Can URML model successfully Drools rules?

Emilian Pascalau and **Adrian Giurca** [1]

Abstract. The use of rules in business modeling is becoming more and more important, in applications requiring dynamic change of behavior. A number of rule languages and tools have been proposed to the software engineering community. However, there are not too many visual languages for rule modeling. The goal of this paper is to investigate the modeling capabilities of UML-based Rule Modeling Language (URML) with respect of Drools rules. We choose Drools because is the most important and well known open source rule platform. It is friendly to both developers and business users, offers a lot of functionality but does not provide a visual modeling environment for rules. The Single Item English Electronic Auction Use Case is used to illustrate the modeling capabilities. The paper concludes that URML rules can model the large part of Drools rules but improvements of the modeling language are necessary.

1 Introduction

Nowadays global information networks like Internet are the environment were business processes take place in automated way. A large part of e-commerce activities is devoted to B2B relationships. The natural way to describe behavior of such businesses is through business rules. However, actually there is no standard way for business rule definitions. Yet there are several rule platforms and rule languages: Drools [13] (also known as JBossRules), F-Logic, Jess, SWRL. The most important initiative in the process of developing a standard for rule interchange is Rule Interchange Format (RIF) [1]. Their main goal is to define a set of requirements and standards to be followed by any translator performing rule interchange between existing rules platforms.

A use case very well suited for such an environment and also very well suited to have his behavior modeled with business rules is automated negotiation. This is a general problem that comprises auctions also. In our paper we take as use case the Single Item English Electronic Auction [3], [9], [10]. We model its behavior using URML, UML-based rule modeling language (URML) [17, 18], a rule modeling extension of UML([12]).

Opposed to the approach taken by the authors in [15], where the vocabulary is presented as an ER model, we express the vocabulary as an UML model. In [5], [8] was argued that UML is a *de facto* standard for modeling vocabularies. Moreover, in the software engineering community UML class diagrams are widely used to express vocabularies. URML, as an extension of UML, it is well suited to capture rules on top of UML vocabularies.

The goal of the paper is to research the capabilities of URML to model rules that can be serialized to Drools.

Drools it is the most important and well known open source rule platform. It is friendly to both developers and business users, offers a lot of functionality but does not provide a visual modeling environment for rules.

It is well known that visual modeling is easier to be understood and to be remembered, therefore we claim that a *visual language for rule modeling is necessary*.

The authors of [17] argued, that rule modeling language should provide ways for representing rule expressions, in a manner easy to be understood by domain experts or by software engineers, who are usually used with UML modeling. URML extends UML meta-model with the concept of rule.

2 UML-based rule modeling language - URML

URML is developed by the REWERSE Working Group I1. Its main goal is to provide visual constructs for modeling rules and business processes. URML is close related to R2ML [16], [17] - a rule language for rule interchange.

URML is wanted to be a general approach for modeling rules in comparison with work introduced in [4]. According to [18], URML supports *derivation rules*, *production rules* and *reaction rules*. URML uses concepts such as *rule condition*, *rule conclusion*, *filters*, *actions* and *events*.

A rule condition is either a *ClassificationCondition*, a *RoleCondition* or *AssociationCondition*. The rule condition may contain a filter expression. For example the condition depicted in Figure 1 models the following logical conjunction:

$Proposal(\$bProposal) \wedge Proposal(\$sProposal) \wedge$
$\wedge product(\$bProposal) = product(\$sProposal) \wedge$
$\wedge price(\$bProposal) > 0$

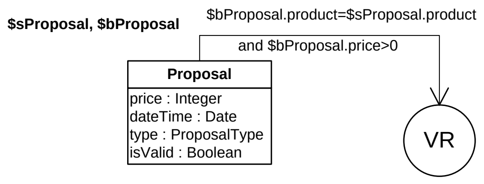


Figure 1. A rule condition

A filter is either an *OCLFilter* or an *OpaqueFilter*. *ClassificationCondition* refers to a UML Class, which is a condition classifier, and consists of an *ObjectVariable*, which is an instance of the Class; For example the expression

```
$bProposal.product == $sProposal.product
and $bProposal.price > 0
```

from Figure 1 is an OCL filter.

A rule conclusion is either a *RoleConclusion*, *ClassificationConclusion*, or *AttributionConclusion*, or *AssociationConclusion* or an

[1] Brandenburg University of Technology, Germany email: {pascalau, giurca}@tu-cottbus.de

Action. A more detailed description of these concepts is not possible because of the lack of space. The Figure 2 depicts an action corresponding to the following state change expression

$$isValid(\$bProposal)$$

i.e. the object property `isValid` is set to `true`.

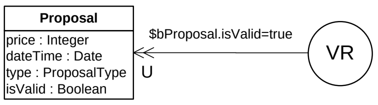
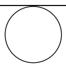
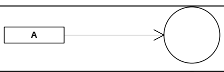
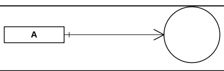
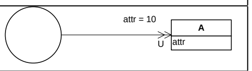


Figure 2. A rule action

Actions are used in production rules and in reaction rules. URML supports the following actions: *AssertAction*, *RetractAction*, *UpdateAction* and *InvokeAction*. They correspond to the OMG Production Rule Representation (PRR) [11].

The main advantage of URML is that it extends UML with only a few visual elements (see Table 1): *circles* for rules, *conditions arrows*, *conclusion arrows*, *action arrows*. Since Drools deals only with production rules, only production rules visual elements of URML are depicted in Table 1. A condition arrow can be negated and is represented as a crossed arrow at origin. Conclusion arrows refer to a class or an association. Action arrows are double-headed arrows referring either to a class (in the case of create, delete, assign or invoke action) or to an activity. Rule action arrow is annotated with an action type (*A* for Assert Action, *R* for Retract Action, *U* for Update Action, and *I* for Invoke Action). Variables are denoted in bold (such as `$bProposal`).

Table 1. URML Production Rule visual elements

Rule	
Rule condition arrow	
Negated rule condition arrow	
Rule action arrow	

3 Drools basics

Drools is an object oriented business rule management system (BRMS) and also a Rule Engine based on Charles Forgy's Rete algorithm [13].

The Drools architecture is based on three main components: *production memory* that stores the rules, *working memory* that stores the facts and the *inference engine*.

Drools development platform comes in two flavors: as an Eclipse plug-in Drools IDE and as web application Drools BRMS. The Drools IDE provides developers with an environment to edit and test rules in various formats, and integrate it deeply with their applications from within Eclipse. The IDE has a textual/graphical rule editor, a RuleFlow graphical editor, a domain specific language editor.

Our claim is that a visual rule editor is necessary and will enrich the Drools IDE with an important and more easy to use "feature". In opposition with the already built in rule text editor of the Drools IDE this will provide a visual way to model rules. Since Drools rules are written on top of Java Beans, visual modeling with URML is appropriate. The actual Drools IDE functionality and configuration is targeted mainly to developers and very technical users as authors argue in [13](*Chapter 5 - The (Eclipse based) Rule IDE*). The new feature will overcome this inconvenient and will allow software architects and engineers to easily describe the business rules in a visual way.

Rules are expressed in *Drools Rule Language* (DRL). It contains *package declaration*, *imports*, *globals*, *functions* and *rules*. Package declaration and usage are similar to those from Java. A *DRL package* defines a collection of rules and other related constructs. It represents a namespace, for the contained rules. Opposed to Java, the DRL package name is not related to files or folders in any way. *DRL import* statements work and have the same meaning as in Java. *Globals* as the name specifies are global variables used mainly to make application objects available to rules, for services or to provide data. According with [13], *DRL functions* provides a way to put semantic code in rule source file and are some how equivalent to helper classes from Java. A *DRL query* is simply a way to query the working memory for facts that match the conditions stated.

Drools manual [13] provides the following example:

```
rule "Approve if not rejected"
  salience -100
  agenda-group "approval"
    when
     not Rejection()
     p : Policy(approved == false,
     policyState:status)
     exists Driver(age > 25)
     Process(status == policyState)
    then
     log("APPROVED: due to no objections.");
     p.setApproved(true);
end
```

The above rule has an unique *name* (`"Approve if not rejected"`), optional *attributes* (such as `salience -100`), conditions identified by *when* (such as `exists Driver(age > 25)`) and actions introduced with *then* (such as `p.setApproved(true);`). The conditional part of a rule corresponds to a logical formula comprising zero or more *Conditional Elements*. The concept of *Pattern* is the main conditional element. *eval* is a Boolean expression evaluator. The action part contains a list of actions that are to be executed. Drools provides predefined logical actions such as: *insert*, *update*, *insertLogical*, *retract* but any valid Java code is also allowed.

4 Modeling Rules with URML

Automated negotiations e.g. electronic auctions are well suited to be modeled with rules. The past research was focused on defining and development of protocols and strategies to be used in multi agent systems that are to perform negotiations [10, 9, 3]. Auctions, a form of negotiation mechanism for electronic commerce, are also discussed in a number of papers such as [19, 20, 14].

English Auction is an important type of auction discussed in a wide range of papers such as [6, 7, 2], and we consider that the subject is far from being finished. The principles of Single Item English Auction are: (1) only one item is sold at a time; (2) bidding is open; (3) all participants bid against each other openly; (4)each successive bid must be higher than the old one; (5) the seller begins the auction;

(6) buyers bid against each other by raising the price, until only one willing buyer remains.

4.1 The Vocabulary

Our work will use a fragment of the vocabulary (see Figure 3) for automated negotiation similar with the one from [2].

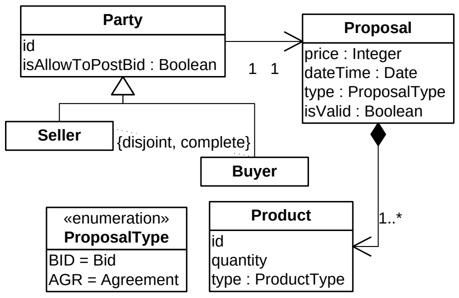


Figure 3. Negotiation Vocabulary

Proposals encapsulate data about price, date and time, and product in auction. They are exchanged between parties. A proposal is either a *Bid* or an *Agreement*. A bid is a commitment from a buyer to pay that price if the bid is declared to be a winning bid (proposal). An agreement is a proposal upon which all parties were agreed.

4.2 The Rules

The aim of this section is to model rules that automate the negotiation in Single Item English Auction.

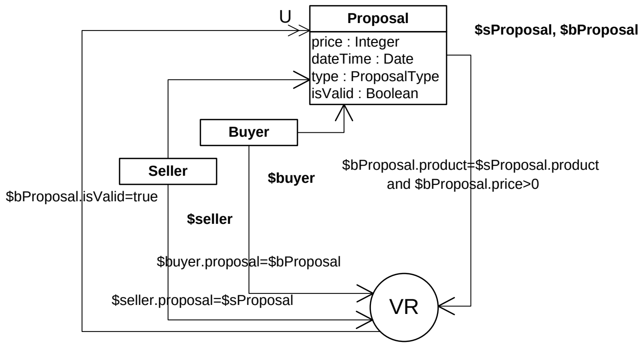


Figure 4. Validation Rule: *"A valid proposal is a proposal that is about the product from seller proposal and the submitted proposal price is greater than 0."*

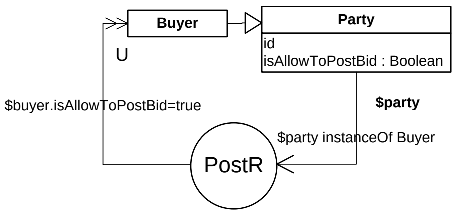
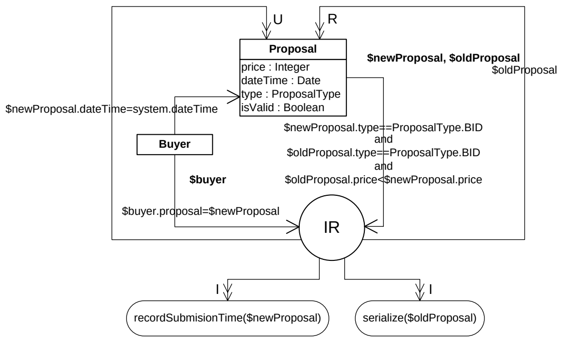


Figure 5. Posting Rule: *"Only buyers parties are allowed to post bids."*

In [3] rules are classified in taxonomies such as: *proposal validity*, *protocol enforcement*, *updating status and information of participants*, *agreement formation*, *termination*. These rules taxonomy



U
R
Proposal
price : Integer
dateTime : Date
type : ProposalType
isValid : Boolean
$newProposal, $oldProposal
$oldProposal
$newProposal.dateTime=system.dateTime
Buyer
$newProposal.type==ProposalType.BID
and
$oldProposal.type==ProposalType.BID
and
$oldProposal.price<$newProposal.price
$buyer
$buyer.proposal=$newProposal
IR
I
I
recordSubmisionTime($newProposal)
serialize($oldProposal)

Figure 6. Bid Improvement Rule: *"Each new bid must be an improvement over the last one. If the submitted bid is an improvement, then update the bid time."*

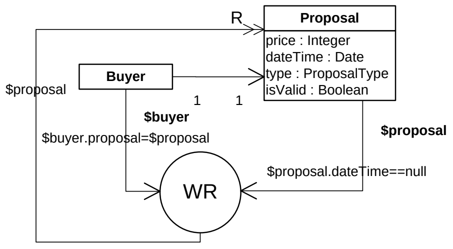


Figure 7. Withdraw Proposal Rule: *"Buyers proposals that are not best proposal have to be withdrawn."*

introduces a packaging model for the rules. Actually URML does not provide a package modeling therefore actually we consider rule diagrams in the same folder to be part of the same package.

To illustrate URML capabilities we model one rule from each package obtained from the ontology.

The URML representation of a *validation rule* is depicted in Figure 4. The rule uses the following vocabulary beans: `Buyer`, `Seller`, `Proposal` and `Product`.

We have a buyer (`$buyer`), a seller (`$seller`), two proposals (`$bProposal` and `$sProposal`): one for the buyer and one for the seller.

The arrow condition comes with variables (e.g. $\$bProposal$) and filter expressions `$bProposal` is buyer's proposal and is bound to the buyer's proposal. For this we use a condition arrow going from buyer to rule that says `$buyer.proposal=$bProposal`. The same works for the seller.

The action performs a setter call on the `isValid` property of the `$bProposal`.

The *posting rule* (Figure 5) determines when a party can post a proposal.

Improvement rules define the way bids are posted. In a Single Item English Auction each successive bid must be an improvement, therefore its price must be greater than the `$oldProposal` price. This is exactly what the rule from Figure 6 does.

Another *protocol enforcement* rule is shown in Figure 7. If the proposal $\$proposal$ belongs to a $\$buyer$ and it is not the *best proposal*(i.e. $dateTime$ is null) then withdraw $\$proposal$.

The Figure 8 depicts the model of a *display rule*. This rule refers to the package *updating status and information of participants* and specifies which party can see a new proposal. In Single Item English Auction every new proposal is known to all parties involved in auction.

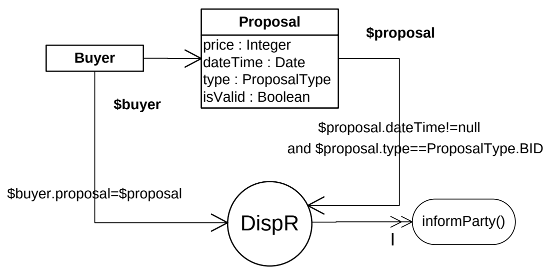


Figure 8. Display Proposal Rule: *"If a new proposal has been posted into the system then all parties must be informed about this."*

Termination rules define conditions when an auction is terminated. The Figure 9 depicts an URML model of such rule.

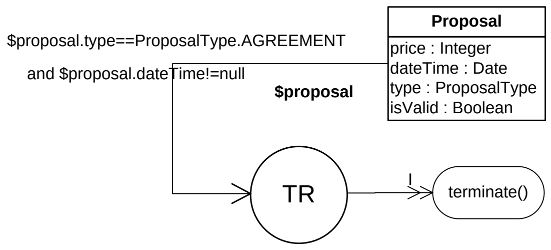


Figure 9. Termination Rule: *"If type of the current proposal from working memory is $AGREEMENT$ and $dateTime$ of proposal is not null then the auction can be terminated."*

5 URML Rules as Drools Rules

This Section presents how a rule modeled with URML can be serialized to Drools. Consider the improvement rule (see Figure 6). The below code is the Drools DRL:

```
package org.ruleapp.rules.improvement;

import org.ruleapp.vocabulary.Proposal;
import org.ruleapp.vocabulary.Buyer;
import org.ruleapp.vocabulary.ProposalType;

rule "IR"
  when
   $oldProposal:Proposal(
    type == ProposalType.BID,
    $oPrice:price
   )
   $newProposal:Proposal(
    type == ProposalType.BID,
    $nPrice:price,
    $nPrice > $oPrice
   )
  then
   recordSubmisionTime($newProposal);
   update($newProposal);
   serialize($oldProposal);
   retract($oldProposal);
end

function recordSubmisionTime(Proposal $p){
 //...
}

function serialize(Proposal $p){
 //...
}
```

The rule is part of a specific package i.e. `org.ruleapp.rules.improvement` encoded in the corresponding DRL package declaration.

The rule (as seen in the URML model) uses the classes *Proposal*, *Buyer* and *ProposalType* therefore all of them should be available to the engine (as Java Beans). We make them available by generating the appropriate *import* commands.

The name of the Drools rule (`IR`) is the same with the name from the visual model.

The filter condition

```
$newProposal.type==ProposalType.BID and
$oldProposal.type==ProposalType.BID and
$oldProposal.price < $newProposal.price
```

generates the conditions (i.e. the `when` part) in the Drools rule. While parts such as `$newProposal.type==ProposalType.BID` have immediate translation, the part `$oldProposal.price < $newProposal.price` requires the generation of new variables (`$nPrice` and `$oPrice`) before the condition evaluation. Readers may notice that this filter can be also implemented by means of an `eval()` call but then the rule become less declarative.

Our actions are:

1. an invoke action (`I`) corresponding to the function call `recordSubmisionTime($newProposal);`
2. an update action which normally translates into a Drools standard action `update`
3. another invoke action (i.e. `serialize($oldProposal);`) and
4. a retract action (`R`) generating the code `retract($oldProposal);`

6 Future Work

While the translation of all of these actions to Drools code is straightforward by using DRL functions one major disadvantage of the actual URML language is that it does not offer a way to specify the order of actions in the rule action part. For example, looking to the rule diagram from Figure 6 it is not clear both for a human expert and a machine in which order the depicted actions have to be performed.

The translation, presented in this paper was done manually and is intended as example for a potential implementation, that would have to perform it automatically.

Our proposal is to extend the URML metamodel by allowing *sequence actions* i.e an ordered sequence of standard actions as in the Figure 10 .

Other open issues are: (1) Drools provides DRL queries while URML does not provide any visual construct modeling that; (2) URML does not provide any annotations to encode various DRL rules attributes.

Drools complex constructs offering integration with databases such as `collect` and `accumulate` are not yet supported by the visual language.

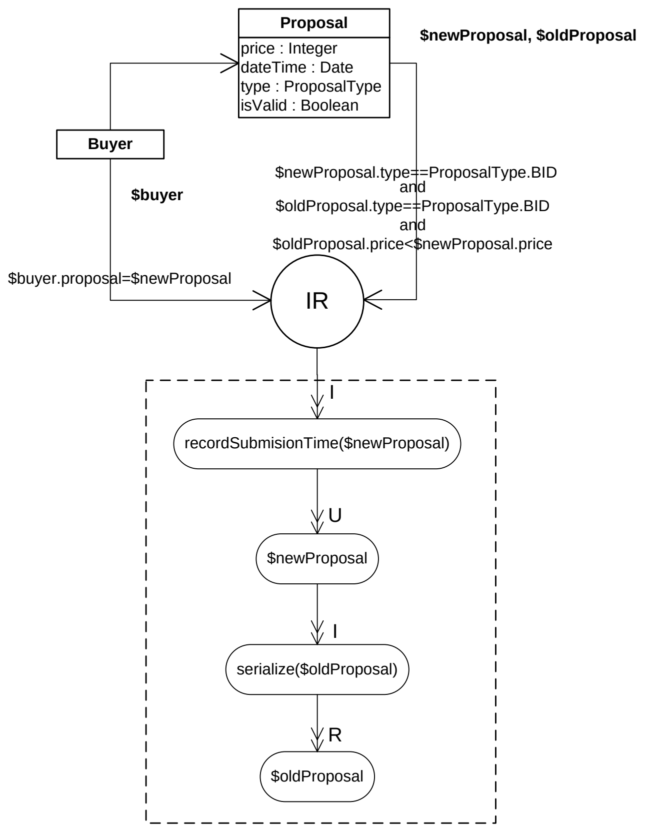

Proposal
price : Integer
dateTime : Date
type : ProposalType
isValid : Boolean
$newProposal, $oldProposal
Buyer
$buyer
$newProposal.type==ProposalType.BID and $oldProposal.type==ProposalType.BID and $oldProposal.price<$newProposal.price
$buyer.proposal=$newProposal
IR
I
recordSubmisionTime($newProposal)
U
$newProposal
I
serialize($oldProposal)
R
$oldProposal

Figure 10. "The New Bid Improvement Rule"

Finally user-defined actions encoded by plain Java code are not yet supported. Our future work will investigate the need of an extension of the visual language that allows UML opaque expressions to encode these actions.

A potential URML implementation for Drools rules must extends the actual Eclipse IDE by allowing at least *UML class diagrams*, *rule diagrams* and *rule packages*.

REFERENCES

[1] RIF Basic Logic Dialect. `http://www.w3.org/2005/rules/wiki/BLD`, October 2007.

[2] Costin Badica, Adrian Giurca, and Gerd Wagner, 'Using Rules and R2ML for Modeling Negotiation Mechanisms in E-Commerce Agent Systems', in *Proceedings of the 2nd International Conference on Trends in Enterprise Application Architecture, TEAA2006*, eds., Dirk Draheim and Gerald Weber, volume 4473 of *Lecture Notes in Computer Science*, pp. 84 – 99. Springer, (November 2006). `http://dx.doi.org/10.1007/978-3-540-75912-6_7`.

[3] Claudio Bartolini, Chris Preist, and Nicholas R. Jennings, 'A Generic Framework for Automated Negotiation', Technical report, HP Labs, (January 2002). `http://www.hpl.hp.com/techreports/2002/HPL-2002-2.pdf`.

[4] Saartje Brockmans, Peter Haase, Pascal Hitzler, and Rudi Studer, 'A Metamodel and UML Profile for Rule-Extended OWL DL Ontologies', in *Proceedings of 3rd European Semantic Web Conference, ESWC 2006, Budva, Montenegro*, volume 4011 of *Lecture Notes in Computer Science*, pp. 303 – 316. Springer Berlin / Heidelberg, (June 2006). `http://dx.doi.org/10.1007/11762256_24`.

[5] Stephen Cranefield and Martin Purvis, 'UML as an Ontology Modelling Language', in *Proceedings IJCAI-99 Workshop on Intelligent Information Integration*, (1999). `http://hcs.science.uva.nl/usr/richard/workshops/ijcai99/UML_Ontology_Modelling.pdf`.

[6] Esther David, Rina Azoulay-Schwartz, and Sarit Kraus, 'An English Auction Protocol for Multi-attribute Items', in *Proceedings of the Workshop on Agent Mediated Electronic Commerce on Agent-Mediated Electronic Commerce IV, Designing Mechanisms and Systems*, volume 2531 of *Lecture Notes in Computer Science*, pp. 361 – 378. Springer Berlin / Heidelberg, (2002). `http://dx.doi.org/10.1007/3-540-36378-5_4`.

[7] Esther David, Alex Rogers, Jeremy Schiff, Sarit Kraus, and Nicholas R. Jennings, 'Optimal Design Of English Auctions With Discrete Bid Levels', in *Proceedings of the 6th ACM conference on Electronic commerce, Vancouver, BC, Canada*, pp. 98 – 107. ACM New York, NY, USA, (2005).

[8] Giancarlo Guizzardi, Gerd Wagner, and Heinrich Herre, 'On the Foundations of UML as an Ontology Representation Language', in *Proceedings of 14th International Conference on Engineering Knowledgein the Age of the Semantic Web EKAW 2004*, eds., Enrico Motta, Nigel Shadbolt, Arthur Stutt, and Nicholas Gibbins, volume 3257 of *Lecture Notes in Computer Science*, pp. 47 – 62. Springer Berlin / Heidelberg, (5-8 October 2004). `http://www.loa-cnr.it/Guizzardi/EKAW.pdf`.

[9] Nicholas R. Jennings, Peyman Faratin, A. R. Lomuscio, Simon Parsons, Michael Wooldridge, and Carles Sierra, 'Automated Negotiation: Prospects, Methods and Challenges', *Group Decision and Negotiation*, **10**(2), 199 – 215, (March 2001). `http://dx.doi.org/10.1023/A:1008746126376`.

[10] Sarit Kraus, 'Negotiation and cooperation in multi-agent environments', *Special issue on economic principles of multi-agent systems*, **94**(1-2), 79 – 98, (1997). `http://iskp.csd.auth.gr/mtpx/agents/material/kraus97negotiation.pdf`.

[11] OMG. Production rule representation (prr), beta 1. `http://www.omg.org/docs/dtc/07-11-04.pdf`, November 2007.

[12] Object Management Group (OMG). UML 2.0 Superstructure Specification. `http://www.omg.org/cgi-bin/doc?ptc/2003-08-02`, August 2002.

[13] Mark Proctor, Michael Neale, Michael Frandsen, Sam Griffith Jr., Edson Tirelli, Fernando Meyer, and Kris Verlaenen. Drools 4.0.5. `http://downloads.jboss.com/drools/docs/4.0.5.19064.GA/html_single/index.html`, January 2008.

[14] Daniel Rolli and Andreas Eberhart, 'An Auction Reference Model for Describing and Running Auctions', *Wirtschaftsinformatik 2005*, 289 – 308, (2005). `http://dx.doi.org/10.1007/3-7908-1624-8_16`.

[15] Valentina Tamma, Michael Wooldridge, Ian Blacoe, and Ian Dickinson, 'An Ontology Based Approach to Automated Negotiation.', in *Proceedings of the Workshop on Ontologies in Agent Systems, Bologna, Italy, AMEC02*, volume 2531 of *Lecture Notes in Computer Science*, pp. 317 – 334. Springer Berlin / Heidelberg, (2002). `http://dx.doi.org/10.1007/3-540-36378-5_14`.

[16] Gerd Wagner, Adrian Giurca, and Sergey Lukichev, 'A General Markup Framework for Integrity and Derivation Rules', in *Principles and Practices of Semantic Web Reasoning*, eds., François Bry, François Fages, Massimo Marchiori, and Hans-Jürgen Ohlbach, number 05371 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, (2005). Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. `http://drops.dagstuhl.de/opus/volltexte/2006/479/pdf/05371.GiurcaAdrian.Paper.479.pdf`.

[17] Gerd Wagner, Adrian Giurca, and Sergey Lukichev, 'A Usable Interchange Format for Rich Syntax Rules. Integrating OCL, RuleML and SWRL', in *Proceedings of Reasoning on the Web 2006, Edinburgh, Scotland*, (May 2006). `http://www.aifb.uni-karlsruhe.de/WBS/phi/RoW06/procs/wagner.pdf`.

[18] Gerd Wagner, Adrian Giurca, Sergey Lukichev, Grigoris Antoniou, Carlos Viegas Damasio, and Norbert E. Fuchs, 'Language Improvements and Extensions', Technical Report I1-D8, REWERSE, (April 2006). `http://rewerse.net/deliverables-restricted/i1-d8.pdf`.

[19] Peter R. Wurman, Michael P. Wellman, and William E. Walsh, 'A Parametrization of the Auction Design Space', *Games and Economic Behavior*, **35**, 304 – 338, (2001). `http://www4.ncsu.edu/~wurman/Papers/Wurman-GEB-00.pdf`.

[20] Peter R. Wurman, Michael P. Wellman, and William E. Walsh, 'Specifying Rules for Electronic Auctions', *AI Magazine*, **23**, (2002). `http://www4.ncsu.edu/~wurman/Papers/AI-Mag-WWW.pdf`.