

Sharing rules between JBoss and Jena

Oana Nicolae, Ion-Mircea Diaconescu, Adrian Giurca and Gerd Wagner

Department of Internet Technology

Institute of Informatics

Brandenburg Technical University at Cottbus, Germany

{nicolae, M.Diaconescu, giurca, G.Wagner}@tu-cottbus.de

Abstract

The actual rule based implementations point to different rule target platforms and languages (Object Oriented Rule Languages, Semantic Web Rule Languages, Artificial Intelligence Rule Languages) in a business landscape dominated by a limited number of tool vendors as well as competitive open-source platforms and rule interoperability initiatives. This work provides a description of business rules translation from JBossRules - an Object Oriented rule-system, to JenaRules - a Semantic Web rule-system, via an interchange language. It focuses on the general lines of the JBoss-R2ML-Jena translation and to outline the limitations and on the correctness of the proposed interchange.

1. Motivation

This paper proposes a solution to interchange rules between two implementation languages: JBossRules [3] and JenaRules [2], using R2ML production rules [4], as interchange format. A tutorial about rule modeling and rule interchange can be read online at: <http://hydrogen.informatik.tu-cottbus.de/moodle/course/view.php?id=24>.

Our rule interchange work addresses JBossRules as source platform and JenaRules as a target platform, bridging this way an object-oriented rule language to a Semantic Web rule language.

JBossRules is a production rule system, used by Java developers to create complex rule-based applications by combining Java platform and business rule technology. It is an open source rule engine, has a quick integration with the mainstream JEE5 technologies and its well known in the actual business rules market, therefore is a good choice in an interchange experiment.

The target language, JenaRules is a complex environment by embedding the Semantic Web ontology reasoning together with rules reasoning i.e. by combining knowledge representation and ontology modeling with natural language processing techniques, information retrieval, artificial intelligence and markup languages.

This paper is a new argument to the necessity of a commonly agreed rule interchange format, a *lingua franca* into which rules can be mapped, published, shared and re-used between different systems and tools.(See RIF [1], R2ML [5]).

2. Mapping JBossRules to JenaRules

This section describes the general lines of JBossRules transformation into JenaRules using R2ML markup language. We introduce the JBossRules informally, in terms of a rule example¹ i.e.:

```
1.import userv.Driver;
2. rule "HighRiskDriver"
3. when
4.   driver:Driver(numberOfAccidents > 2)
5. then
6.   driver.setIsHighRiskDriver(true);
7.   modify(driver);
8.end
```

2.1. Mapping rules vocabularies

Object oriented rules as JBossRules are build on top of Java vocabularies. JBoss business rules vocabulary consists of Java beans. This vocabulary is used by the rules through the `import` declarations, which are specified inside of the rules file (`dr1` files or `xml` files). The above JBoss rule use `Driver` bean as vocabulary.

¹The JBoss rule example belongs to the UServ Product Derby 2005, which is an Use Case defined by the Business Rules Forum <http://www.businessrulesforum.com/>. as a benchmark for rule systems. We implemented this Use Case at <http://oxygen.informatik.tu-cottbus.de/userv/>

An R2ML rule may refer to a vocabulary which can be R2ML own vocabulary or an imported one (such as RDF(S)² and OWL³). R2ML vocabulary is a serialization of an UML fragment of class diagrams. R2ML uses XML Schema datatypes⁴ as its standard datatype set, therefore all standard Java datatypes maps into XML Schema datatypes (See JSR 31⁵).

Any Java qualified class names will be translated in R2ML into appropriate qualified names, as a reference to the class name, together with their corresponding namespace declarations (see Line 1 from the before example).

2.2. Rule mapping

The general mapping principles to translate a JBoss rule into an R2ML production rule are:

- The LHS part of a JBoss rule is mapped into the content of `r2ml:conditions` role element, while the RHS part of a JBoss rule containing multiple actions, maps into the content of `r2ml:producedAction` role element.
- Any JBoss variables (i.e. *declarations* in the JBoss terminology) translates into R2ML variables. The JBoss *fact variable* (e.g. `driver:Driver()`) is mapped into `r2ml:ObjectVariable` using the value of `r2ml:name` attribute to represent the variable name. The optional `r2ml:classID` attribute specifies the type of the object variable. `r2ml:ObjectVariable` identify a variable that can be only instantiated by objects.


```
<r2ml:ObjectVariable r2ml:name="driver"
  r2ml:classID="userserv:Driver" />
```
- Relational operations from JBossRules (e.g.>) translates into `r2ml:DatatypePredicateAtom` using SWRL built-ins as predicate names. Functions and operators like addition, subtraction etc are translated into corresponding R2ML function terms involving XPath⁶ functions and operators as function symbol.

The operands implied by the relational operations are comprised into `r2ml:dataArguments` attribute. Depending on the involved property they can be:

1. `r2ml:AttributeFunctionTerm` if the property represents a class attribute (see Line 4).
2. `r2ml:DataOperationTerm` if the property represents a class operation.
3. `r2ml:DatatypeFunctionTerm` if the property is an operator (e.g. `+`, `*` etc). The operator is designated using the appropriate XPath 2 built-ins (e.g. `"op:numeric-subtract"` defines the subtract arithmetic operation) as a value for `r2ml:datatypeFunctionID` attribute.
4. `r2ml:DataVariable` if the operand denotes a data typed JBoss variable.
5. `r2ml:TypedLiteral` if the operand represents a literal value.

- In JBossRules the conjunction of columns is by default represented as a column enumeration. A JBoss column may contain many field constraints all of them referring the same context variable.

An Excerpt of mapping from JBossRules to R2ML	
<pre>rule "ruleName" when // conditions then // actions end</pre>	<pre><r2ml:ProductionRule r2ml:ruleID="ruleName"> <r2ml:conditions> ... </r2ml:conditions> <r2ml:producedAction> ... </r2ml:producedAction> </r2ml:ProductionRule></pre>
import Java beans files	declare appropriate namespaces, corresponding qualified names (xs:QName) and R2ML vocabulary using <code><r2mlv:Vocabulary></code>
fact variable (optionally \$ symbol)	<code><r2ml:ObjectVariable></code>
field variable (optionally \$symbol)	<code><r2ml:ObjectVariable></code> or <code><r2ml:DataVariable></code>
column	<code><r2ml:ObjectClassificationAtom></code>
relational operations as field constraints	<code><r2ml:DatatypePredicateAtom></code> using SWRL builtins
bind field variable inside column (\$var:property) or check equality with bounded variable inside column (property == \$var)	<code><r2ml:ReferencePropertyAtom></code> for object type property or <code><r2ml:AttributionAtom></code> for data type property

- The comma represents the implicit connector between constraints and also corresponds to the JBoss conjunction of columns.
- The rule action consists in a setter call (see Line 6). It translates into appropriate `r2ml:AssignActionExpression`. The intended meaning of such an atom is to update a property of a specific object denoted by `r2ml:contextArgument` content. The property can be updated with a particular value or with an *operation*, which is also our case. The particular

²<http://www.w3.org/TR/rdf-schema/>

³<http://www.w3.org/2004/OWL/>

⁴XML Schema Part 2: Datatypes Second Edition, W3C Recommendation, 28 October 2004, <http://www.w3.org/TR/xmlschema-2/>

⁵<http://jcp.org/en/jsr/detail?id=031>

⁶XQuery 1.0 and XPath 2.0 Functions and Operators, W3C recommendation, 23 January 2007, <http://www.w3.org/TR/xpath-functions/>

JBoss structure (see Line 7) is used to dynamically update the property value of an object.

Jena rules can be created manually using a common text editor, or inside applications using a String data type i.e.:

```
String myRule = "[rule-name: conditions -> actions]";
```

R2ML serialization of the rule translates into JenaRules implementation i.e.:

```
9. [HighRiskDriver:
10.   (?driver rdf:type userv:Driver)
11.   (?driver userv:numberOfAccidents ?numberOfAccidents)
12.   greaterThan(?numberOfAccidents,2)
13.   ->
14.   remove(1)
15.   (?driver userv:isHighRiskDriver 'true' ^^xs:boolean)]
```

- The `r2ml:conditions` role element translates into Jena rule body, containing the RDF triples (see Lines 10-12 from Jena rule example).
- All R2ML variables are mapped into Jena variables, which have as first character the `?` symbol.⁷

An Excerpt of mapping from R2ML to JenaRules	
<pre><r2ml:ObjectClassificationAtom r2ml:classID="className"> <!--ObjectVariable/ObjectName--> </r2ml:ObjectClassificationAtom></pre>	<pre>(?objectVar rdf:type className)/ (objectName rdf:type className)</pre>
<pre><r2ml:ReferencePropertyAtom r2ml:referencePropertyID= "propertyName"> <r2ml:subject> <!--ObjectVariable/ObjectName--> </r2ml:subject> <r2ml:object> <!--ObjectVariable/ObjectName--> </r2ml:object> </r2ml:ReferencePropertyAtom></pre>	<pre>(?objVar propertyName ?objVar)/ (?objVar propertyName objName)/ (objName propertyName ?objVar)/ (objname propertyName objName)</pre>
<pre><r2ml:AttributionAtom r2ml:attributeID="attrName"> <r2ml:subject> <!--ObjectVariable/ObjectName--> </r2ml:subject> <r2ml:dataValue> <!--DataVariable/TypedLiteral/ PlainLiteral --> </r2ml:dataValue> </r2ml:AttributionAtom></pre>	<pre>(?objectVar attrName ?dataVar)/ (objectName attrName ?dataVar)/ (?objectVar attrName typedLit)/ (objectName attrName typedLit)/ (?objectVar attrName plainLit)/ (objectName attrName plainLit)</pre>
<pre><r2ml:DatatypePredicateAtom></pre>	<p>Jena primitive builtins, or user defined builtins</p>
<pre><r2ml:AssignActionExpression r2ml:propertyID="propName"> <r2ml:contextArgument> <!--ObjectVariable/ObjectName--> </r2ml:contextArgument> <!--DataVariable/TypedLiteral/ PlainLiteral --> </r2ml:AssignActionExpression></pre>	<pre>(?objectVar propName ?dataVar)/ (objectName propName ?dataVar)/ (?objectVar propName typedLit)/ (objectName propName typedLit)/ (?objectVar propName plainLit)/ (objectName propName plainLit)</pre>
<pre><r2ml:ObjectVariable r2ml:name="objectVarName"></pre>	<p>the value of the property <code>r2ml:name</code> become the subject of the RDF triple</p>
<pre><r2ml:AttributionFunctionTerm r2ml:attributeID="attrName"></pre>	<p>the value of the property <code>r2ml:attributeID</code> become the predicate of an RDF triple</p>
<pre><r2ml:TypedLiteral r2ml:lexicalValue="value" r2ml:datatypeID="type"></pre>	<p>become the object position of the RDF triple in the form of: "value"^^type</p>

⁷When the R2ML rule conditions does not contain a specific typing condition i.e. an `"r2ml:ObjectClassificationAtom"` or an `"r2ml:ObjectDescriptionAtom"` the appropriate typing information can be obtained from the attribute `r2ml:classID="className"` appearing in `r2ml:ObjectVariable` or `r2ml:ObjectName`.

- Some of the R2ML builtins predicates and operators (e.g. `"swrlb:greaterThan"`) translate into corresponding Jena *primitive built-ins* (see Line 12).

These built-ins can be used either in the rule *body*, in the rule *head part* or in both.

However, the user can define its own built-in predicates, in the form of a Java class. The `r2ml:DatatypePredicateAtom` translates into the Jena triple and appropriate primitive builtin (See Lines 11-12).

- The possible actions of a R2ML production rule are defined using OMG's PRR Proposal (the content of `r2ml:producedAction` role element). `r2ml:AssignActionExpression` from our R2ML rule serialization corresponds to the following Jena triple which asserts a new fact referring the property `isHighRiskDriver`. (Line 15).

- Jena language doesn't implement dynamically the updating procedure. When changing the value of a property, a new property-value pair fact will be added to the RDF factbase. To simulate the update of the property the old property-value pair must be removed. We achieve this, by invoking the `remove(index)` builtin, where `index` denotes the Jena triple index inside the rule condition part, starting from zero (see Line 14).

3. Limitations of translation process referring target platforms

When building a functional translation between two rule systems belonging to different rules semantic areas (i.e. JBossRules, an Object Oriented rule-system and JenaRules a Semantic Web rule-system), many problems regarding the correctness and the completeness of the translation may appear.

Discussions about the correctness of an interchange approach, usually involve the language semantics. Both JBossRules and JenaRules, don't provide such a semantics of the language, therefore the correctness of the translation was established by testing rules. The tests are available in the application online version⁸. We translate the JBoss production rules into JenaRules implementation via R2ML, and execute those rules based on analogous facts from the Working Memory. The correctness of the translation implies the same obtained results regarding the facts from Working Memory.

⁸UServ, <http://oxygen.informatik.tu-cottbus.de/userv/>

When translating rules from PSM to PIM, we need also the rules vocabulary. The JBossRules to R2ML translator is a Java application that demands access to the JBossRules vocabulary, that consists of Java beans compiled classes, in order to establish the types of objects and primitives. As R2ML supports Production Rules format[4], the JBossRules to R2ML translation of rule conditions part relies naturally. The problems appear in the translation of the actions part of a JBoss rule, which unfortunately is allowed to contain any Java valid code.

This implies even non-declarative approaches such as: `if...then` structures, variable declarations or cycling structures(i.e. `while`, `for`). We will not discuss here the weakness of the JBossRules actions syntax, but only the problems concerning our translation to R2ML language. To solve the problem of all unsuitable code for a rule action, R2ML will provide in its future versions, an `<r2ml:OpaqueExpression>` to encapsulate the code that don't find its semantic equivalent into R2ML actions.

Some of the problems we have encountered in the translation process from R2ML to JenaRules refers the arguments representation inside an operation call. JenaRules can represent, using its default RDF triples, only variables and data literals as operation arguments. JenaRules could not represent function calls as arguments for a operation call i.e.:

```
//returnVariable expression in JBossRules
price == $car.calculatePrice($vip.convertToEuro($premium),27000)
```

maps into following representation:

```
<r2ml:AttributionAtom r2ml:attributeID="price">
<r2ml:subject>
  <r2ml:ObjectVariable r2ml:name="car" r2ml:classID="Car"/>
</r2ml:subject>
<r2ml:dataValue>
  <r2ml:DataOperationTerm r2ml:operationID="calculatePrice">
    <r2ml:arguments>
      <r2ml:DataOperationTerm r2ml:operationID="convertToEuro">
        <r2ml:contextArgument>
          <r2ml:ObjectVariable r2ml:name="vip"
            r2ml:classID="VehicleInsurancePolicy"/>
        </r2ml:contextArgument>
        <r2ml:arguments>
          <r2ml:DataVariable r2ml:name="premium"
            r2ml:datatypeID="xs:decimal"/>
        </r2ml:arguments>
      </r2ml:DataOperationTerm>
      <r2ml:TypedLiteral r2ml:lexicalValue="27000"
        r2ml:datatypeID="xs:integer"/>
    </r2ml:arguments>
  </r2ml:DataOperationTerm></r2ml:dataValue>
</r2ml:AttributionAtom>
```

For the same reasons, Jena could not represent standard logic atoms $p(f(0),g(f(a)),2)$, that in R2ML are represented using `<r2ml:GenericAtom>`.

JenaRules language also finds its limitations in the impossibility to represent constructs such as `(car.createList(index).size==15)`, where the R2ML serialization implies an

```
<r2ml:ObjectOperationTerm> as content for
<r2ml:subject>.
```

4. Conclusion and future work

This paper aims to provide a brief, informal description of the JBossRules-JenaRules translation, based on the R2ML markup language.

Our work try to outline a way to interchange rules between UML modelers and ontology architects developers communities. Both of them use different tools and languages to achieve their purposes, but they make use of similar constructs like classes, properties or relations in order to designate same/similar domains.

Business rules aim to express rules in a platform independent syntax,therefore our work followed the principles initiated on rules inter-operability such as W3C (i.e. RIF[1]), OMG's standards and EU network of excellence REWERSE(i.e R2ML[5]).

References

- [1] H. Boley, M. Kifer (2007). *RIF Core Design*, W3C Working Draft, March 30, 2007 <http://www.w3.org/TR/rif-core/>
- [2] D. Reynolds, *JenaRules*, Jena User Conference, May 10-11, 2006, Bristol, UK., <http://jena.hp1.hp.com/juc2006/proceedings/reynolds/rules-slides.ppt>
- [3] *JBossRules*, <http://labs.jboss.com/jbosrules/docs>
- [4] OMG (2007). *Production Rule Representation Ver. 1.0*, March 5, 2007 <http://www.omg.org/docs/bmi/07-03-05.pdf>
- [5] G. Wagner, A. Giurca and S. Lukichev (2005). *R2ML: A General Approach for Marking up Rules*, Dagstuhl Seminar Proceedings 05371, In F. Bry, F. Fages, M. Marchiori, H. Ohlbach (Eds.) Principles and Practices of Semantic Web Reasoning, ISSN:1862-4405,<http://drops.dagstuhl.de/opus/volltexte/2006/479/pdf/05371.GiurcaAdrian.Paper.479.pdf>