Towards a Financial Service Rule-Based Implementation Using Jena and Jboss

Oana NICOLAE, Ion Mircea DIACONESCU, Adrian GIURCĂ, 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. A business rule approach provides the ability to respond and to quickly adapt to changes. It captures the essence of any business mechanism and it is suitable for all scenarios where rules serve to shape behavior and to guide business processes. These factors motivate us for choosing the business rule approach on implementing our Financial Service Use Case (i.e. UServ Product Derby 2005), which emulates a complete vehicle insurance service. The paper is a practical guide for those interested on the serialization and deployment of a Business Rules Model, from some Natural Language description to some target Rule-Based Platforms/Engines (i.e. JBossRules and JenaRules), via R2ML markup language. It is not in our intention to provide a comprehensive discussion about the R2ML to JBossRules or about R2ML to JenaRules translation, therefore the paper only captures the modeling and translation basic aspects.
Keywords: business rules, R2ML, RIF, rule interchange, JBossRules, JenaRules.
Math. Subject Classification 2000: 68T30, 68T35

1 Introduction

On the actual business market context, where constant change is the first characteristic, business rules are inevitable and the adaptability tends to become a necessity for all professional software products and services. Business rules are suitable for all cases where there is an acute need for guiding and influencing the behavior in a desired way. Therefore, business rules are used to employ different scenarios: from business games to financial services applications and one of the goals of advanced information system technologies is to provide a solid support for business rules and business processes.

Business rules, expressed in a declarative way, represent a basic component of the business IT architecture. There are actually a limited number of rules engines vendors around (i.e Jess, JRules from ILog). They are seriously competed by open-source rule engines like JBossRules [8] or JenaRules [9]. This

proves that business market communities, from both UML modelers and ontology architects sides, are re-orienting towards open-source business rules tools.

Another trend that regards the business rules adaptability as main characteristic, involves the concept of business rules technology standardization. Here we mention the efforts of OMG's MDA inter-operability standards, the W3C[1] initiative on rules interchange (i.e. RIF [1]) and last but not least, the EU network of excellence REWERSE (i.e. R2ML [12]).

This paper describes two rule-based implementations of UServ Product Derby 2005 Use Case [2] each of them addressing a different target platform and language area (i.e. Object Oriented Rule Languages via JBossRules[2] and Semantic Web Rule Languages via JenaRules[3]).

The paper covers the basic steps followed by I1 Rules Framework model (`http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=I1Framework`), in order to achieve the UServ 2005 Business Rules Model translation into JBossRules and JenaRules implementations, using R2ML as interchange language.

On Section 2 we motivate the decisions we took vis-a-vis our applications. Our work follows the principles of OMG's MDA [6] by considering three different levels of abstractions (CIM[4], PIM[5], PSM[6]) for business rules modeling. These levels will be further explained based on a concrete rule example on Section 3 which describe the architecture and the implementation of the application. Finally, Section 4 presents our conclusions based on the modeling and implementation stages of our work.

2 Motivation

JBoss Rules is an open-source, forward-chaining Production Rule System written entirely in Java language. It provides business logic (rules) and data (facts) separation which results in reusing the rules across applications and Service-Oriented Architectures. JBossRules employs a fast and efficient Rule Engine based on ReteOO, a descendant of Rete algorithm.

JBossRules features such as:

- The runtime provides dynamic assertion and remove of rules.
- Employs Conflict Resolutions like salience rule attribute and LIFO.
- Light and easy to understand syntax (i.e.DRL syntax) uses Java to express field constraints, functions and consequences.
- Easy to integrate with the mainstream JEE5 technologies.
- Collect complex decision-making logic and work with large data sets.

[1] World Wide Web Consortium - `http://www.w3.org`
[2] JBossRules - `http://www.jboss.com/products/rules`
[3] JenaRules - `http://jena.sourceforge.net/`
[4] CIM - Computation Independent Model - business level modeling.
[5] PIM - Platform Independent Model - independent level modeling.
[6] PSM - Platform Specific Model - implementation level modeling.

makes it a good candidate for implementing applications like UServ.

Jena is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF[7], RDFS[8], OWL[9], and SPARQL[10] Jena includes JenaRules, a rule-based inference engine. Implementing Userv with JenaRules allows the application to be integrated in other Semantic Web applications by sharing knowledge on the Web.

A tutorial[11] about UServ Product Derby rule-based implementation in JBossRules and JenaRules is available online. You can also test the application at `http://oxygen.informatik.tu-cottbus.de/userv/` in both JBossRules and JenaRules implementations.

To expose the translation process of the rules from plain English language to target platforms implementations, the application provides Rules Browser[12], where the rules, grouped in logical rulesets can be visualized. For each rule is available an URML (See [10] Chapter 1 and [3]) Model created with Strelka[13] [11], the corresponding R2ML Code, JenaRules implementation and JBossRules implementation. A view of the vocabulary model of the UServ Product Derby 2005 is also available.

3 Userv rule-based implementation

The use case expresses the rules in plain English language, and provides a scenario which emulates a complete vehicle insurance service. The final goal of the scenario is the computation of the UServ *annual premium* for a *vehicle insurance policy*, which belongs to an *eligible client*.

In order to achieve this purpose, UServ divides its business rules into business rules sets, each of them addressing different goals and contexts i.e.

- **Automobile Eligibility** - establishes the eligibility category for a car;
- **Driver Eligibility** - sets the eligibility category for a driver;
- **Eligibility Scoring** - determines the client's eligibility category based on a scoring system, by testing the risk ratings for: driver, car and client categories. If the client is eligible for vehicle insurance, then the annual premium is calculated;
- **Automobile Premiums** - calculates the car premium, based on model year, fabrication year, medical or uninsured motorist coverage;

[7] Resource Description Framework (RDF) - `http://http://www.w3.org/TR/rdf-syntax-grammar/`

[8] Resource Description Framework Schema (RDFS) - `http://http://www.w3.org/TR/rdf-schema/`

[9] Ontology Web Language (OWL) - `http://www.w3.org/2004/OWL/`

[10] Query Language for RDF (SPARQL) - `www.w3.org/TR/rdf-sparql-query/`

[11] UServ Tutorial - `http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/33`

[12] Rules Browser - `http://oxygen.informatik.tu-cottbus.de/userv/index.html`

[13] Strelka - `http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=Strelka`

- **Driver Premiums** - calculates the premium for every particular driver;
- **Automobile Discounts** - lowers the car premium with specific percents, if the car has or not airbags or alarm system;
- **Market Discounts** - which lowers the total premium (sum of car premium and driver premium) based on client segmentation (elite or preferred).

UServ business rules document address the OMG's Model Driven Architecture CIM level. Lets look together to the following rule AE_PTC04 from 2005 Product Derby Case Study:

Rule AE_PTC04: *If all of the following are true, then the car's Potential Theft Rating is moderate:*

- *car's price is between $20000 and $45000.*
- *car model is not on the list of "High Theft Probability Auto".*

From the plain English rule description, we can identify:

- The concepts referenced in the rule (i.e. car, car model).
- The concepts object/data types properties (i.e. potentialTheftRating, price, hasHighTheftProbability).
- The properties constraints (i.e. 20000 <= price, price <= 45000 and hasHighTheftprobability == false).

Therefore, the UServ Product Derby business rules needs a rule language, based on vocabulary, in order to be represented.

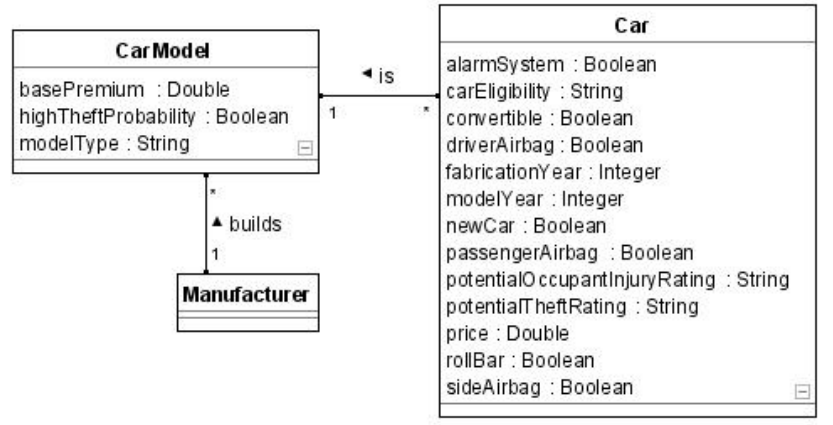

Fig. 1. An excerpt of the UServ Vocabulary

The I1 Rules Framework models, serializes and deploys rules according with the following steps:

- Rules are modeled visually using Strelka.
- Strelka tool generates the serialization of rules in R2ML language.
- Next step is to use the R2ML-to-JBossRules translator and R2ML-to-Jena translator in order to obtain JBossRules translation, respectively JenaRules implementation.
- Finally, we use JBossRules and JenaRules execution platforms to execute the obtained UServ business rules.

3.1 Visual rule modeling with Strelka

Strelka allows visual rule modeling on top of UML vocabularies, therefore the first step is to create an UML class diagram model in order to obtain the UServ vocabulary. At this stage, a PIM model is built. An excerpt of this model is depicted in Figure 1.

The complete model can be found at
`http://oxygen.informatik.tu-cottbus.de/userv/data/tmp78154785/pim.png`.

The visual representation of the rule AE_PTC04 is the following:

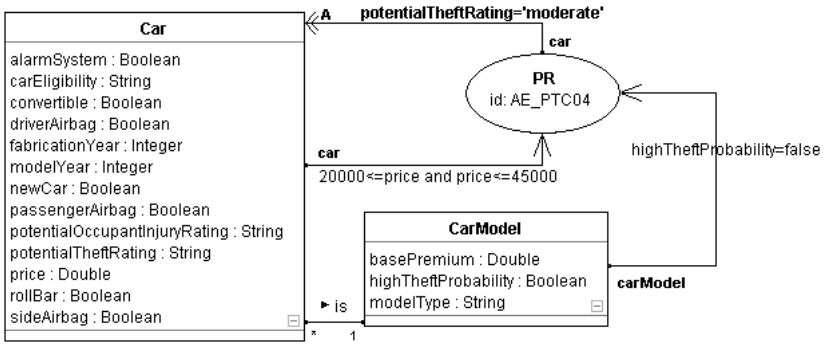

Fig. 2. UServ Product Derby - rule AE_PTC04.

Strelka is based on URML, an extension of UML class diagrams with additional concepts and graphical signs to represent rule specific elements. Visually

- A Production Rule is represented as an *ellipse* with an internal label `PR` and an additional rule identifier: `AE_PTC04`.
- The rule conditions are OCL filter expressions such as `(20000 <= price and price <= 45000)` representing price attribute constraints on car object or `(highTheftProbability = false)` representing an attribute constraint on carModel object. The conditions are represented by an incoming *condition arrow*.

– The rule action is: `potentialTheftRating = 'moderate'` and is represented by an outgoing, double headed, *action arrow*, together with the `A` letter symbol.

3.2 R2ML markup

The next step is to provide a serialization into the rule interchange format R2ML. Strelka tool is used to serialize rule models into the rule interchange format R2ML. Starting from this format by applying R2ML translators we obtain rules for different rule-based languages (i.e. JBoss Rules and Jena).

R2ML language is capable to express rules which don't require any conceptual changes in order to be implemented in Object Oriented Rule Systems such as: JBossRules, JRules (ILOG) or Semantic Web Rule Languages (i.e. JenaRules). In order to achieve this, R2ML has to comply Web naming concepts like (URI and Namespaces), datatype concepts of RDF and ontological distinction between objects and data.

An R2ML rule always refers to a vocabulary which can be R2ML own vocabulary or an imported one (RDF(S) and OWL). R2ML vocabulary is a serialization of an UML fragment of class diagrams. In our implementation every R2ML rule contains its own R2ML vocabulary i.e. elements from the vocabulary:
`xmlns:r2mlv="http://www.rewerse.net/I1/2006/R2ML/R2MLV"`.

Below is an excerpt from AE_PTC04 rule vocabulary:

```
<r2ml:RuleBase xmlns:r2ml="http://www.rewerse.net/I1/2006/R2ML"
xmlns:r2mlv="http://www.rewerse.net/I1/2006/R2ML/R2MLV"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://oxygen.informatik.tu-cottbus.de/R2ML/0.4/R2ML.xsd">
<r2mlv:Vocabulary>
 <r2mlv:Class r2mlv:ID="Car">
  <r2mlv:Attribute r2mlv:ID="price">
   <r2mlv:range><r2mlv:Datatype r2mlv:ID="xs:integer"/></r2mlv:range>
  </r2mlv:Attribute>
  <r2mlv:Attribute r2mlv:ID="potentialTheftRating">
   <r2mlv:range><r2mlv:Datatype r2mlv:ID="xs:sting"/></r2mlv:range>
  </r2mlv:Attribute>
 </r2mlv:Class>
</r2mlv:Vocabulary>
 <!-- ... -->
</r2mlv:RuleBase>
```

An R2ML *Production Rule* has *conditions*, *an action* and *a post-condition*. In R2ML framework the conditions and the post-condition of a production rule are Logical Formulas (i.e. the content of `r2ml:conditions` and `r2ml:postcondition` role elements), which corresponds to a general first order formula such as quantified formula (i.e. existentially quantified or universally quantified).

R2ML rules also have an optional attribute (e.g. `r2ml:ruleID = "AE_PTC04"`) which unique identifies a rule in a Production Rule Set.

The `r2ml:conditions` attribute is used to capture the R2ML conditions. The `r2ml:ObjectClassificationAtom` expresses the `instanceOf` relationship between objects and classes.

Any R2ML object classification atom consists into a mandatory attribute `r2ml:classID="Car"` (xs:QName) and an object term as argument: i.e. `r2ml:ObjectVariable r2ml:name="car"`.

```
1.<r2ml:ObjectClassificationAtom r2ml:classID="Car">
2. <r2ml:ObjectVariable r2ml:name="car"/>
3.</r2ml:ObjectClassificationAtom>
```

The markup of the condition `(20000 <= price)` is the following:

```
4.<r2ml:DatatypePredicateAtom r2ml:datatypePredicateID="swrlb:lessThanOrEqual">
5. <r2ml:dataArguments>
6.  <r2ml:TypedLiteral r2ml:datatypeID="xs:integer" r2ml:lexicalValue="20000"/>
7.  <r2ml:AttributeFunctionTerm r2ml:attributeID="price">
8.   <r2ml:contextArgument>
9.    <r2ml:ObjectVariable r2ml:name="car" r2ml:classID="Car"/>
10.  </r2ml:contextArgument>
11.  </r2ml:AttributeFunctionTerm>
12. </r2ml:dataArguments>
13.</r2ml:DatatypePredicateAtom>
```

The `(<=)` operation corresponds to the value of the `r2ml:datatypePredicate ID="swrlb:lessThanOrEqual"` attribute.

The `r2ml:attributeID` of the `r2ml:AttributeFunctionTerm` is used to represent the `price` attribute.

R2ML actions are built according with the OMG PRR Specification [7]. The actions part of an R2ML rule is represented by `r2ml:producedAction` role element:

```
14.<r2ml:producedAction>
15. <r2ml:AssignActionExpression r2ml:propertyID="potentialTheftRating">
16.  <r2ml:contextArgument>
17.   <r2ml:ObjectVariable r2ml:name="car" r2ml:classID="Car"/>
18.  </r2ml:contextArgument>
19.  <r2ml:TypedLiteral r2ml:lexicalValue="moderate"
20.                    r2ml:datatypeID="xs:string"/>
21. </r2ml:AssignActionExpression>
22.</r2ml:producedAction>
```

The `r2ml:AssignActionExpression` is used in order to markup the setting of the value `moderate` for the `r2ml:propertyID="potentialTheftRating"` attribute.

When referring to the above R2ML code, further in this paper, we will use the numbered code lines notation.

3.3 UServ architecture approach

Our approach involves a Web Application which runs under the Tomcat Application Server and was designed using MyEclipse. The Web Application exposes two users interfaces, one for JBossRules implementation and the other one for JenaRules implementation. These users interfaces consist in fact in one HTML pattern which collects the users inputs.

For each user, a new session is created. The sessions store information about the type of the module which is invoked (i.e. JBossRules module or JenaRules module). The users data are collected from the HTML forms and further processed and validated with the help of the JSP code. Then, the users inputs

are stored in the Knowledge Base(s) of the appropriate Rule Engine which is further invoked (JBossRules or JenaRules).

The UServ solution we propose is based on some modular software architecture that can be seen in the Figure 3 and implies at its basis, two rule-based implementations. Each of the implementations addresses different rule languages (Drools Rule Language (DRL) syntax for JBossRules and RDF triples for JenaRules) and distinct rule engine behaviors. JBossRules is a Rete-based forward chaining Production Rule Engine, meanwhile JenaRules (also a Rete-based Rule Engine) allows different types of reasoning behavior (i.e. forward-chaining, backward-chaining and hybrid).

The Jena rules provided in UServ Product Derby were executed using the `hybrid` mode of the inference engine, even they are implemented as `forward` rules. The reason the default rule sets use the hybrid mode is a performance trade-off trying to balance the better performance of forward reasoning with the cost of computing all possible answers when an application might only want a few.

Both JBoss and Jena modules are written in Java code and each of them embed an Inference Engine which apply reasoning on data stored in the Knowledge Base against the rules from the Rules Base (i.e. Production Memory in JBossRules).

When facts match the rules conditions, the rules are activated and eventually executed. The obtained results refer the client eligibility for auto insurance, and if the case, the annual premium for its vehicle insurance policy/policies.

During the Inference process, each of the rule implementations dynamically generates an execution log containing information such as which rules were executed, the rule sets they belong, what facts changes they implied an so on.

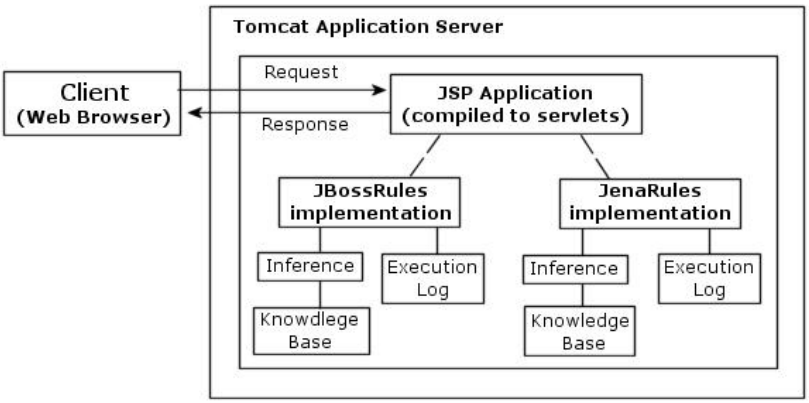

Fig. 3. UServ - Application Architecture

3.4 JBossRules experiences

The Drools/JBossRules system of JBoss provides a PSM for the representation of Production Rules.

JBossRules is designed to use *Java Beans* instances as facts. These facts represent the domain of the rules, meaning the rules vocabulary. Therefore, the input data from the HTML forms represent values for the properties of some JavaBeans instances.

Using vocabulary classes from R2ML, corresponding Java Beans are generated and all needed beans are imported into the DRL rule file through the import declarations, which are specified inside of the rules file (drl files). Assuming the namespace `xmlns:ex=http://www.drools.org/userv` and the qualified name `ex:Car`, this translates into JBoss declaration:
`import org.drools.userv.Car`.

JBoss rules are stored in text files usually having the `drl` extension.

The JBossRules representation for the "AE_PTC04" rule is the following:

```
import org.drools.userv.Car;
import org.drools.userv.CarModel;

rule "AE_PTC04"
    when
      $carModel:CarModel(highTheftProbability == false)
      $car:Car(carModel == $carModel, price >= 20000, price <= 45000)
    then
      $car.setPotentialTheftRating(PotentialTheftRating.MODERATE);
      modify($car);
    end
```

The JBoss central concept for conditions is the *Column*. It consists of zero or more*Field Constraints*, meaning the Column terms i.e.
`(highTheftProbability == false)` or `(price >= 20000)`. For example, the construct `price >= 20000` represents the translation of R2ML markup from lines 4-13. *Field Constraints* can be combined with a conjunctive logical operator (i.e. comma).

`$car` represents an instance of `Car` class. It is a bound variable constraint, named *declaration* in JBoss terminology. This instance give us the possibility to call attributes and functions of CarModel class in RHS part of the rule. It also represents the JBossRules translation from the R2ML markup (i.e. lines 1-3). Analogous explanations are for `$carModel`.

According with the R2MLtoJBoss mapping [5], the action expression from the above R2ML markup (see lines 14-22) translates into the following RHS setter:
`$car.setPotentialTheftRating(PotentialTheftRating.HIGH)`.

Additionally, a JBoss Rules specific structure: `modify($car)` is used in order to update the facts from working memory.

3.5 JenaRules experiences

JenaRules is a Semantic Web rule-based engine. RDF triples are used to represent the conditions and the conclusions or produced actions. An optional RDF Schema can be provided for facts in order to have a valid knowledge base.

JenaRules can use user defined built-ins in order to define some actions or to make some calculations or tests. Jena built-ins are Java classes with a predefined structure, and registered into a general built-ins register.

In our rules, disjunction can be also simulated by using many rules. For example, if we have in the condition *driver is from New York or driver is from Vancouver*, this splits in two rules, with the same conclusion but in the condition part of the first rule we have *driver is from New York* and in the second rule we have *driver is from Vancouver*.

Rules are divided into logical rule sets following the rule sets defined by the use case document. Each rule set is loaded into the memory the applied over the knowledge base.

The JenaRules syntax for the "AE_PTC04" rule is the following:

```
@prefix xs: http://www.w3.org/2001/XMLSchema
1. [AE_PTC04:
2.  (?car rdf:type Car)
3.  (?car carModel ?carModel)
4.  (?carModel highTheftProbability 'false'^^xs:boolean)
5.  (?car price ?price)
6.  ge(?price,20000)
7.  le(?price,45000)
8.  ->
9.  (?car potentialTheftRating 'moderate')]
```

In JenaRules the condition part of a rule is expressed by triples such as: `(?carModel highTheftProbability 'false'^^xs:boolean)` or built-ins (e.g. `le` or `ge`). Produced actions are also expressed by triple or built-ins. Built-ins can be the ones provided by the API or defined by user in the form of a Java class.

In the above example, the condition part contains triples which represent conditions (line 4) and triple which help to define the condition part(line 2,3,5). In order to express data types in JenaRules, XML Datatypes are used.

According with R2MLtoJena mapping (See [4]), triples representing `instanceOf` (see line 2) corresponds to `r2ml:ObjectClassificationAtom`. Triples representing conditions such as those from line 5 and 9 correspond to `r2ml:AttributionAtom`, those from line 3 to `r2ml:ReferencePropertyAtom` and those from line 6 and 7 to `r2ml:DatatypePredicateAtom`.

4 Conclusions

The paper traces the UServ 2005 Business Rules Model from plain English language descriptions based on core ontological concepts like classes and variables, to target rule systems like JBossRules and JenaRules, via the R2ML markup language.

Following the inter-operability principles of W3C and EU network of excellence REWERSE (i.e. RIF [1] and R2ML [12] respectively), our approach provides a platform independent syntax in order to further express the UServ production rules in two rule-based languages that came from distinct areas: Object-Oriented Rule Systems (i.e. JBossRules) and Semantic Web Rule Systems (i.e. JenaRules).

References

[1] **H. Boley, M. Kifer** (Eds.):- RIF Core Design, W3C Working Draft, March 30, 2007 http://www.w3.org/TR/rif-core/

[2] ***:- Business Rules Forum. UServ Product Derby 2005 Use Case http://www.businessrulesforum.com/2005_Product_Derby.pdf

[3] **S. Lukichev, G. Wagner**:- UML-Based Rule Modeling with Fujaba, In: H. Giese, B. Westfechtel (Eds.), Proceedings of the 4th International Fujaba Days 2006, University of Bayreuth, Germany. 28-30 September-2006. pp. 3135, http://oxygen.informatik.tu-cottbus.de/i1papers/LukichevWagnerFujabaDevDays2006.pdf

[4] **O. Nicolae, M. Diaconescu, A. Giurca, G.Wagner**:- Sharing rules between JBoss and Jena, Proc. of 9th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'2007), September 26-29, 2007, Timisoara, Romania.(to appear)

[5] **O. Nicolae, A. Giurca, G. Wagner**:- On Interchange between JBossRules and Jess, In: C. Badica, M. Paprzycki (Eds.) Proc. of 1st International Symposium on Intelligent and Distributed Computing (IDC'2007), October 18-20, 2007, Craiova, Romania. (to appear in Studies in Computational Intelligence, Springer)

[6] ***:- Model Driven Architecture (MDA), 2005, http://www.omg.org/mda/

[7] ***:- Production Rule Representation Ver. 1.0, March 5, 2007 http://www.omg.org/docs/bmi/07-03-05.pdf

[8] ***:- JBossRules, http://labs.jboss.com/jbossrules/docs

[9] **D. Reynolds**:- JenaRules, Jena User Conference, May 10-11, 2006, Bristol, UK., http://jena.hpl.hp.com/juc2006/proceedings/reynolds/rules-slides.ppt

[10] **G. Wagner, A. Giurca, S. Lukichev, G. Antoniou, C. V. Damasio, N. E. Fuchs**:- Language Improvements and Extensions. REWERSE IST 506779 Report I1-D8, April 2006, Munchen, Germany, http://rewerse.net/deliverables-restricted/i1-d8.pdf

[11] **G. Wagner, A. Giurca, S. Lukichev, G. Antoniou, M. Berndtsson**:- Strelka - A Visual Rule Modeling Tool, REWERSE IST 506779 Report I1-D4, April 2006, http://rewerse.net/deliverables-restricted/i1-d4.pdf

[12] **G. Wagner, A. Giurca and S. Lukichev**:- 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, 2005, http://drops.dagstuhl.de/opus/volltexte/2006/479/pdf/05371.GiurcaAdrian.Paper.479.pdf