value">
<!ENTITY % namespaceID "namespaceID">
<!ENTITY % name "name">

<!-- atoms -->
<!ENTITY % Atom "Atom">
<!ENTITY % ObjectAtom "ObjectAtom">
<!ENTITY % AssociationAtom "AssocAtom">
<!ENTITY % associationPredicateRef "assocPredRef">
<!ENTITY % ObjectDescriptionAtom "DescrAtom">
<!ENTITY % ReferencePropertyObjectTermPair "PropOTermPair">
<!ENTITY % referencePropertyRef "propRef">
<!ENTITY % AttributeDataTermPair "AttrDTermPair">
<!ENTITY % attributeRef "attrRef">
<!ENTITY % ObjectClassificationAtom "OClassAtom">
<!ENTITY % classRef "classRef">
<!ENTITY % AttributionAtom "AttrAtom">
<!ENTITY % ReferencePropertyAtom "PropAtom">
```

```
<!ENTITY % DataClassificationAtom "DClassAtom">
<!ENTITY % RDFdataTypeRef "rdf:resource">
<!ENTITY % DataRelationAtom "RelAtom">
<!ENTITY % dataPredicateRef "relPredRef">
<!ENTITY % EqualityAtom "Equal">
<!ENTITY % InequalityAtom "Different">

<!-- formulas -->
<!ENTITY % LogicalStatement "Filter">
<!ENTITY % AndOrNafNegFormula "AndOrNafNegFormula">
<!ENTITY % NegObjectFormula "NegObjectFormula">
<!ENTITY % NegObjectAtom "Neg">
<!ENTITY % QF-WeakNegation "Naf">
<!ENTITY % QF-StrongNegation "Neg">
<!ENTITY % QF-Conjunction "And">
<!ENTITY % QF-Disjunction "Or">


<!-- rules -->
<!ENTITY % DerivationRule "DerivationRule">
<!ENTITY % AlethicIntegrityRule "AlethicIntegrityRule">
<!ENTITY % DeonticIntegrityRule "DeonticIntegrityRule">
```

Example 1 *The integrity rule example from Figure 21 is based on the case study of a fictitious car rental company called EU-Rent.*

```
<!-- Integrity Rule in a language based on R2ML -->
<AlethicIntegrityRule xmlns:srv="http://www.services.org/EU-Rent/">
 <constraint>
  <OClassAtom classRef="srv:Rental">
   <OVar name="r1"/>
  </OClassAtom>
  <PropAtom propRef="srv:returnBranch">
   <subject><OVar name="r1"/></subject>
   <object><OVar name="rb"/></object>
  </PropAtom>
  <PropAtom propRef="srv:pickupBranch">
```

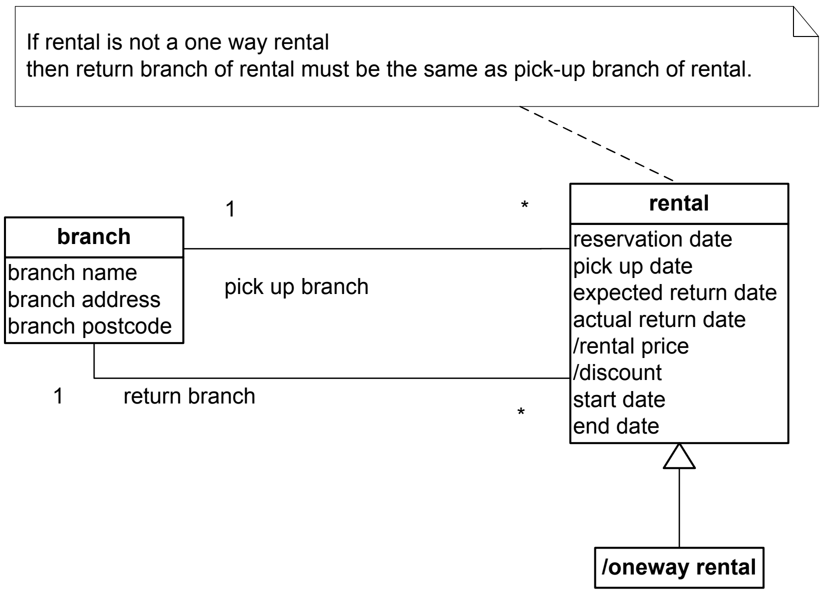

Figure 21: If rental is not a one way rental then return branch of rental must be the same as pick-up branch of rental.

```
   <subject><OVar name="r1"/></subject>
   <object><OVar name="pb"/></object>
  </PropAtom>
  <Neg>
   <OClassAtom classRef="srv:OneWayRental">
    <OVar name="r1"/>
   </OClassAtom>
  </Neg>
  <Equal>
   <OVar name="rb"/>
   <OVar name="pb"/>
  </Equal>
 </constraint>
</AlethicIntegrityRule>
```

The following derivation rule example is based on the EU-Rent case study as well.

Example 2 *The first condition of this rule "no rental associated with the rental car", corresponds to a negation-as-failure, which is expressed by the tag* `<Naf>`*. The second condition,* `not scheduled for service` *is a categorization and corresponds to a strong*

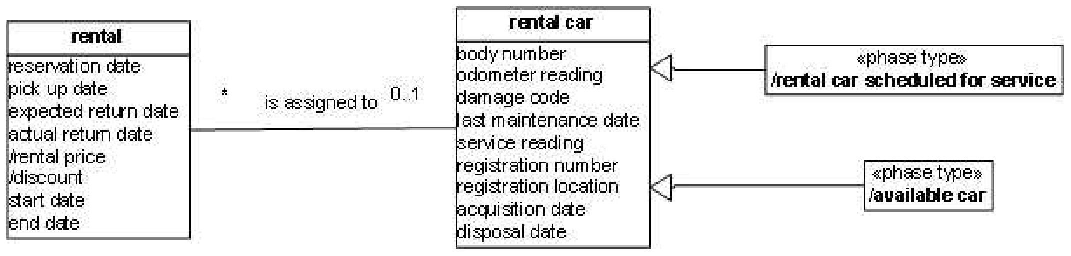

Figure 22: A car is available for rental if it is not assigned to any rental and is not scheduled for service.

negation, because it requires the value to be explicitly false.

```
<!-- Derivation Rule in a language based on R2ML -->
<DerivationRule xmlns:srv="http://www.services.org/EU-Rent/">
<condition>
 <OClassAtom classRef="srv:RentalCar">
  <OVar name="c1"/>
 </OClassAtom>
 <OClassAtom classRef="srv:RentalContract">
  <OVar name="rc"/>
 </OClassAtom>
 <Naf>
  <PropAtom propRef="srv:isAssignedTo">
   <subject><OVar name="c1"/></subject>
   <object><OVar name="rc"/></object>
  </PropAtom>
 </Naf>
 <Neg>
 <OClassAtom classRef="srv:rentalCarScheduledForService">
  <OVar name="c1"/>
 </OClassAtom>
 </Neg>
</condition>
<conclusion>
 <OClassAtom classRef="srv:isAvailable">
  <OVar name="r1"/>
 </OClassAtom>
```

```
</conclusion>
</DerivationRule>
```

5 Conclusion and Future work

In this paper, we have presented R2ML, a framework for rule markup language design of integrity and derivation rules. The modeling is based on the method of MOF/UML meta-modeling for defining the abstract syntax of languages. We have constructed all fundamental concepts needed for modeling of integrity and derivation rules in semantic web and provided parameterized DTD definitions for each concept. Section 3 is devoted to the position of the R2ML framework and SWRL/RuleML approaches. As a conclusion we accommodate all constructs from SWRL and RuleML. The rule examples from Section 4 demonstrate that R2ML is a powerful framework and could represent various kinds of rules. This paper is a first step in R2ML framework development and immediate future work will be devoted to: *developing a mechanism for automatically schema generation from visual models*, *providing an abstract syntax and an abstract semantics* and *developing use-cases for rules in different areas of information systems.*

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., How to Design a General Rule Markup Language, Proceedings of the Workshop XML Technologies for the Semantic Web (XSW 2002), HU Berlin, Insti tut fur Informatik, June 2002, Lecture Notes in Informatics, Gesellschaft f. Informatik.

[5] 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

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

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

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

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

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

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

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

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

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

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

[16] Standard Query Language (SQL1999),

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

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

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