I1-D6

Verbalization of the REWERSE I1 Rule Markup Language

Project title:	Reasoning on the Web with Rules and Semantics
Project acronym:	REWERSE
Project number:	IST-2004-506779
Project instrument:	EU FP6 Network of Excellence (NoE)
Project thematic priority:	Priority 2: Information Society Technologies (IST)
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	IST506779/Cottbus/I1-D6/D/PU/b1
Responsible editors:	Sergey Lukichev
Reviewers:	Norbert E. Fuchs (internal) Maria Keet (external)
Contributing participants:	Cottbus, Zurich
Contributing workpackages:	I1
Contractual date of deliverable:	September 3, 2006
Actual submission date:	September 7, 2006

Abstract
In this technical report we describe a verbalization component for verbalization of the rule markup language R2ML, developed by the REWERSE Working Group I1. The verbalization approach is based on templates in a form of XSL transformations for each type of R2ML rule.

Keyword List
Rules, Verbalization, R2ML, Rule Markup Language, Semantic Web

Project co-funded by the European Commission and the Swiss Federal Office for Education and Science within the Sixth Framework Programme.

Verbalization of the REWERSE I1 Rule Markup Language

Sergey Lukichev[1], **Gerd Wagner**[1]

[1] Institute of Informatics, Brandenburg University of Technology at Cottbus, Email: {G.Wagner, Lukichev}@tu-cottbus.de

September 7, 2006

Abstract
In this technical report we describe a verbalization component for verbalization of the rule markup language R2ML, developed by the REWERSE Working Group I1. The verbalization approach is based on templates in a form of XSL transformations for each type of R2ML rule.

Keyword List
Rules, Verbalization, R2ML, Rule Markup Language, Semantic Web

Contents

Chapter 1

Introduction

In this technical report we describe a verbalization component for verbalization of the rule markup language R2ML, developed by the REWERSE Working Group I1. The verbalization approach is based on templates in a form of XSL transformations for each type of R2ML rules. The implementation of the R2ML verbalizer consists of the following parts:

1. Transformation of R2ML rule bases into controlled natural language, where all vocabulary terms are taken from the R2ML rule base as URIs.

2. Resolving natural language representation of terms from the underlying rule base vocabulary.

The current verbalization method is inspired by the verbalization approach of ORM models ([4], [10], [5]). The ORM verbalization approach uses XML to represent the template structure for the pseudo-natural language sentences of each ORM constraint. Since our work is oriented towards the Semantic Web [2], we have to deal with Web artifacts like URIs and namespaces and use Web technologies, such as XSLT [3] and XPath [1] to represent templates. An XSL transformation is an XML document, which contains processing instructions and which we use to describe the structure for the pseudo-natural language sentences. The use of XSLT in our approach has an advantage against custom XML format. The advantage is a strong tools and libraries support for XSLT as well as a number of efficient parsers available.

The described verbalization is for two audiences: business people and technical engineers. Both groups need verbalization in order to validate the correctness of their rules. The verbalizer, described in this report, produces output in the controlled natural language, which is understandable by both communities, but sometimes may be too technical for business people. We discuss in the report when the output may be difficult to understand by business people and list further improvements to the verbalizer in the Chapter 5.

In Chapter 2 we describe verbalization of main rule constituents: terms and atoms. In Chapter 3 we describe XSLT templates for all kinds of R2ML rules. In Chapter 4 we briefly describe a vocabulary support in R2ML and provide a mechanism of rule terms resolution from the vocabulary by term URIs. In Chapter 5 we list further planned improvements to the described verbalizer.

For the purpose of better understanding of this report we recommend reading the REWERSE I1 Deliverable D8 "Language Improvements and Extensions", which contains full specification of R2ML, description of all supported terms, atoms and rules and examples[9].

Chapter 2

Verbalizing Terms and Atoms

In this chapter we describe verbalization of main rule constituents: terms and atoms.

2.1 Verbalizing Terms

Since terms are atoms constituents they are not verbalized directly to English sentences but to parts of a sentence. Good verbalization quality is obtained if roles (r2ml:contextArgument, r2ml:subject, r2ml:object) are simple terms: ObjectName, Variable, DataLiteral. The proper verbalization of rules, which contain roles with complex terms like ObjectOperationTerm of DataOperationTerm is a subject for immediate further improvements.

2.1.1 R2ML Variables, Object and Data Literals

The R2ML variables (r2ml:ObjectVariable, r2ml:GenricVariable, and r2ml:DataVariable) are mapped into strings using the value of r2ml:name attribute. An XSLT template:

```
<xsl:template match="r2ml:ObjectVariable | r2ml:GenericVariable| r2ml:DataVariable">
 <xsl:value-of select="replace(string(@r2ml:name),':','#')"/>
</xsl:template>
```

The r2ml:ObjectVariable is mapped into a string constant by using the value of the r2ml:objectID attribute.

An XSLT template:

```
<xsl:template match="r2ml:ObjectVariable">
 <xsl:value-of select="replace(string(@r2ml:objectID),':','#')"/>
</xsl:template>
```

Both r2ml data literals i.e. r2ml:TypedLiteral and r2ml:PlainLiteral are mapped into the value of the r2ml:lexicalValue attribute.

An XSLT template:

```
<xsl:template match="r2ml:TypedLiteral">
 <xsl:value-of select="string(@r2ml:lexicalValue)"/>
</xsl:template>
```

```
<xsl:template match="r2ml:PlainLiteral">
 <xsl:value-of select="string(@r2ml:lexicalValue)"/>
</xsl:template>
```

2.1.2 R2ML Arguments

There are different R2ML arguments: r2ml:contextArgument, r2ml:arguments, r2ml:dataArguments, and r2ml:objectArguments. They are used in order to represent context argument and arguments of operations and functions. Context argument is used to specify a context of different terms. For instance, "r1.pickupBranch" is the ReferencePropertyFunctionTerm, where "r1" is an object variable of class Rental and "pickupBranch" is a reference property. The class identifier of a reference property function term is optional in R2ML. If it is defined, the above example is verbalized as "Rental r1" and just "r1" otherwise.

There are two solutions to the verbalization of r2ml:arguments, r2ml:dataArguments and r2ml:objectArguments:

1. Use general templates, which are defined for each type of R2ML terms and actions (rule set level). For instance, an operation with two arguments "sendMessage(sender, receiver)", can be verbalized as "invoke send message with arguments sender as argument 1 and receiver as argument 2". This is so called technical style, which is used by programmers for verbalizing technical specifications.

2. Define a template for each operation in the vocabulary as a special annotation property (vocabulary level). This allows verbalizing the above example as "send message from sender to receiver".

For business people the second solution is preferable, since it is more natural and easier to read. The current version of the verbalizer supports only the first solution and we plan to implement the second solution in the next version.

2.1.3 GenericFunctionTerm

The R2ML GenericFunctionTerm covers the standard concept of a first order logic functional term. It consists of a function ID reference and an ordered list of "arguments" that are terms. FunctionTerm is mapped into:

<genericFunctionID> with <arguments>

An XSLT template:

```
<xsl:template match="r2ml:FunctionTerm">
 <xsl:value-of select="replace(string(@r2ml:genericFunctionID),':','#')"/>
 <xsl:if test="count(r2ml:arguments/*)>0">
     <xsl:text> with </xsl:text>
     <xsl:apply-templates select="r2ml:arguments"/>
 </xsl:if>
</xsl:template>
```

For example, the following GenricFunctionTerm

```
<r2ml:GenericFunctionTerm r2ml:genericFunctionID="successor">
 <r2ml:arguments>
  <r2ml:GenericVariable r2ml:name="N"/>
 </r2ml:arguments>
</r2ml:FunctionTerm>
```

is verbalized as:

successor with N as argument 1

2.1.4 ReferencePropertyFunctionTerm

A ReferencePropertyFunctionTerm corresponds to a UML functional association end (of a binary association) and consists of an ObjectTerm as contextArgument and a reference to a reference property (via r2ml:referencePropertryID attribute). Since its intended meaning is to associate the reference property with the context argument, ReferencePropertyFunctionTerm is mapped into:

<referencePropertyID> of <contextArgument>

An XSLT template:

```
<xsl:template match="r2ml:ReferencePropertyFunctionTerm">
 <xsl:value-of select="replace(string(@r2ml:referencePropertyID),':','#')"/>
 <xsl:text> of </xsl:text>
 <xsl:apply-templates select="r2ml:contextArgument"/>
</xsl:template>
```

For example, the ReferencePropertyFunctionTerm

```
<r2ml:ReferencePropertyFunctionTerm
      r2ml:referencePropertyID="returnBranch">
 <r2ml:contextArgument>
  <r2ml:ObjectVariable r2ml:name="rental"/>
 </r2ml:contextArgument>
</r2ml:RoleFunctionTerm>
```

is verbalized as

returnBranch of rental

2.1.5 DatatypeFunctionTerm

DatatypeFunctionTerm is mapped into:

<datatypeFunctionID> with <dataArguments>

An XSLT template:

```
<xsl:template match="r2ml:DatatypeFunctionTerm">
 <xsl:value-of select="@r2ml:datatypeFunctionID"/>
 <xsl:if test="count(r2ml:dataArguments/*)>0">
   <xsl:text> with </xsl:text>
   <xsl:apply-templates select="r2ml:dataArguments"/>
 </xsl:if>
</xsl:template>
```

For example, the following DatatypeFunctionTerm

```
<r2ml:DatatypeFunctionTerm r2ml:datatypeFunctionID="orderValue">
 <r2ml:dataArguments>
   <r2ml:ObjectVariable r2ml:name="c1" r2ml:classID="Customer"/>
 </r2ml:dataArguments>
</r2ml:DatatypeFunctionTerm>
```

is verbalized as:

> orderValue with Customer c1 as argument 1

2.1.6 ObjectOperationTerm and DataOperationTerm

Both R2ML ObjectOperationTerm and DataOperationTerm model the concept of a UML operation. The distinction is made by the return type of the operation. As in UML, any operation has an ObjectTerm as a "contextArgument" and a list of terms as operation "arguments". ObjectOperationTerm and DataOperationTerm are mapped into:

> <operationID> of <contextArgument> with <arguments>

An XSLT template:

```
<xsl:template match="r2ml:ObjectOperationTerm|r2ml:DataOperationTerm">
 <xsl:value-of select="replace(string(@r2ml:operationID),':','#')"/>
 <xsl:if test="r2ml:contextArgument">
  <xsl:text> of </xsl:text>
  <xsl:apply-templates select="r2ml:contextArgument"/>
 </xsl:if>
 <xsl:if test="r2ml:arguments">
  <xsl:text> with </xsl:text>
  <xsl:apply-templates select="r2ml:arguments"/>
 </xsl:if>
</xsl:template>
```

For example, the DataOperationTerm

```
<r2ml:DataOperationTerm r2ml:operationID="spending">
 <r2ml:contextArgument>
  <r2ml:ObjectVariable r2ml:name="c1" r2ml:classID="Customer"/>
 </r2ml:contextArgument>
 <r2ml:arguments>
  <r2ml:DataVariable r2ml:name="previous_year" r2ml:datatypeID="xs:gYear"/>
```

```
 </r2ml:arguments>
</r2ml:DataOperationTerm>
```

is verbalized as

spending of Customer c1 with previous_year as argument 1

The described template is in technical style and for better readability by business people it needs to be improved with an operation verbalization template. Since future versions of R2ML will allow to define an operation in the vocabulary with possibility to annotate them and define flexible verbalization templates on the vocabulary level, the verbalization of the above example will look like:

customer's spending in previous_year

2.1.7 AttributeFunctionTerm

An R2ML AttributeFunctionTerm represents a value of an attribute in the context of an object. AttributeFunctionTerm is mapped into:

<attributeID> of <contextArgument>

An XSLT template:

```
<xsl:template match="r2ml:AttributeFunctionTerm">
 <xsl:value-of select="replace(string(@r2ml:attributeID),':','#')"/>
 <xsl:text> of </xsl:text>
 <xsl:apply-templates select="r2ml:contextArgument"/>
</xsl:template>
```

For example, the AttributeFunctionTerm

```
<r2ml:AttributeFunctionTerm r2ml:attributeID="last_maintenance_date">
 <r2ml:contextArgument>
  <r2ml:ObjectVariable r2ml:name="rentalCar"/>
 </r2ml:contextArgument>
</r2ml:AttributeFunctionTerm>
```

is verbalized as

last_maintenance_date of rentalCar

2.2 Verbalizing Atoms

2.2.1 ObjectClassificationAtom

Since the main purpose of an object classification atom is to classify an object term and to derive (using a derivation rule) new classification for the object term we will use the controlled language sentence "is a".

A positive r2ml:ObjectClassificationAtom template:

<ObjectTerm> is a <classID>

If the atom is negated (i.e. r2ml:isNegated="true") then

<ObjectTerm> is not a <classID>

An XSLT template:

```
<xsl:template match="r2ml:ObjectClassificationAtom">
 <xsl:choose>
  <xsl:when test="string(./@r2ml:isNegated)='true'">
   <xsl:apply-templates  select="*[1]"/>
   <xsl:text> is not a </xsl:text>
   <xsl:value-of select="replace(string(@r2ml:classID), ':', '#')"/>
  </xsl:when>
  <xsl:otherwise>
   <xsl:apply-templates select="*[1]"/>
   <xsl:text> is a </xsl:text>
   <xsl:value-of select="replace(string(@r2ml:classID), ':', '#')"/>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

For example, the R2ML ObjectClassificationAtom

```
<r2ml:ObjectClassificationAtom r2ml:classID="Male">
  <r2ml:ObjectName r2ml:objectID="John"/>
</r2ml:ObjectClassificationAtom>
```

it is verbalized as

John is a Male.

In this example "John" is the name of an object of the class "Male". In order to express "subclass of" relationship in R2ML we use the general template for R2ML implication 3.4.1.

2.2.2 DataClassificationAtom

Similar with r2ml:ObjectClassificationAtom, a r2ml:DataClassificationAtom classify data terms to datatypes. Here we use the same controlled language sentence, "is a".

- A positive DataClassificationAtom is verbalized as:

  <DataTerm> is a <datatypeID>

- If the atom is negated then

  <DataTerm> is not a <datatypeID>

An XSLT template:

```
<xsl:template match="r2ml:DataClassificationAtom">
 <xsl:choose>
  <xsl:when test="string(./@r2ml:isNegated)='true'">
```

```
   <xsl:apply-templates select="*[1]"/>
   <xsl:text> is not </xsl:text>
   <xsl:value-of select="@r2ml:datatypeID"/>
  </xsl:when>
  <xsl:otherwise>
   <xsl:apply-templates select="*[1]"/>
   <xsl:text> is </xsl:text>
   <xsl:value-of select="@r2ml:datatypeID"/>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

2.2.3 ReferencePropertyAtom

The reference property atom uses a referencePropertyID to associate an object term as "subject" with other object term as "object". This atom corresponds to the RDF triple concept. The patterns used here are:

- A positive r2ml:ReferencePropertyAtom is mapped into

  <subject> has <object> as <referencePropertyID>

- A negative r2ml:ReferencePropertyAtom is mapped into

  <subject> does not have <object> as <referencePropertyID>

An XSLT template:

```
<xsl:template match="r2ml:ReferencePropertyAtom">
 <xsl:choose>
  <xsl:when test="string(./@r2ml:isNegated)='true'">
   <xsl:apply-templates select="r2ml:subject"/>
   <xsl:text> does not have </xsl:text>
   <xsl:apply-templates select="r2ml:object"/>
   <xsl:text> as </xsl:text>
   <xsl:value-of select="@r2ml:referencePropertyID"/>
  </xsl:when>
  <xsl:otherwise>
   <xsl:apply-templates select="r2ml:subject"/>
   <xsl:text> has </xsl:text>
   <xsl:apply-templates select="r2ml:object"/>
   <xsl:text> as </xsl:text>
   <xsl:value-of select="@r2ml:referencePropertyID"/>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

Note, that this template in its present form does not provide a good natural verbalization all the time since a value of *referencePropertyID*, used in the template above can be non-natural (for instance, "hasCar"). In order to provide a natural verbalization of a reference property, the

range role name annotation should be available in the vocabulary. For example, the following ReferencePropertyAtom

```
<r2ml:ReferencePropertyAtom r2ml:referencePropertyID="hasCar">
 <r2ml:subject>
   <r2ml:ObjectName r2ml:objectID="Maria"/>
 </r2ml:subject>
 <r2ml:object>
  <r2ml:ObjectName r2ml:objectID="Audi"/>
 </r2ml:object>
</r2ml:ReferencePropertyAtom>
```

is verbalized as:

> Maria has Audi as car.

We assume, that the value "car" is taken from the underlying vocabulary for the rule and contains the "car" as an annotation for the reference property "hasCar". The current implementation of the verbalizer does not take this issue into consideration, but it is in our immediate plans to add it. The example above is not a correct English sentence since the noun "car" misses a determiner. If the property "hasCar" is functional, then the correct verbalization is "Maria has Audi as the car". If the property is not functional, then the correct verbalization is "Maria has Audi as a car". The current verbalizer does not take the determiners issue into account, but will support it in future versions.

2.2.4 AssociationAtom

- A positive r2ml:AssociationAtom:

  <objectArgument1> <associationPredicateID> <objectArgument2>

- A negated AssociationAtom is verbalized according to a annotation property, stored for the associationPredicateID in the vocabulary.

An XSLT template:

```
<xsl:template match="r2ml:AssociationAtom">
 <xsl:choose>
  <xsl:when test="string(./@r2ml:isNegated)='true'">
   <xsl:apply-templates select="r2ml:objectArguments/*[1]"/>
   <xsl:text> not </xsl:text>
   <xsl:value-of select="@r2ml:associationPredicateID"/>
   <xsl:text> </xsl:text>
   <xsl:apply-templates select="r2ml:objectArguments/*[2]"/>
  </xsl:when>
  <xsl:otherwise>
   <xsl:apply-templates select="r2ml:objectArguments/*[1]"/>
   <xsl:text> </xsl:text>
   <xsl:value-of select="@r2ml:associationPredicateID"/>
   <xsl:text> </xsl:text>
```

```
   <xsl:apply-templates select="r2ml:objectArguments/*[2]"/>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

The following AssociationAtom

```
<r2ml:AssociationAtom r2ml:isNegated="false"
                      r2ml:associationPredicateID="srv:isStoredAt">
 <r2ml:objectArguments>
  <r2ml:ObjectVariable r2ml:name="RentalCar"/>
  <r2ml:ObjectVariable r2ml:name="branch"/>
 </r2ml:objectArguments>
</r2ml:AssociationAtom>
```

verbalize as:

RentalCar isStoredAt branch.

The verbalization quality of this atom can be improved by retrieving natural language values from the underlying vocabulary in a way, similar to ReferencePropertyAtom 2.2.3. Then the above example will look like "RentalCar is stored at branch".

We describe the issue of accessing annotation property values for predicates in section 4. The annotation of OWL ontologies with natural language annotations is describe in [9].

The verbalization of n-ary associations is a subject for further improvements of the verbalizer. Ternary associations have wide practical usage. In order to verbalize them we have to introduce special patterns, which are stored in the vocabulary and describe the structure of the natural language phrase for a particular association.

2.2.5 AttributionAtom

An attribution atom consists of a reference to an attribute, an object term as "subject" and a data term as "value". Since its intended meaning is to associate a value to an attribute of an object we verbalize by using the pattern:

- A positive r2ml:AttributionAtom is mapped into:

  <ObjectTerm> has <DataTerm> as <attributeID>.

- A positive r2ml:AttributionAtom is mapped into:

  <ObjectTerm> does not have <DataTerm> as <attributeID>.

An XSLT template:

```
<xsl:template match="r2ml:AttributionAtom">
 <xsl:choose>
  <xsl:when test="string(./@r2ml:isNegated)='true'">
   <xsl:apply-templates select="r2ml:subject"/>
   <xsl:text> does not have </xsl:text>
   <xsl:apply-templates select="r2ml:value"/>
```

```
   <xsl:text> as </xsl:text>
   <xsl:value-of select="string(@r2ml:attributeID)"/>
  </xsl:when>
  <xsl:otherwise>
   <xsl:apply-templates select="r2ml:subject"/>
   <xsl:text> has </xsl:text>
   <xsl:apply-templates select="r2ml:value"/>
   <xsl:text> as </xsl:text>
   <xsl:value-of select="string(@r2ml:attributeID)"/>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

For example, the AttributionAtom

```
<r2ml:AttributionAtom r2ml:attributeID="discount">
 <r2ml:subject>
  <r2ml:ObjectVariable r2ml:name="customer"/>
 </r2ml:subject>
 <r2ml:value>
  <r2ml:TypedLiteral r2ml:datatypeID="xs:decimal" r2ml:lexicalValue="7.5"/>
 </r2ml:value>
</r2ml:AttributionAtom>
```

is verbalized as

> Customer has 7.5 as discount.

2.2.6 EqualityAtom and InequalityAtom

The r2ml:EqualityAtom and r2ml:InequalityAtom are mapped into

> <term1> is equal to <term2>
> <term1> is different from <term2>

An XSLT template:

```
<xsl:template match="r2ml:EqualityAtom">
 <xsl:choose>
  <xsl:when test="string(./@r2ml:isNegated)='true'">
   <xsl:apply-templates select="*[1]"/>
   <xsl:text> is different from </xsl:text>
   <xsl:apply-templates select="*[2]"/>
  </xsl:when>
  <xsl:otherwise>
   <xsl:apply-templates select="*[1]"/>
   <xsl:text> is equal to </xsl:text>
   <xsl:apply-templates select="*[2]"/>
  </xsl:otherwise>
 </xsl:choose>
```

```
</xsl:template>
<xsl:template match="r2ml:InequalityAtom">
 <xsl:choose>
  <xsl:when test="string(./@r2ml:isNegated)='true'">
   <xsl:apply-templates select="*[1]"/>
   <xsl:text> is equal to </xsl:text>
   <xsl:apply-templates select="*[2]"/>
  </xsl:when>
  <xsl:otherwise>
   <xsl:apply-templates select="*[1]"/>
   <xsl:text> is different from </xsl:text>
   <xsl:apply-templates select="*[2]"/>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

For example, the EqualityAtom

```
<r2ml:EqualityAtom r2ml:isNegated="true">
 <r2ml:ObjectVariable r2ml:name="returnBranch"/>
 <r2ml:ObjectVariable r2ml:name="pickupBranch"/>
</r2ml:EqualityAtom>
```

is verbalized as

> ReturnBranch is different from pickupBranch.

2.2.7 DatatypePredicateAtom

DatatypePredicateAtom captures different built-in predicates. For instance, the template for SWRL built-in swrlb:greaterThan is:

> <argument1> is greater than <argument2>

An XSLT template:

```
<xsl:template match="r2ml:DatatypePredicateAtom"
              xmlns:swrlb="http://www.w3.org/2003/11/swrlb">
 <xsl:for-each select="r2ml:dataArguments">
  <xsl:choose>
   <xsl:when test="string(../@r2ml:datatypePredicate)
                 = 'swrlb:greaterThan'">
    <xsl:apply-templates select="*[1]"/>
    <xsl:text> is greater than </xsl:text>
    <xsl:apply-templates select="*[2]"/>
   </xsl:when>
  </xsl:choose>
 </xsl:for-each>
</xsl:template>
```

For other built-ins templates are similar.

2.2.8 ObjectDescriptionAtom

An ObjectDescriptionAtom is a convenience construct used to encapsulate an object together with a number of property/term pairs (attribute data term pairs and reference property object term pairs). It refers to a class as a base type and to zero or more classes as categories. The template for ObjectDescriptionAtom is:

- A positive ObjectDescriptionAtom is mapped into:

  The <classID> <subject> has <dataSlotValue> as <attributeID> , ... and <objectSlotObject> as <referencePropertyID>.

- A negative ObjectDescriptionAtom is mapped into:

  The <classID> <subject> does not have <dataSlotValue> as <attributeID> or ..., or <objectSlotObject> as <referencePropertyID>.

Below is an XSLT template for a positive ObjectDescriptionAtom, template for a negated one can be easily created in a similar way:

```
<xsl:template match="r2ml:ObjectDescriptionAtom">
 <xsl:text> The </xsl:text>
 <xsl:value-of select="replace(string(@r2ml:classID),':','#')"/>
 <xsl:apply-templates select="r2ml:subject"/>
 <xsl:text> has </xsl:text>
 <xsl:for-each select="r2ml:DataSlot|r2ml:ObjectSlot">
  <xsl:apply-templates select="r2ml:object|r2ml:value"/>
  <xsl:text> as </xsl:text>
  <xsl:value-of select="replace(string(@r2ml:referencePropertyID|@r2ml:attributeID),':','#')"/>
  <xsl:choose>
  <xsl:when test="position() != last()">
   <xsl:text>, </xsl:text>
  </xsl:when>
  <xsl:otherwise>
   <xsl:text> and </xsl:text>
  </xsl:otherwise>
  </xsl:choose>
 </xsl:for-each>
</xsl:template>
```

For example, the ObjectDescriptionAtom

```
<r2ml:ObjectDescriptionAtom r2ml:class="Person">
 <r2ml:subject>
  <r2ml:ObjectName r2ml:objectID="John"/>
</r2ml:subject>
 <r2ml:DataSlot r2ml:attributeID="age">
 <r2ml:value>
   <r2ml:TypedLiteral r2ml:datatypeID="xs:positiveInteger" r2ml:lexicalValue="25"/>
</r2ml:value>
```

```
</r2ml:DataSlot>
<r2ml:ObjectSlot r2ml:referencePropertyID="wife">
  <r2ml:object>
    <r2ml:ObjectName r2ml:objectID="Mary"/>
  </r2ml:object>
</r2ml:ObjectSlot>
</r2ml:ObjectDescriptionAtom>
```

is verbalized as

> The Person John has 25 as age and Mary as wife.

Chapter 3

Rule Templates

The R2ML supports four rule types: integrity rules, derivation rules, production rules, and reaction rules ([8], [9]). Each rule type is verbalized according to a template, defined in a form of XSL transformation. A template describes a structure of the rule in a controlled natural language. A template contains terms of controlled language like "If", "then", etc. For instance, an English template for a derivation rule is as follows:

```
<xsl:template match="r2ml:DerivationRule">
 <xsl:text> If </xsl:text>
 <xsl:apply-templates select="r2ml:conditions"/>
 <xsl:text> then </xsl:text>
 <xsl:apply-templates select="r2ml:conclusion"/>
 <xsl:text>.</xsl:text>
</xsl:template>
```

A derivation rule then is verbalized as follows:

If <conditions> then <conclusion>.

Such XSLT template is resolved by the verbalizer for each language via `xml:lang` attribute. Then the XSLT template is applied to the rule base.

3.1 Derivation Rules

The derivation rule template is defined in the previous section and consists of conditions and a conclusion.

3.1.1 Conditions and conclusion

Conditions and conclusion of a derivation rule as well as postconditions of production rule and reaction rule are verbalized as a conjunction of the corresponding R2ML atoms:

<atom1> and <atom2>

We make a restrictive assumption here that each element in the conditions is a sentence literal (atom or negated atom).

```
<xsl:template match="r2ml:conditions|r2ml:Conjunction|
                     r2ml:conclusion|r2ml:postconditions">
 <xsl:for-each select="child::node()[name() != name(comment())]">
  <xsl:choose>
   <xsl:when  test="position() != last()">
    <xsl:apply-templates  select="."/>
    <xsl:text> and </xsl:text>
   </xsl:when>
   <xsl:otherwise>
    <xsl:apply-templates  select="."/>
   </xsl:otherwise>
  </xsl:choose>
 </xsl:for-each>
</xsl:template>
```

Disjunction is verbalized in similar way by using disjunction term "or".

3.2 Production Rules

A production rule consists of conditions, a produced action and, optionally, a post-condition. From the verbalization point of view it is similar to derivation rules just adding a conjunctive construct if the post-condition exists. A production rule is verbalized as follows:

> If <conditions> then do <producedAction> and <postCondition>.

```
<xsl:template match="r2ml:ProductionRule">
 <xsl:text> If </xsl:text>
 <xsl:apply-templates select="r2ml:conditions"/>
 <xsl:text> then do </xsl:text>
 <xsl:apply-templates select="r2ml:producedAction"/>
 <xsl:text> and </xsl:text>
 <xsl:apply-templates select="r2ml:postCondition"/>
 <xsl:text>.</xsl:text>
</xsl:template>
```

A production rule postcondition is verbalized as a condition of a derivation rule.

3.2.1 AssignActionExpression

The intended meaning of an AssignActionExpression is to update a property (Attribute or ReferenceProperty) value (Term) of a specific object ("contextArgument"). The controlled English template for the assign action expression is as follows:

> assign <value> to the <property> of the <contextArgument>

XSLT template:

```
<xsl:template match="r2ml:AssignActionExpression">
 <xsl:text> assign </xsl:text>
```

```
 <xsl:apply-templates select="child::node()[position() =2]"/>
 <xsl:text> to the </xsl:text>
 <xsl:value-of select="string(@r2ml:propertyID)"/>
 <xsl:text> of the </xsl:text>
 <xsl:apply-templates select="r2ml:contextArgument"/>
</xsl:template>
```

For example, the following AssignActionExpression

```
<r2ml:AssignActionExpression r2ml:propertyID="discount">
 <r2ml:contextArgument>
  <r2ml:ObjectVariable r2ml:name="o1" r2ml:classID="Order"/>
 </r2ml:contextArgument>
 <r2ml:TypedLiteral r2ml:lexicalValue="10" r2ml:datatypeID="xs:positiveInteger"/>
</r2ml:AssignActionExpression>
```

is verbalized as

> assign 10 to the discount of the Order o1

3.2.2 InvokeActionExpression

InvokeActionExpression models a request to invoke an operation on an object. 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. English template for the invoke action expression is as follows:

> invoke <operation> with <arguments> and with the context <contextArgument>

XSLT template:

```
<xsl:template match="r2ml:InvokeActionExpression">
 <xsl:text> invoke </xsl:text>
 <xsl:value-of select="local-name-from-QName(@r2ml:operationID)"/>
 <xsl:text> with </xsl:text>
 <xsl:apply-templates select="r2ml:arguments"/>
 <xsl:text> and with the context </xsl:text>
 <xsl:apply-templates select="r2ml:contextArgument"/>
</xsl:template>
```

The next InvokeActionExpression

```
<r2ml:InvokeActionExpression r2ml:operationID="sendMessage">
 <r2ml:arguments>
  <r2ml:ObjectVariable r2ml:name="sender" r2ml:classID="Contact"/>
  <r2ml:ObjectVariable r2ml:name="receiver" r2ml:classID="Contact"/>
 </r2ml:arguments>
</r2ml:InvokeActionExpression>
```

is verbalized as

> invoke sendMessage with Contact sender as argument 1, Contact receiver as argument 2

3.2.3 Create Action

CreateAction refers to a Class and contains a list of slots (object slots and/or data slots). The execution of this action is a constructor call for the creation of a new object in the system. The controlled English template for the create action expression is as follows:

create <class> with <slot1>,... and <slotN>

$< slot >$ is a list of parameters for the $< class >$ constructor.
XSLT template:

```
<xsl:template match="r2ml:CreateActionExpression">
 <xsl:text> create </xsl:text>
 <xsl:value-of select="local-name-from-QName(@r2ml:classID)"/>
 <xsl:text> with </xsl:text>
 <xsl:apply-templates select="r2ml:slot"/>
</xsl:template>
```

A list of R2ML slots is verbalized as a list of parameters, separated by ", and". Each slot is verbalized as a

<value> as <attribute>

The following CreateActionExpression

```
<r2ml:CreateActionExpression r2ml:classID="Person">
 <r2ml:DataSlot r2ml:attribute="name">
  <r2ml:value>
   <r2ml:TypedLiteral r2ml:lexicalValue="John" r2ml:datatypeID="xs:string"/>
  </r2ml:value>
 </r2ml:DataSlot>
 <r2ml:DataSlot r2ml:attribute="dateOfBirth">
  <r2ml:value>
   <r2ml:PlainLiteral r2ml:lexicalValue="January 1, 1980" r2ml:languageTag="en"/>
  </r2ml:value>
 </r2ml:DataSlot>
</r2ml:CreateActionExpression>
```

is verbalized as

create Person with John as name and January 1, 1980 as dateOfBirth

3.2.4 DeleteActionExpression

DeleteActionExpression refers to a Class and contains an ObjectTerm. This action removes an instance of the Class. The English template for the delete action expression is as follows:

delete <classID> <contextArgument>

XSLT template:

```
<xsl:template match="r2ml:DeleteActionExpression">
 <xsl:text> delete </xsl:text>
 <xsl:value-of select="local-name-from-QName(@r2ml:classID)"/>
 <xsl:text> </xsl:text>
 <xsl:apply-templates select="r2ml:contextArgument"/>
</xsl:template>
```

The following DeleteActionExpression

```
<r2ml:DeleteActionExpression r2ml:classID="Person">
 <r2ml:contextArgument>
  <r2ml:ObjectName r2ml:objectID="p123"/>
 </r2ml:contextArgument>
</r2ml:DeleteActionExpression>
```

is verbalized as

> delete Person p123

3.3 Reaction Rules

A reaction rule is a statement that specifies the execution of one or more actions in the case of a triggering event occurrence and if rule conditions are satisfied. A reaction rule may have an optional postcondition to be satisfied. A reaction rule template is as follows:

> On <triggeringEvent> if <conditions> then do <producedAction> resulting in <postcondition>.

XSLT template:

```
<xsl:template match="r2ml:ReactionRule">
  <xsl:text> On </xsl:text>
  <xsl:apply-templates select="r2ml:triggeringEvent"/>
  <xsl:text> if </xsl:text>
  <xsl:apply-templates select="r2ml:conditions"/>
  <xsl:text> then do </xsl:text>
  <xsl:apply-templates select="r2ml:producedAction"/>
  <xsl:text> resulting in </xsl:text>
  <xsl:apply-templates select="r2ml:postcondition"/>
  <xsl:text>.</xsl:text>
  <xsl:text>&#xA;</xsl:text>
 </xsl:template>
```

3.3.1 R2ML MessageEventExpression

Reaction rule MessageEventExpression is mapped into:

> <messageType> from <sender> with <arguments>

XSLT template:

```
<xsl:template match="r2ml:MessageEventExpression">
 <xsl:value-of select="@r2ml:eventType"/>
 <xsl:text> from </xsl:text>
 <xsl:value-of select="replace(string(@r2ml:sender), ':','#')"/>
 <xsl:text> with </xsl:text>
 <xsl:apply-templates select="r2ml:arguments"/>
</xsl:template>
```

The following MessageEventExpression

```
<r2ml:MessageEventExpression
         r2ml:eventType="alert"
         r2ml:startTime="2006-03-21T09:00:00"
         r2ml:duration="P0Y0M0DT0H0M0S"
         r2ml:sender="http://www.mywebsite.org">
 <r2ml:arguments>
  <r2ml:ObjectVariable r2ml:name="car"/>
  <r2ml:ObjectVariable r2ml:name="customer"/>
 </r2ml:arguments>
</r2ml:MessageEventExpression>
```

is verbalized as

> alert from http://www.mywebsite.org with car, customer

3.4 Integrity Rules

There are different types of integrity rules [4]. In this report we describe templates for R2ML integrity rules in a form of implication and cardinality restrictions in a form of R2ML AtLeastQuantifiedFormula, AtMostQuantifiedFormula and AtLeastAndAtMostQuantifiedFormula. In the current implementation of the verbalization method R2ML implications are verbalized using "If - then" (Derivation rules-like style) template.

3.4.1 R2ML Implication

XSLT template:

```
<xsl:template match="r2ml:Implication">
 <xsl:text> If </xsl:text>
 <xsl:apply-templates select="r2ml:antecedent"/>
 <xsl:text> then </xsl:text>
 <xsl:apply-templates select="r2ml:consequent"/>
</xsl:template>
```

The described template for the implication can be used for verbalizing "subclass of" relationship. Other templates for verbalizing R2ML integrity rules, which can be used to express "subclass of" relationship is a subject for further investigation.

3.4.2 R2ML UniversallyQuantifiedFormula

For all <variables>, <LogicalFormula>.

XSLT template:

```
<xsl:template match="r2ml:UniversallyQuantifiedFormula">
 <xsl:text> For all </xsl:text>
 <xsl:apply-templates select="r2ml:ObjectVariable|r2ml:DataVariable|r2ml:Variable"/>
 <xsl:text>, </xsl:text>
 <xsl:apply-templates select="r2ml:Implication|r2ml:Conjunction|r2ml:Disjunction"/>
</xsl:template>
```

The following

```
<r2ml:UniversallyQuantifiedFormula>
 <r2ml:ObjectVariable r2ml:name="x"/>
 <r2ml:Disjunction>
  <r2ml:ObjectClassificationAtom r2ml:classID="Meat">
   <r2ml:ObjectVariable r2ml:name="x"/>
  </r2ml:ObjectClassificationAtom>
  <r2ml:ObjectClassificationAtom r2ml:classID="Fish">
   <r2ml:ObjectVariable r2ml:name="x"/>
  </r2ml:ObjectClassificationAtom>
 </r2ml:Disjunction>
</r2ml:UniversallyQuantifiedFormula>
```

is verbalized as:

For all x, x is a Meat or x is a Fish

3.4.3 R2ML ExistentiallyQuantifiedFormula

Exists <variables>, such that <LogicalFormula>.

XSLT template:

```
<xsl:template match="r2ml:UniversallyQuantifiedFormula">
 <xsl:text> Exists </xsl:text>
 <xsl:apply-templates select="r2ml:ObjectVariable|r2ml:DataVariable|r2ml:Variable"/>
 <xsl:text>, such that </xsl:text>
 <xsl:apply-templates select="r2ml:Implication|r2ml:Conjunction|r2ml:Disjunction"/>
</xsl:template>
```

The following ExistentiallyQuantifiedFormula

```
<r2ml:ExistentiallyQuantifiedFormula>
<r2ml:ObjectVariable r2ml:name="x"/>
 <r2ml:Conjuction>
  <r2ml:Disjunction>
   <r2ml:ObjectClassificationAtom r2ml:classID="Professor">
```

```
    <r2ml:ObjectVariable r2ml:name="x1"/>
   </r2ml:ObjectClassificationAtom>
   <r2ml:ObjectClassificationAtom r2ml:classID="AssociateProfessor">
    <r2ml:ObjectVariable r2ml:name="x1"/>
   </r2ml:ObjectClassificationAtom>
  </r2ml:Disjunction>
  <r2ml:ReferencePropertyAtom r2ml:refPropertyID="advisor">
   <r2ml:subject>
    <r2ml:ObjectVariable r2ml:name="x1"/>
   </r2ml:subject>
   <r2ml:object>
    <r2ml:ObjectVariable r2ml:name="x"/>
   </r2ml:object>
  </r2ml:ReferencePropertyAtom>
 </r2ml:Conjuction>
</r2ml:ExistentiallyQuantifiedFormula>
```

is verbalized as

> exists x, such that x1 is a Professor or x1 is a AssociateProfessor and x1 has x as advisor

3.4.4 AtMostQuantifiedFormula

The R2ML AtMostQuantifiedFormula is used to specify cardinality constraints. It can be used with any kind of R2ML atom. The most widely used are AssociationAtom, ReferenceProperty-Atom and AttributionAtom, which help to specify cardinality of associations, properties and attributes. The example template for binary AssociationAtom is as follows:

> Each <subject> <associationPredicate> at most <maxCardinality> <object>

XSLT template for verbalizing cardinality restrictions on associations:

```
<xsl:template match="r2ml:UniversallyQuantifiedFormula">
  <xsl:text> Each </xsl:text>
  <xsl:apply-templates select="*[1]"/>
  <xsl:text> </xsl:text>
  <xsl:value-of
     select="r2ml:AtMostQuantifiedFormula/r2ml:AssociationAtom/@r2ml:associationPredicate"/>
  <xsl:text> at most </xsl:text>
  <xsl:value-of select="r2ml:AtMostQuantifiedFormula/@r2ml:maxCardinality"/>
  <xsl:text> </xsl:text>
  <xsl:apply-templates
    select="r2ml:AtMostQuantifiedFormula/r2ml:AssociationAtom/r2ml:objectArguments/*[2]"/>
 </xsl:template>
```

For example, the following R2ML formula:

```
<r2ml:UniversallyQuantifiedFormula>
 <r2ml:ObjectVariable r2ml:name="car"/>
 <r2ml:AtMostQuantifiedFormula  r2ml:maxCardinality="1">
  <r2ml:ObjectVariable r2ml:name="branch"/>
  <r2ml:AssociationAtom r2ml:associationPredicate="isStoredAt">
   <r2ml:objectArguments>
     <r2ml:ObjectVariable r2ml:name="car"/>
     <r2ml:ObjectVariable r2ml:name="branch"/>
   </r2ml:objectArguments>
  </r2ml:AssociationAtom>
 </r2ml:AtMostQuantifiedFormula>
</r2ml:UniversallyQuantifiedFormula>
```

is verbalized as

> Each car isStoredAt at most 1 branch.

The verbalization templates for R2ML AtLeastQuantifiedFormula and AtLeastAndAtMostQuantifiedFormula are similar. The varieties of different formulas, which can be used to express cardinality restrictions are still under investigation.

Chapter 4

Resolving Vocabulary Terms by URIs

The R2ML supports internal and external vocabularies, which can be expressed in different ontology languages. Internal vocabulary can be expressed in R2ML and the full vocabulary support by R2ML will be provided starting from the R2ML version 0.4. As an external vocabulary any existing ontology language can be used. The R2ML verbalizer uses a specially developed Java API to access any external vocabulary. Currently, there is a support of the API for OWL [7] and support for RDF [6] is in plans.

The verbalization of OWL ontologies has been described in the REWERSE I1 Deliverable D8 [9] and OWL verbalizer has been implemented. The I1 method of the OWL ontologies verbalization is based on the ontology annotation with special annotation properties, which stores natural language expressions for noun phrases, verb phrases and domain and range roles.

XSLT templates, described in the Section 3 use term URIs. In order to provide natural language expressions instead of URIs, the templates have to be modified with Java API function calls. First, we have to define a namespace for Java in the XSLT:

```
<xsl:stylesheet version="2.0"
     xmlns:java="de.tu_cottbus.ontology.OntologyAccessInt" ... >
```

Then we have to instantiate an appropriate class for the ontology language, specified in the R2ML rule base:

```
1  <xsl:variable name="vocURI" select="/r2ml:RuleBase@externalVocabulary"/>
2  <xsl:variable name="vocLanguage"
        select="/r2ml:RuleBase@externalVocabularyLanguage"/>
3  <xsl:variable name="verbAccess" select="java:new($vocURI, $vocLanguage)" />

4  <xsl:variable name="nc" select="/r2ml:RuleBase"/>
```

The variable `verbAccess` is an instance of a Java class `OntologyAccessInt` for the OWL ontology with URI `vocURI`. Line 4 stores the root element of R2ML rule base in the `nc` variable. Rule vocabulary namespaces are defined in this element. Class and property URIs, which are used in rules, are in the form of prefix:localname, for instance, `srv:PremiumCustomer`, where

prefix `srv` corresponds to the namespace, defined in the `/r2ml:RuleBase` element. The variable `nc` is used as an argument for XPath functions in order to resolve full URI for classes and properties, for instance, to resolve `http://www.rewerse.net/I1/eu-rent#PremiumCustomer` from `srv:PremiumCustomer`. There is the following code in the template for object classification atom 2.2.1:

```
<xsl:value-of select="replace(string(@r2ml:class), ':', '#')"/>
```

It returns the URI of a class, for instance, `srv:PremiumCustomer`. In order to insert a natural language value for this class, which is stored in the ontology, for instance, "premium customer", we have to modify the code:

```
1    <xsl:variable name="fu" select="resolve-QName(string(@r2ml:classID), $nc)"/>
2    <xsl:variable name="classID" select="concat(namespace-uri-from-QName($fu),
                                        local-name-from-QName($fu))"/>
3    <xsl:value-of select="java:getNounPhraseByClassID($verbAccess, $classID)"/>
```

The first line creates a QName variable from the prefixed URI of the class and namespace of the element $nc. The second line resolves a full (expanded) URI of the class and stores it in the variable `classID`. The third line invokes java method `getNounPhraseByClassID`, which returns corresponding natural language value for the class from the OWL ontology.

Chapter 5

Further Improvements

The current version of the R2ML verbalizer is available as a Web service[1].

Below is the list of further improvements to the described verbalizer:

1. Improvements of verbalization templates to cover different types of integrity rules. As described in [4], there are a lot of types of integrity rules. All these rule types can be expressed in R2ML and verbalized, but since R2ML has a rich syntax, one rule can be marked up in different ways. We still investigate the common and most usable templates for different types of R2ML integrity rules and for different rule modalities.

2. The use of correct determiners in ReferencePropertyAtom, AssociationAtom, and AttributionAtom.

3. Verbalization templates for operations, defined in the R2ML vocabulary, in order to improve the verbalization quality of function and operation terms.

4. The verbalizer is going to be integrated with I1 rule interchange service[2]. The service uses R2ML as a rule interchange language between other existing rule languages. Any rule language, supported by the service, can be verbalized using described verbalizer. The verbalization feature of this service will show an added value of the verbalization when a user wants to validate the correctness of her rules. Natural language representation of the rule, provided by the verbalizer during the transformation process may help the user to find inconsistencies in his rule bases and rule modeling mistakes.

5. Integration of the verbalizer with I1 rule modeling tool Strelka is a subject for future work.

6. The R2ML verbalizer supports OWL as an ontology language. Newer versions of R2ML will support vocabularies in R2ML and the verbalizer will be extended to support vocabularies in RDF as well.

[1] R2ML verbalizer homepage: http://oxygen.informatik.tu-cottbus.de/verbalization/index.jsp
[2] I1 rule interchange service: http://oxygen.informatik.tu-cottbus.de:8080/translators/r2ml

Acknowledgment

We thank Adrian Giurca for fruitful discussions.

Bibliography

[1] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernandez, Michael Kay, Jonathan Robie, Jerome Simeon, XML Path Language (XPath) 2.0, W3C Candidate Recommendation 8 June 2006, `http://www.w3.org/TR/xpath20/`.

[2] Berners Lee, T., Hendler J., Lassila, O., The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities, Scientific American May 2001.

[3] Michael Kay (Ed.), XSL Transformations (XSLT) Version 2.0, W3C Candidate Recommendation 8 June 2006, `http://www.w3.org/TR/xslt20/`.

[4] Jarrar M., Keet, C. M.: An English Verbalization Template for ORM conceptual models and rules. A technical report of the article: Jarrar, M., Keet, C.M., Dongilli, P. Multilingual verbalization of ORM conceptual models and axiomatized ontologies. [Submitted].

[5] Jarrar M.: Towards methodological principles for ontology engineering . PhD Thesis. Vrije Universiteit Brussel. (May 2005)

[6] Klyne G., Caroll J.J. (Eds.), Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C, 2004.

[7] Patel-Schneider, Peter F., Horroks I., OWL Web Ontology Language Semantic and Abstract Syntax, `http://www.w3.org/2004/OWL`.

[8] Wagner, G., Giurca, A., Lukichev, S. (2006). A Usable Interchange Format for Rich Syntax Rules. Integrating OCL, RuleML and SWRL. In proceedings of Reasoning on the Web Workshop at WWW2006, May 2006.

[9] Wagner, G., Giurca, A., Lukichev, S. REWERSE I1 Deliverable D8, Language Improvements and Extentions, March 2006.

[10] Mustafa Jarrar, Maria Keet, Paolo Dongilli: Multilingual verbalization of ORM conceptual models and axiomatized ontologies. Technical report. Vrije Universiteit Brussel, February 2006.