Preface

Rules are becoming increasingly important in business modeling and requirements engineering, and as a high level programming paradigm especially in the engineering of e-business applications and of Semantic Web applications. In each of these the fields different rule languages and tools are being used.

Rules are used in applications to model and manage some parts of the application business logic. They are best used in applications with a dynamic business logic i.e. applications where changes in the business logic are frequently and they need to be immediately reflected in the application behavior.

Applications in domains such as insurance (for example, insurance rating), financial services (loans, claims routing and management, fraud detection), government (tax calculations), telecom customer (care and billing), e-commerce (personalizing the user's experience, recommender systems, auctions), and so on benefit greatly from using rule engines.

This volume presents some results of researchers in the rules community. The maturity of the research in the discipline and the recent development in commercial/industrial rule applications provided an opportunity to produce this workshop.

This workshop aims for contributions contributions that address theoretic foundations, practical techniques, empirical studies, experience, and lessons learned related to

- Applications and Use Cases using languages for Enterprise Rules such as JBoss Rules and Oracle Business Rules (including Rule-based Auctions, Classification rules, Fuzzy Rules, Association Rules in Data Mining)
- Artificial Intelligence Rules and Rule Systems (such as F-Logic and Jess)
- Best Practices in Business Rules Applications
- Combining rules and ontologies
- Implemented tools and systems
- Languages for Rule Interchange (such as RIF and R2ML)
- Modeling of Business Rules (including Production Rules and ECA Rules)
- Rule base Visualization, Verbalization, Validation, Verification and Exception Handling.
- Rule-based modeling of mechanisms, policies, strategies and contracts.
- Rule Engines Architecture
- Rules and Web Services Integration
- Web Rules and Semantic Web Applications

RuleApps'2008, Program Committee Chairs

Conference Organization

Programme Chairs

Adrian Giurca
Anastasia Analyti
Gerd Wagner

Programme Committee

Grigoris Antoniou
Costin Badica
Nick Bassiliades
Philippe Bonnard
Carlos Viegas Damasio
Vladan Devedzic
Jens Dietrich
Dragan Gasevic
Antoni Ligeza
Grzegorz Nalepa
Viorel Negru
Paula-Lavinia Patranjan
Mark Proctor
Dave Reynolds
Kuldar Taveter
Laurentiu Vasiliu
Sanja Vranes

External Reviewers

Francisco Azevedo
Mirel Cosulschi
Mihai Gabroveanu
Elvira Popescu

Table of Contents

Extracting Semantic Annotations from Moodle Data

Mihai Gabroveanu[1] and **Ion-Mircea Diaconescu**[2]

Abstract. The purpose of this paper is to provide a solution which allows automatic reasoning processes over Moodle activities logs, in order to obtain user-personalized recommendations. Activities logs are mined for association rules, which are the translated into Jena Rules. The information is then used by specific learning rules to create recommendations for specific users. Using this technique, additional information is obtained starting from activities database.
`Keywords:` e-Learning, association rules, Jena, RDF(S), ERDF.

1 Introduction

Nowadays, e-Learning systems are widely used, specially in schools, colleges and universities but not only. More and more corporations involve continuous learning in their management systems. A number of e-Learning systems such as Moodle, Sakai, ATutor, CLIX are available either open source or commercial either as a standalone applications or online learning platforms (such as Microsoft Learning Manager, BlackBoard).

All these systems accumulate a large amount of data suitable for analyzing the users behavior using data mining technics. The goal of extracted information is to improve the educational process.

This work describes an extension of Moodle e-Learning system, which extracts semantic metadata helpful in delivery of user personalized content. In a previous work ([6] and [13]) we improved Moodle by adding rules and semantics to enrich the reports generation. In this work we follow the idea that, additionally to the standard information that users can see, it is possible to obtain supplementary information indirectly available (i.e. obtained by processing data stored in Moodle activity logs). Using this Moodle module, users are informed about some specific changes or are advised to do some actions. For example a student can be advised to read some specific resources in order to obtain necessarily skills for a specific test. This module is user-based, meaning that all information and suggestions are made depending of which user authenticates to the system. For example may be unnecessarily to suggest for some users to follow a specific course, since they already followed that course, but for others users this can be a valid option. In order to create the module we use Weka to extract association rules from Moodle activities logs and Jena Rules to infer additional information.

The paper is organized as follows: (1) in the first part explain the steps followed to extract association rules from Moodle activities logs; (2) the second part explain the mapping from association rules to Jena Rules and discuss an improvement for Moodle activities logs; (3) finally, we describe the architecture of the module implementation.

[1] Dept. of Computer Science, University of Craiova, Romania, e-mail: mihaiug@central.ucv.ro
[2] Brandenburg University of Technology, Germany, e-mail: M.Diaconescu@tu-cottbus.de

2 Mining information from Moodle activities logs

E-Learning systems provides databases where information about students profile, courses, academic results and performed activities (reading, writing, taking tests) are stored. A huge quantity of data is collected and can be very difficult to perform a manually analyze over it. Data mining provides technics and algorithms useful to perform an automatically analyze over activities logs databases. Instructors use available data to improve the courses quality or to build recommendations for system users.

The process of discovering association rules is an important task in data mining. An association rule provides a relationship among different attributes. Our Moodle module use algorithms for mining association rules in order to identify possible relations between courses, resources, student activities. Particularly, a selection process regarding the information we are interested to mine is performed. This allows us to obtain only specific association rules which is relevant for our needs.

2.1 Basic Knowledge on Association rules

In this subsection we presents a basic introduction of concepts related to association rules and the mining process.

The initial problem of mining association rules was formulated by Agrawal in [1] and is called the *market-basket problem*.

Considering $\mathcal{T}$ to be a non-empty data table containing transactions, an *association rule* is an expression with the following form: $A \Rightarrow B$. Formally speaking, this means that transactions including A will include B as well, with a high probability. A and B are called *the antecedent*, respectively *the consequent* of the rule.

The quality of an association rule is expressed by several measures. Two of them, namely the *support* and the *confidence* are essential [1]:

- the *support* of $A \Rightarrow B$ is defined as *the percentage of transactions in $\mathcal{T}$ that contain both A and B.*
- the *confidence* of $A \Rightarrow B$ is defined as *the percentage of transactions in $\mathcal{T}$ containing A which also contain B.*

Example 1 *Table 1 contains courses followed by students. We see that student having ID 1 followed Web Technologies (WT), Web Applications (WA) and Web Documents (WD) courses, the student having ID 2 followed Web Applications (WA) and E-Business Technologies (EBT) courses, etc.*

An example of association rule is $WT \Rightarrow WD$. This express that some of the students who followed Web Technologies (WT) course, also followed Web Documents (WD) course. The support of this association rule is calculated as:

$$supp(WT \Rightarrow WD) = \frac{|\{1,3,5\}|}{|\mathcal{T}|} = \frac{3}{6} = 0.50$$

StudentID	List of courses
1	WT, WA, WD
2	WA, EBT
3	WT, WD, EBT
4	WA, WD, EBT
5	WT, WD
6	WT, EBT

Table 1. The list of courses

expressing that 50% of students followed both the Web Technologies (WT) course and the Web Documents (WD) course.

The confidence can be calculated as:

$$conf(WT \Rightarrow WD) = \frac{|\{1,3,5\}|}{|\{1,3,4,5\}|} = \frac{3}{4} = 0.75$$

and it express that: from all students who follow the Web Technologies (WT) course, 75% of them also followed the Web Documents (WD) course.

Rules having support and confidence greater than an user-specified minimum support ($minsup$) and respectively a minimum confidence ($minconf$) are named *strong association rules*.

In this work the goal is to obtain only strong association rules inferring new information relevant in our context.

To extract strong association rules many algorithms were proposed. The most popular are: Apriori [2], DHP [12], PARTITION [14], DIC [5].

2.2 Mining Logs to extract useful data

The Knowledge Discovery [10] consist in the following steps: collecting data, preprocessing data, applying the data mining algorithms and post-processing. The mining association rule process in e-Learning systems [9] follows some steps:

- *Collecting data*. The Moodle database store detailed logs with all activities that users performs.
- *Data pre-processing*. Typical tasks are performed in this phase: data selection, derivation of new attributes and selection of some attributes (new attributes are created starting from the existing ones and only a subset of relevant attributes are finally chosen), transforming the data format (to a format required by the used data mining algorithms or framework).
- *Applying the mining algorithms*. In this phase we need:
  - to choose specific association rule mining algorithm;
  - to configure the parameters of the algorithm (such as support and confidence threshold, $minsup$ and $minconf$);
  - to identify table(s) or data file are used in the mining process;
  - and to specify some other restrictions, such as the maximum number of items and what specific attributes can be present in the antecedent or consequent of the discovered rules.
- *Data post-processing*. Strong association rules which are obtained are represented in a comprehensible format.

Our interest is to extract association rules such as:

- *82% of the students who followed Web Technologies (WT) course, also followed Web Application (WA) course.*
- *70% of the students that solve home-works from Web Technologies (WT) course pass the WA exam.*
- *74% of the students that read resource A and B from course E-Business Technologies (EBT) read also resource C.*

In order to extract association rules from Moodle logs we use an existing data mining tool, namely Weka, which implements several algorithms for extracting association rules. For our purpose, we choose to we use Apriori [2], but also other algorithms can be taken into consideration. The mined models will be exported into PMML[3] (Predictive Model Markup Language). The Predictive Model Markup Language (PMML) is an XML-based language which provides a way for applications to define statistical and data mining models and to share models between PMML compliant applications.

Example 2 *Let consider the relational data table obtained from Moodle logs (Table 2) containing courses followed by students. This data table is obtained after pre-processing step and corresponding to transactional data table presented in Table 1.*

StudentID	CourseID	StudentID	CourseID
1	WT	4	WA
1	WA	4	WD
1	WD	4	EBT
2	WA	5	WT
2	EBT	5	WD
3	WT	6	WT
3	WD	6	EBT
3	EBT		

Table 2. Excerpt from Moodle data

Executing the Apriori algorithm implemented in Weka over the data depicted below and providing a minimum support value (0.4) and a minimum confidence value (0.5) as parameters we obtain two association rules $WT \Rightarrow WD$ (supp=0.50, conf=0.75) and $WD \Rightarrow WT$ (supp=0.50, conf=0.75). Our module translate rules in the PMML form. By example, association rules obtained after post-processing step is depicted below:

```
<?xml version="1.0" encoding="UTF-8"?>
<PMML xmlns="http://www.dmg.org/PMML-3_1">
 <DataDictionary numberOfFields="2">
  <DataField dataType="integer" name="SudentID"
     optype="continuous">
   <Extension extender="weka" name="storageType"
     value="numeric"/>
  </DataField>
  <DataField dataType="string" name="CourseID"
    optype="categorical">
   <Extension extender="weka" name="storageType"
     value="string"/>
   <Value property="valid" value="EBT"/>
   <Value property="valid" value="WA"/>
   <Value property="valid" value="WD"/>
   <Value property="valid" value="WT"/>
  </DataField>
 </DataDictionary>
 <AssociationModel algorithmName="Apriori"
   functionName="associationRules"
   minimumConfidence="0.5"
   minimumSupport="0.4" modelName="Sudents_Courses"
   numberOfItems="2" numberOfItemsets="2"
   numberOfRules="2" numberOfTransactions="6">
  <MiningSchema>
   <MiningField name="SudentID" usageType="group"/>
   <MiningField name="CourseID" usageType="active"/>
  </MiningSchema>
  <Item id="1" value="WD"/>
```

[3] PMML - http://www.dmg.org/pmml-v3-1.html

```
  <Item id="2" value="WT"/>
  <Itemset id="1" numberOfItems="1" support="0.667">
   <ItemRef itemRef="1"/>
  </Itemset>
  <Itemset id="2" numberOfItems="1" support="0.667">
   <ItemRef itemRef="2"/>
  </Itemset>
  <AssociationRule id="1" antecedent="1"
    consequent="2" support="0.5" confidence="0.75"/>
  <AssociationRule id="2" antecedent="2"
    consequent="1" support="0.5" confidence="0.75"/>
 </AssociationModel>
</PMML>
```

An advantage os using this representation (PMML) is that it is XML based, it has a schema and it is easy to translate to another representation types, XML-based or not. Our solution use an XSLT transformation to map association rules from PMML representation to Jena rules syntax.

3 Generate Recommendations in Moodle

In this section, we describe a translation from association rules, extracted from Moodle activities logs based on *support* and *confidence* factors, to Jena rules. Using such rules, complex reports and recommendations on the page of each authenticated user are created.

3.1 Brief introduction to Jena Rules

Jena is a framework which allows reasoning over RDF(S) ([8], [4]). It use a triple based syntax for rules (e.g. `(?x rdf:type moodle:Student)`), and built-ins to represent user defined operations (actions). Atoms are represented by RDF nodes, and the used syntax for representing URI's, variables, blank nodes and literals (plain or typed) is based on SPARQL. In Jena rules, components of a triple are: (1) the *subject* - is the first node, and it can be variable, URI reference or blank node; (2) the *predicate* - the second node of the triple, is expressed by using a variable or an URI reference; (3) the *object* - the last node of the triple, can be a variable, an URI reference, a blank node or a literal.

Jena rules offers support for a form of `negation-as-failure`, expressed by using the `noValue` built-in, who's parameters are the nodes of the triple (e.g. `noValue(?x moodle:passedExam moodle:WT)`). Conjunction is used by default and disjunction is not supported. Three types of rules are supported by Jena Rules engines, namely *forward*, *backward* and *hybrid* (forward rules having backward rules in the head). This paper describe a Moodle improvement dealing with forward rules executed by a RETE [7] forward engine.

3.2 Mapping association rules to Jena Rules

In order to generate reports and recommendations, we use Moodle activities logs as knowledge base and a translation from the extracted association rules to Jena rules is performed. We consider the association rule, obtained above in the mining process:

82% of the students who followed Web Technologies (WT) course, also followed Web Application (WA) course.

Using such rules, we can recommend to some students, *who already followed* WT *course and do not followed yet the* WA *course*, to consider follow that course, (e.g. in the next semester). Particularly, for currently authenticated student *Tom Miller*, we can recommend him to consider the *WA* course for the next semester but this is not an available information for another authenticated student *John Smith* who already followed *WA* course. Newly obtained data is not stored into Moodle database. Instead, it is used by the Moodle view module when the page for this student is generated.

The above association rule translate into the following Jena rule:

```
[R:
(?x rdf:type moodle:Student)
(?x moodle:username moodle:Tomy)
(?x moodle:takenCourse moodle:WT)
noValue(?x moodle:takenCourse moodle:WA)
->
(?x moodle:followCourse moodle:WA)]
```

We have to note that the second triple of our rule is dynamically created when the user access a page, and isn't part of the association rule. The subject of the triple `(?x moodle:username moodle:Tomy)` is obtained by using the *usernames* of the current logged users. Using this technique, we can express user-based recommendations. We don't want to recommend a course for a student which already followed that course. For the rule expressed in Jena, we use the `noValue` builtin to check in the working memory the triple denoted by the built-in parameters and it fails if the triple is found. Assuming that our rule is expressed as $WT \Rightarrow WA$, and denoting with D_{WT} the set of all students who followed WT course and with D_{WA} the set of all students who followed the *WA* course, then have to analyze four possible situations:

- *x is a positive example* - $x \in D_{WT} \wedge x \in D_{WA}$ - for our case, this express that the student already followed the WT course and also the WA course. This situation is covered: if the student already followed both courses, then the rule do not fire.
- *x is a non-positive example* - $x \notin D_{WT} \vee x \notin D_{WA}$ - for our case, this express that the student hasn't followed the WT course or hasn't followed the WA course. In our rule, if the student hasn't followed the WT course, the rule will do not fire, and if the student hasn't followed the WA course then the rule fire only if he followed the WT course.
- *x is a negative example* - $x \in D_{WT} \wedge x \notin D_{WA}$ - in this case, the rule fire and we recommend for that student to consider the WA course.
- *x is a non-negative example* - $x \notin D_{WT} \vee x \in D_{WA}$ - in this case the rule don't fire, since the student either hasn't followed WT course or already followed WA course.

We can note that in the case of a *positive* and *non-negative* example, the rule do not fire and in the case of a *negative* example the rule always fire. For the case of *non-positive* example, the rule fire only if the student followed the WT course but not followed the WA course. We have this situation because we want to recommend some actions only to students which cover the conditions of the boolean association rule but do not cover all conclusions.

The rule from the above example has a simple structure: only one antecedent atom (translating into a condition) and one precedent atom (translating into a conclusion). Sometimes, rules are more complex:

74% of the students who get WT course and passed the test T2 have accessed resource Res1 and solved assignment A1.

A result of applying a reasoning using such a rule, can be to guide the student to read some resources and to do some specific actions in order to prepare himself for a specific test.

There is a need to to make suggestions only for students which accomplish all conditions and we need to recommend only those parts from conclusion which are not accomplished yet. For the student *Tom Miller* who follows the *WT* course, and already accessed resource *Res1*, we need only to suggest him to consider solving the assignment *A1*. For this case we have also a supplementary condition, expressing that he hasn't passed yet the test *T2*. For the rest of the students, this recommendation will not be useful. Such statements, from association rules, translate into negated conditions in Jena rules.

We translate this rule into two Jena rules:

```
[R1:
(?x rdf:type moodle:Student)
(?x moodle:username moodle:Tomy)
(?x moodle:takenCourse moodle:WT)
noValue(?x moodle:passTest moodle:T2)
noValue(?x moodle:accessedResource moodle:Res1)
->
(?x moodle:accessedResource moodle:Res1)]

[R2:
(?x rdf:type moodle:Student)
(?x moodle:username moodle:Tomy)
(?x moodle:takenCourse moodle:WT)
noValue(?x moodle:passTest moodle:T2)
noValue(?x moodle:solvedAssignment moodle:A1)
->
(?x moodle:solvedAssignment moodle:A1)]
```

Consider having the student *Tom Miller*, with the username *Tomy*. It has followed the *WT* course (but not the *WA* course), hasn't passed the *T2* test, hasn't accessed the *Res1* resource and also hasn't solved the *A1* assignment. According with those statements, Tom Miller accomplish conditions from rules **R**, **R1** and **R2**. Conform with the rule **R**, we recommend him to consider follow the *WA* course, and according with **R1** and **R2** we recommend him to solve the assignment *A1* and to read information refereed by *Res1*. All those information are inferred and cannot by obtained directly from the activities logs.

For the general case, a boolean association rule:

$$A_1 \wedge A_2 \wedge ... \wedge A_n \Rightarrow B_1 \wedge B_2 \wedge ... \wedge B_m$$

translate into many Jena rules:

$$R_1 : A_1 \wedge A_2 \wedge ... \wedge A_n \wedge \neg B_1 \Rightarrow B_1$$
$$...........$$
$$R_m : A_1 \wedge A_2 \wedge ... \wedge A_n \wedge \neg B_m \Rightarrow B_m.$$

The general case, for boolean association rules, already implies extraction of simple rules having the same conditions and each of them having the conclusion formed by one of the atoms from the association rule conclusion:

A simple association rule:

$$A_1 \wedge A_2 \wedge ... \wedge A_n \Rightarrow B$$

translate in the Jena rule:

$$R_1 : A_1 \wedge A_2 \wedge ... \wedge A_n \wedge \neg B \Rightarrow B$$

In order to obtain Jena rules from association rules, select only association rules having good a probability (*strong association rules*). For this reason, a minimum value is selected for both support and confidence measure factors. For different rule sets, different values for maximum and minimum factors are set.

3.3 Jena Rules inference submodule

After obtaining Jena rules starting from association rules, the next step is to use those rules, and Moodle activities logs to infer supplementary information. Jena API contains a module capable of extracting models used in the reasoning process, directly from a MySQL database. The inference submodule directly link to the Moodle database and extract all necessarily data from tables, creating RDF triples which are stored in the working memory. Those triples represents the initial facts base. At the next step, Jena rules obtained from association rules, are loaded by the engine into a *RuleStore* object. When the inference process in finished, the working memory contains new facts obtained by applying rules over initial facts base. New information (facts) are used to create additional information and recommendations in the user view page. The new information is temporarily stored, and is processed by the view submodule of the extension.

3.4 Adding strong negation for Moodle data

We saw that in Jena Rules we use `noValue` built-in for checking the existence of some specific facts in the working memory (it implements a form of *negation-as-failure*). Assuming that our goal is to find out for some accessed resources (from a specific course), which of them are considered useful by students, it is possible to obtain both useful not useful resources. Also it is possible to have an overlap. The meaningful recommendations address useful resources, therefore we want to suggest only those resources which are considered useful by the others. This can be naturally expressed, by using negative facts (e.g. we can have facts expressing that a resource is marked as not useful by some of the students). This can't be expressed by using *negation-as-failure*, since a student can mark the resource as useful, other student mark the same resource as not useful and other student don't mark at all. Using *negation-as-failure*, we may conclude that a resource is *not useful* just because it was not marked as *useful*. This is not always true: not marking as useful, sometimes means that the student has not marked the resource since it has no opinion about that resource at the moment of questioning. This is related to Open World Assumption (OWA) and Closed World Assumption (CWA). In the case of CWA, not marking the resource means that we have a not useful resource. In the case of OWA, not marking the resource means that it's status is *undetermined*.

Introduced in [3], and based on Partial Logic[11], ERDF comes with a solution to allows such facts. It use *strong negation* in order to represent negative information, e.g. *not-useful* resources. In this way, the property `moodle:usefulResource` is represented as a *partial property*, and it can represent positive information, negative information, ambiguous resource, or don't represent information at all (undetermined). Moreover, ERDF supports closed and open world assumption. Some predicates are closed (are totally represented in the knowledge base), and for those we can infer negative information if positive information can't be inferred. Other predicates are partial, and for those we can express multiple truth values (true, false, overdetermined and undetermined).

A prototype of an ERDF engine was developed and is available for online[4] testing. It is based on Jena and supports strong negation and a form of negation-as-failure.

4 System Architecture and Implementation

Figure 1 illustrates the overall architecture of the system. The system contains four specific parts (modules):

- *Mining Association Rules module* - extract association rules using Weka API;

[4] `http://oxygen.informatik.tu-cottbus.de/JenaRulesWeb`

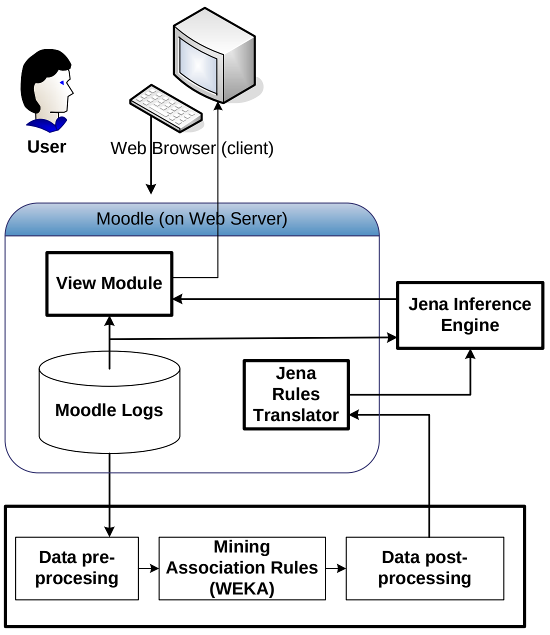


Figure 1. System Architecture

- *Jena Rules Translator module* - maps association rules to Jena rules.
- *Inference Engine module* - interact with the Jena inference engine. It uses the Jena rules obtained from the previous module, and Moodle activities logs as initial working memory.
- *View module* - improve user views, by adding new information obtained from the inference process and possible obtained recommendations.

The *Mining Association Rules module* connects to Moodle database, obtains activities logs, select and prepare data in order to extract association rules. For the mining process, WEKA is used.

Using the second module, association rules which are obtained from the mining process are then translated to Jena rules. The inference engine runs as a servlet and uses Jena API and rules obtained before in order to obtain new information. Finally, a PHP module improve the final view of the authenticated user with new information and possible recommendations obtained after the inference process.

Some operations are dynamical (e.g. the reasoning process, creating views), and others are created timely by a *cron* process (e.g. mining logs to obtain boolean association rules, translate association rules into Jena rules). Before Jena rules are passed to the inference engine, for each rule, a triple expressing the identity of the currently logged user is added (e.g. `(?x moodle:username moodle:Tomy)`). Also, new triples regarding authenticated users are added to memory when a user login to the Moodle system. Multiple users authentication is supported by adding a new triple for each new authenticated user. Those triples allow us to identify relevant information for specific users.

5 Conclusion and future work

The paper describes a Moodle extension used to create improved views for users by adding recommendations based on the existing data about user activities. The view is created by using the user-data as input for a rule-based learning recommendation processing.

Future work include representation of negative facts in Moodle activities and using fuzzy association rules instead of boolean association rules. In addition, we intend to develop a rule designer module which allows (for tutors) to create general/specific interest rule based on diverse criteria. General rules apply to all users of the system (e.g. create a message for every student which has not passed an exam).

REFERENCES

[1] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami, ‘Mining association rules between sets of items in large databases’, in *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993*, eds., Peter Buneman and Sushil Jajodia, pp. 207–216. ACM Press, (1993).

[2] Rakesh Agrawal and Ramakrishnan Srikant, ‘Fast algorithms for mining association rules’, in *Proc. 20th Int. Conf. Very Large Data Bases, (VLDB)*, eds., Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, pp. 487–499. Morgan Kaufmann, (12–15 1994).

[3] Anastasia Analyti, Grigoris Antoniou, Carlos Viegas Damasio, and Gerd Wagner, ‘Negation and Negative Information in the W3C Resource Description Framework’, *Annals of Mathematics, Computing and Teleinformatics*, **1**(2), 25–34, (2004).

[4] D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation February 2004. http://www.w3.org/TR/rdf-schema/.

[5] Sergey Brin, Rajeev Motwani, Jeffrey D. Ullman, and Shalom Tsur, ‘Dynamic itemset counting and implication rules for market basket data’, in *Proceedings ACM SIGMOD International Conference on Management of Data*, pp. 255–264. ACM Press, (May 1997).

[6] Mircea Diaconescu, Sergey Lukichev, and Adrian Giurca, ‘Semantic Web and Rule Reasoning inside of E-Learning Systems’, in *Proceedings of 1st International Symposium on Intelligent and Distributed Computing*, eds., C. Badica and M. Paprzycki, Studies in Computational Intelligence, Craiova, Romania, (18-20 October 2007). Springer.

[7] C. Forgy, ‘Rete – A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem’, *Artificial Intelligence*, **19**, 17–37, (1982).

[8] Klyne G. and Caroll J.J. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation 10 February 2004. http://www.w3.org/TR/rdf-concepts/.

[9] Enrique Garcia, Cristobal Romero, Sebastian Ventura, and Toon Calders, ‘Drawbacks and solutions of applying association rule mining in learning management systems’, in *Proceedings of the International Workshop on Applying Data Mining in e-Learning (ADML’07)*, (September 2007).

[10] Jiawei Han, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[11] Heinrich Herre, Jan O. M. Jaspars, and Gerd Wagner, ‘Partial Logics with Two Kinds of Negation as a Foundation for Knowledge-Based Reasoning’, in *What is Negation?*, eds., D.M. Gabbay and H. Wansing, Kluwer Academic Publishers, (1999).

[12] Philip S. Yu Jong Soo Park, Ming-Syan Chen, ‘An effective hash-based algorithm for mining association rules’, in *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pp. 175–186, San Jose, Canada, (1995).

[13] Sergey Lukichev, Adrian Giurca, and Mircea Diaconescu, ‘Empowering moodle with rules and semantics’, in *Proceedings of 3rd Workshop on Scripting for the Semantic Web (SFSW2007)*, eds., T. Heath S. Auer, C. Bizer and G. A. Grimnes, Innsbruck, Austria, (6 June 2007).

[14] Ashok Savasere, Edward Omiecinski, and Shamkant B. Navathe, ‘An efficient algorithm for mining association rules in large databases’, in *Proceedings of 21th International Conference on Very Large Data Bases (VLDB’95)*, eds., Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, pp. 432–444. Morgan Kaufmann, (September 1995).

Using Rules for the Integration of Heterogeneous and Autonomous Context-Aware Systems

David Mosén[1] and **Arantza Illarramendi**[2] and **Mohand-Said Hacid**[3]

Abstract. In this paper we introduce NUBIA, a middleware that combines, through business rules, information generated by heterogeneous and autonomous systems. Communication between NUBIA and systems is loosely-coupled and highly compatible, as Web Services and other standards are used. The main component of NUBIA is a rule engine, sensible to temporal knowledge and with integrated functions (e.g. frequencies, percentages). A user friendly interface allows entering, using a script language, customized rules, whose conditions depend on the information sent from the systems. For each rule, settings such as the maximum firing frequency and the activation within a group may be defined. Moreover, NUBIA also supports a rule editor role, which allows a more realistic viewpoint of context-aware rules customization. Finally, automatic rule translation to the user's language and a role-oriented interface facilitate the interaction with NUBIA.

1 INTRODUCTION

The term context awareness has evolved through time, but has always maintained a sense of somehow gathering data from the environment. In the last years, the definition of context by Dey et al. [1] seems to be the most widely embraced. It states that context is "any information that can be used to characterize the situation of entities that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity and state of people, groups and computational and physical objects". Notice that this definition implies that context is not necessarily physical, so it can be sensed through virtual sensors (e.g. Web Services).

Nowadays, context-aware systems work in general in an autonomous way and do not interact among them. However, connecting them can, in many situations, increase the advantages that they provide separately. Let us take the example of two systems, in charge of monitoring a house and vital signs of a person, respectively. A connection between them can boost alertness in a home for elderly people scenario, where a single unit reacts to information about both the house and the owner's health. Of course, each system should still be able to be in control of its own domain.

For a tight coupling among systems, a complex manual process may be needed. With this in mind, we present in this paper NUBIA, a middleware that integrates, through loosely-coupled connections, any kind of context-aware systems, independently of the domain that they consider, while preserving the autonomy of each system. Thus, internal management in each domain remains within the respective system, whereas a combined management across domains is handled through NUBIA.

To test the middleware and demonstrate its usefulness, we constructed a typical homecare scenario, oriented towards a single inhabitant (i.e. the user), although visits pose no actual problem. Abundant previous work about homecare can be found [2] [3], and we refer to it for a more thorough study on the subject. However, our aim is to show how an integral homecare system can be build adding up independent, separately developed, components. Thus, the personal healthcare application, provided by SaludNova [4], and the domotics system we integrated were already developed, in order to test out the integration issue in a real scenario.

Benefits from integration through rules of the mentioned systems can be seen in several cases. For example, the healthcare system lacks fall detection, so if domotics determines using its sensors that the user fell, a warning is sent to NUBIA, which in turn warns the healthcare system. In the same way, the domotics system can be commanded to turn off the oven and stove if the user suffers a heart failure. Further, if a risk situation requires both systems to detect certain alarms, then a third system, in charge of handling joint alarms, is notified. As a final example, this additional alarm system can also be notified of errors in the main systems (see Subsection 3.2.1).

Summing up, the main features of NUBIA are the following ones:

1. Because sensed context is information sent from connected systems through Web Services, it is a context-aware application too.
2. It allows combining context-aware systems information through customized business rules sensible to knowledge extracted from a stored history.
3. Finally, it provides a user friendly graphical interface, which supports different user roles (final, rules editors and administrators).

2 RELATED WORK

Concerning related works, we can observe that trends in software development move toward generic design patterns. In that direction, there seems to be an agreement on the need for finding a context-aware standardized architecture [5]. This area has been widely studied, with several proposals of context-aware frameworks [6] [7]. We also present a proposal of a standardized architecture; however, our goal in this paper is to focus on the use of rules for getting the integration of context-aware systems.

On context integration, the work by Tao Gu et al. on SOCAM [8], an architecture for developing context-aware services, is the closest

[1] University of the Basque Country, Spain, email: david@mosen.es
[2] University of the Basque Country, Spain, email: a.illarramendi@ehu.es
[3] Univ. Claude Bernard Lyon, France, email: mshacid@liris.univ-lyon1.fr

to our proposal. We basically differentiate on what is integrated, as we aim at autonomous context-aware systems integration, while SOCAM directly integrates physical context. Anyhow, both proposals carry out the idea of reasoning with the context to offer some kind of service to external applications.

Similarly to NUBIA, the system proposed by Agarwal et al. on electronic medical records [9] uses business rules to combine gathered history information, namely, from RFID sensors and patient monitoring. Thus, our proposal is similar in the sense that we use gathered information in business rules. However, in the case of NUBIA, information used is context which comes from any kind of context-aware system, thus not restricted to the medical field.

Finally, our approach towards personalization, based on customized rules managed by a rule editor, can be situated among those that appear on the existing literature, neatly explained by Henricksen & Indulska [10]. They classify personalization approaches on context-aware systems into three categories: end user programming, machine learning and preference-based. The one used in NUBIA is similar to the end user programming, but it provides as a novelty a scheme where the user relies on a rule editor to make the adjustments he/she requires.

3 NUBIA ARCHITECTURE

The middleware core is divided into three main units. First, the *context handling module*, which transforms context to a standardized representation and deals with history knowledge. The *rule engine module*, which manages the combined information using business rules. Finally, the *communication module*, which provides the context handling module with the context coming from systems, and allows the rule engine to communicate about actions to execute in the systems. Hence, notice that communication flows both ways between NUBIA and systems.

In this section, we briefly explain the main features of the mentioned modules, which can be seen in Figure 1.

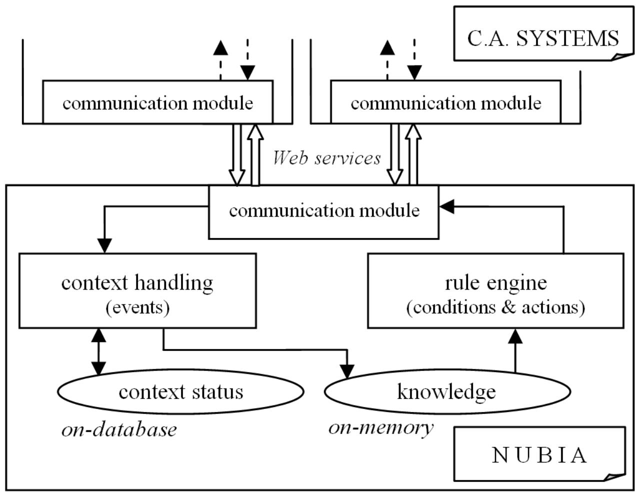


Figure 1. NUBIA's architecture

3.1 Context handling module

Taking into account that systems can communicate heterogeneous context, there is a need for its classification into predefined groups. This process is called categorization.

3.1.1 Categorization

Two categorization viewpoints exist [11]: *conceptual*, based on what the context symbolizes (e.g. person, network, physical environment), and *measurement*, based on how the context is represented (e.g. a value, a predicate). There exists a bigger tendency to follow the conceptual categorization [12] [13]. However, we chose a measurement categorization in order to explore its possibilities in relation to the management of history knowledge. Hence, context is classified into the following four categories:

1. Single. Simple events, which happen at a very precise moment. Alerts and punctual detections fit into this context category.
2. Discrete. The context can only take one value once at a time from a finite set of values, each of which represents a state. Examples in this category are device's power state, which toggle between on and off, and generally any context whose possible states are well-defined.
3. Continuous. In this case, the value representing the context is a real number, so we can make comparisons to see whether it is within a certain range. Uncountable data belong to this category.
4. Descriptive. Compound content cannot be represented by any of the three categories above. This category is based on the description statement and uses the notion of predicate. A person's location, for example, is represented as location(person,place).

Furthermore, context reported by systems may correspond to different levels of reasoning, ranging from raw data to thoroughly reasoned knowledge. Figure 2 shows a layered scheme [14] where context information flows to the middleware from any of the two first layers of a context-aware system.

Nevertheless, the category to which context belongs is the only relevant distinction and context is manipulated based on it. For example, a light switch, the energy level of an electron and a storm emergency level are all considered as discrete context, regardless the complexity of the process to obtain them.

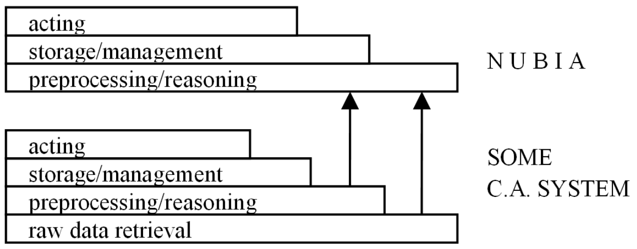


Figure 2. Information flow in a layered scheme

3.1.2 The summarizing log

NUBIA manages a special type of history, called a *summarizing log*, which gets updated every time some context information is received from a system. A typical history logs everything, for an eventual use, without any further modification than the addition of new records. In a summarizing log, instead, a logging action causes a modification that updates key information about the current situation. This logging method helps to extract knowledge from the history, that we refer as temporal knowledge.

Stored information such as number of times in a state or elapsed time within a range, together with interesting timestamps, are enough to infer the above mentioned temporal knowledge (e.g. frequencies, time since last state shift).

Let us suppose that we want to know the frequency of a given simple event. Two fields are required in the summarizing log: the first time the event is registered, and the times count. Thus, the frequency can be calculated:

$frequency = (now - first\_time)/times\_count$

There is also information directly extracted from the summarizing log, as it is useful without any further operation. Examples are: within which ranges from a defined set is some continuous value; and the number of times in a certain state. For these two cases, the following information could be part of the log:

(name:corporalTemp(at)temp(at)biometrics, currentValue:36.6)
(range:[36.0,36.9], lastBegan:1206000000[4]*, lastEnded:-)*
(range:[36.0,36.6],
lastBegan:1206000800, lastEnded:1206000900)
(range:[36.8,max], lastBegan:1206001000, lastEnded:-)

(name:faintRisk(at)alarms(at)biometrics, currentState:low,
lastShift:1206010000, shifts:7)
(state:low, times:5, last:1206010000)
(state:average, times:2, last:12060000300)
(state:high, times:0, last:-)

Conditions in the rules (see Section 3.2) are checked against all this knowledge, temporal and non-temporal.

Finally, notice that summarizing logs are not aimed at applications with infrequent or full history reasoning [16] [17], as they are not powerful enough. Context-aware systems, however, have a strong requirement on time, and a summarizing log helps to get quick response times. Moreover, it is compatible with a full-fledged history, so that the best of both worlds is available.

3.2 Rule engine module

This module evaluates rules that trigger depending on the context knowledge extracted from the summarizing log. The rule engine is independent from the communication process. As a result, systems can continue to report context information even if the rule engine is not active.

3.2.1 Rule structure

Two classes of business rule engines exist. First, and the one used in our proposal, a production rule engine deals with rules with ”IF condition THEN action” semantics. Usually, an external agent invokes the engine, so in the scenario of a context-aware system with production rules, the system typically invokes the engine whenever some context is sensed. If, given the new situation, the conditions of a rule are true, it fires.

Second, reactive rule engines. In this case rules fire when their conditions are true as well, but they need some event to happen in order to get evaluated. Hence their name, Event Condition Action (ECA). This class of rule engine is suited for most context-aware systems, because sensed context adjusts well to the concept of event. Thus, the system does not need to explicitly invoke the rule engine, as it is already aware of generated (context) events.

In NUBIA, many defined conditions depend on time, so they cannot be evaluated only when some context information arrives, because temporal knowledge must also be taken into consideration. There are two possible mechanisms to deal with this situation. Let us take the following as an example condition:

The light has been in state off for 5 minutes

The first option (continuous evaluation) is to constantly check whether 5 minutes have elapsed since the last shift to off. The alternative (evaluation scheduling) is to schedule the system to invoke the rule engine 5 minutes after each time the light changes to the off state. This alternative is more efficient, as it uses processing resources more wisely. However, it is also non-trivial, because depending on the condition semantics, evaluations should be scheduled in different ways (e.g. with a certain frequency for a limited time, when the event is detected). In either case (continuous or scheduled evaluation), the middleware is in charge of telling its rule engine when to evaluate the rules (i.e. events do not directly trigger the rules) so we chose to implement them as production rules. In particular, rules are implemented using JBoss Rules [19], following a forward-chaining production structure.

The right hand side of the rules comprises two kinds of actions: internal and external.

External actions are not executed by the middleware, but in a connected system. In addition to sending context information, systems may expose actions to NUBIA through Web Services, so that they can be ordered to execute the actions.

Internal actions control NUBIA itself and gathered information. This includes error count resetting. NUBIA detects both incoming communication errors (i.e. context reported by a system is invalid or the message is corrupt) and outgoing communication errors (i.e. the system to which to connect is unreachable). Data can also be reset if, for example, the information about a certain context should be initialized. Finally, a system may refuse to execute an ordered action, so this can be used in the condition part too.

3.2.2 Settings

Some of the incorporated settings in the rule engine include:

Maximum firing frequency. Controls repetition of rule firing. Even if a rule is evaluated to true, it will not be fired unless the defined time has elapsed. In that case, it will only fire if it is still true. For example, the user may want to be notified of new mail after at least 2 hours since the last notification, even if mail arrives in the meanwhile.

Activation group. This setting has been extended from the JBoss Rules option with the same name. The rule within an activation group with the highest priority is executed; the rest, albeit evaluated as true, do not get a chance to be executed until the time defined by the group’s maximum firing frequency goes by (all rules in a group have the same maximum firing frequency). For example, if the user has a tumble, the system should call a relative, but if, additionally, the user suffers a heart-attack, this action may be overridden by a call to the

[4] Timestamps are described in Unix time [15] (seconds elapsed since 01 Jan 1970, 00:00:00 UTC).

emergency number.

Other settings, such as firing count limit and expiry dates, may eventually be included as well. The existance of some of these settings in JBoss Rules might facilitate their implementation.

3.2.3 NIRE language

As a final point concerning the rule engine module, we designed a script language to facilitate the definition of rules and check their validity, so that they do not cause errors during execution. The NUBIA Input Rule Editing (NIRE) language provides the following benefits:

1. Transparency and independence from the underlying rule implementation engine.
2. A compact syntax, with no unneeded verbosity.
3. Translation extensibility through XML, allowing the definition of new rule translations to other user languages without recompiling the application.

The following is an example of a rule in NIRE. Notice that settings are not defined in the language, as they are introduced through the graphical interface.

```
if
  is-true
    presence@locator@wear $somebody
  last-time-in-range
    temperature@temp01@domotics 15 27 > 3600
  time > 18:00:00
then
  turn-heater (using heater@domotics) "on"
  display (phone(at)aux-phone)
    "$somebody is home, turning heater on."
```

The rule has a typical ”IF condition THEN action” structure. Each of the first two conditions are stated over a certain context, while the third is an internal NUBIA condition which controls the time of the day. Each action is defined by a name and its corresponding device and system. In this rule, both actions require one parameter each, delimited by double quotation marks. Concerning symbols, ”@” (i.e. at) denotes in which device and system a context is sensed or an action is executed, whereas the dollar symbol, denotes variables. Thus, the first condition states that somebody must be detected, and saves his/her name in the ”somebody” variable.

If the chosen translation language is English, the user would see this resulting text:

If presence is true for a certain person ’somebody’, temperature has not been between 15 and 27 °C in the last hour and it is more than 6 in the afternoon, then turn heater on and display in the phone ”’somebody’ is home, turning heater on.”.

3.3 Communication

Information flows between systems and NUBIA in both ways: context information is reported to the middleware and orders to execute actions are sent back. In either case, Web Services are used. Thus, to make communication possible, developers who wish to have their systems integrated need to: make methods to be used by NUBIA available through a Web Service; and report desired context to NUBIA’s Service.

Usually, applications use a fixed set of Web Services, but sometimes they may require to call beforehand unknown Web Services. Dynamic invocation allows client applications to invoke Web Services whose descriptions are unknown until the application is used. As an example implementation, the Dynamic Invocation Interface (DII) [18] for CORBA supports this functionality. NUBIA needs the dynamic invocation, given that it is aimed at working with beforehand unknown systems.

For a higher decoupling from inner operation, serialized XML are sent, so if more communication-related functionalities are added to the middleware, only the XML representation would change, whilst Web Services in connected systems and the middleware would remain the same. The following XMLs are examples of incoming and outgoing messages, respectively:

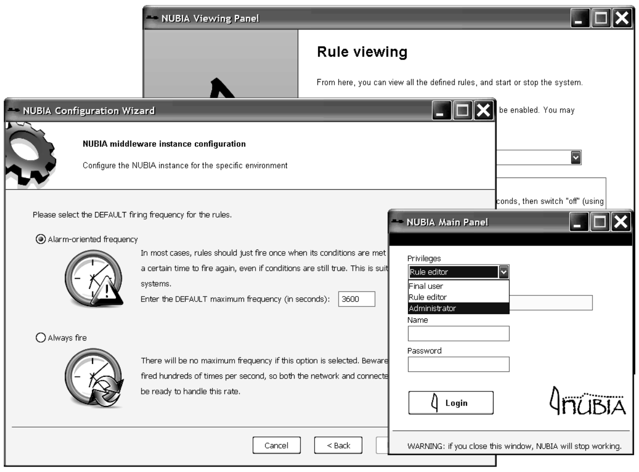
<event time=”1206000000” xmlns=”http://www.tempuri.org”>
<signal name=”smoke” device=”smk” system=”domotics”/>
<continuousInfo>0.32</continuousInfo>
</event>

<request time=’1206000000’ xmlns=’http://www.tempuri.org’>
<action name=’switch’ device=’light01’ system=’domotics’/>
<parameter>off</parameter>
</request>

Despite Web Services are the best communication option because of its wide de facto standardization, there are systems that do not fully accept them, such as smartphones, most of which cannot host a web server. To cope with this difficulty, socket communication stands in NUBIA as an alternative to Web Services.

4 INTERFACE

The task of integrating autonomous context-aware systems is not easy, so NUBIA provides a graphical interface that focuses on the separation of the different types of users (i.e. roles) for a more specific interaction with each of them. Therefore, before we show the main features of the GUI, we present the user roles considered by NUBIA.



From here, you can view all the defined rules, and start or stop the system.

be enabled. You may

conds, then switch "off" (using

Configure the NUBIA instance for the specific environment

Please select the DEFAULT firing frequency for the rules.

In most cases, rules should just fire once when its conditions are met

a certain time to fire again, even if conditions are still true. This is suit

systems.

Enter the DEFAULT maximum frequency (in seconds): 3600

There will be no maximum frequency if this option is selected. Beware

fired hundreds of times per second, so both the network and connecte

be ready to handle this rate.

WARNING: if you close this window, NUBIA will stop working.

Figure 3. Some windows from NUBIA’s interface

4.1 Roles

Relation with end users is a thoroughly studied issue in context-aware systems. Giving users control over a system they use implies they need to learn, in some degree, how to interact with it. Different approaches towards this interaction exist [10], but they lack either power of control or simplicity, so we define a role that fills the gap between the end user and the administrator, to keep the user somehow in control and yet increase simplicity in their interaction with the system. Thus, for our middleware NUBIA, we establish a three tiered role division to better focus on each role requirements:

1. Administrator. There only exists one, and defines which systems to connect to NUBIA and their specifications, probably handed by other administrators or developers. Also, he/she configures settings such as working mode (dedicated server or shared machine) and application defaults. Finally, the administrator is in charge of account management.
2. Rule editor. Manages rules without the need to know about application programming or technical details. A user with a little bit of technical knowledge may manage rules and act as a rule editor too.
3. End user. Can only see the defined rules and decide whether to start or stop NUBIA.

In this scheme, roles are incremental, with the administrator having privileges as a rule editor and user as well.

4.2 Ease of interaction

A configuration wizard à la MySQL [20] (see figure 3) guides the administrator to easily configure, for example, system defaults and working mode.

Systems specifications are defined in XML, so they can be checked through an XML-schema and translated to internal representation through an XSL transformation. This way, portability, independence and easy handling are achieved and hence, the administrator selects the file with the definitions and NUBIA does the rest.

Maximum firing frequency, validity expiration or activation policies within a group are settings that allow a more refined execution control. Nevertheless, defined defaults (by the administrator or NUBIA itself) make simple rule creation an easier task. A rule preview, an auxiliary panel with available data and syntax documentation, and an accurate error checking facilitate rule creation even more.

Automatic translation of rules to the user's language allows less technical users to easily understand them, whilst not giving extra work to the rule editor.

5 CONCLUSIONS

We have presented a middleware that successfully connects, with loosely coupled Web Services, autonomous context-aware systems, combining and making use of their information to trigger business rules. Interaction is made through an easy to use interface, designed taking roles into account.

Using the implemented prototype, we observed the following behavior: a rule may be triggered by incoming context information or because some defined time has elapsed. Taking both possible cases into account, the average response time starts at 10 milliseconds with a few defined rules, each of which adds a 4 microseconds overhead.

This last fact confers the middleware scalability concerning rules. Finally, the fast processing of Web Service messages supports heavy communication between NUBIA and the context-aware systems, so a great scalability is also achieved in the amount of integrated systems and communication with them.

ACKNOWLEDGEMENTS

This work is supported by the Spanish Ministry of Education and Science (TIN2007-68091-C02-01) and the Basque Government (IT-427-07).

REFERENCES

[1] A. Dey, D. Salber, and G. Abowd, *A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications*, Human-Computer Interaction, vol. 16, pp. 97-166, 2001.

[2] A. Cesta & F. Pecora, *Integrating Intelligent Systems for Elder Care in RoboCare*, W.C. Mann & A. Helal (Eds): Promoting Independence for Older Persons with Disabilities, IOS Press, pp. 65-73, 2006.

[3] G. Virone et al., *An Advanced Wireless Sensor Network for Health Monitoring*, D2H2, Arlington, Virginia, 2006.

[4] SaludNova Sociedad Cooperativa, website: http://www.saludnova.com/ (as of April 2008).

[5] M. Baldauf & S. Dustdar, *A Survey on Context-Aware Systems*, Technical Report Number TUV-1841-2004-24, November 2004.

[6] J.E. Bardram, *The Java Context Awareness Framework (JCAF)*, Tech. Report CfPC 2004-PB-61, Centre for Pervasive Computing, Aarhus, Denmark, 2003.

[7] K. Henricksen & J. Indulska, *A Software Engineering Framework for Context-Aware Pervasive Computing*, Second IEEE International Conference on Pervasive Computing and Comms. (PERCOM 2004), March 2004.

[8] T. Gu, H. Pung , D. Zhang *A service-oriented middleware for building context-aware services*, Journal of Network and Computer Applications 28(1): 1-18, 2005.

[9] S. Agarwal, *Context-Aware System to Create Electronic Medical Encounter Records*, Technical Report Number TR-CS-06-05, 2006.

[10] K. Henricksen & J. Indulska, *Personalising Context-Aware Applications*, in OTM Workshop on Context-Aware Mobile Systems, Springer-Verlag, pages 122–131, 2005.

[11] M.A. Razzaque, *Categorization and Modeling of Quality in Context Information*, in Proceedings of the IJCAI Workshop on AI and Autonomic Communications, 2005.

[12] Eleftheria Katsiri & Alan Mycroft, *A first-order logic model for context-awareness in distributed sensor-driven systems*, RSPSI Workshop, 2006.

[13] Anjum Shehzad, Hung Q. Ngo, Kim Anh Pham and Sungyoung Lee, "Formal Modeling in Context Aware Systems", KI-Workshop Modeling and Retrieval of Context (MRC2004), 2004.

[14] S.W. Loke, *Context aware pervasive systems : the architecture of a new breed of applications* (ISBN: 0849372550), Abstract layered architecture (page 25) . Boca Raton, FL : Auerbach Publications, 2006.

[15] *Unix time* article from Wikipedia, available at http://en.wikipedia.org/wiki/Unix_time.

[16] G.M. Youngblood et al., *Automation Intelligence for the Smart Environment*, IJCAI-05, page 1513, 2005.

[17] V. Jakkula & D. Cook, *Anomaly detection using temporal data mining in a smart home environment*, Methods of Information in Medicine, 2008.

[18] BEA Documentation on *the Dynamic Invocation Interface*, available at http://e-docs.bea.com/tuxedo/tux80/creclien/dii.htm (as of April 2008).

[19] JBoss Rules, http://www.jboss.com/products/rules (as of April 2008).

[20] MySQL 5.0 Server Configuration Wizard, documentation available at http://dev.mysql.com/doc/refman/5.0/en/mysql-config-wizard.html.

XTT+ Rule Design Using the ALSV(FD)

Grzegorz J. Nalepa and **Antoni Ligęza** [1]

Abstract. This paper presents advances in Set Attributive Logic and its application to develop tabular rule-based systems within the XTT framework. The primary goal is to extend the expressive power of simple attributive languages so that it becomes satisfactory for complex applications, including the business rules support. A formal framework of extended Attributive Logic with Set Values over Finite Domains (ALSV(FD)) is presented and specific inference rules are provided with their corresponding prototype in PROLOG.

1 INTRODUCTION

Rule-based systems (RBS) are one of the most efficient paradigms for knowledge representation and automated inference. This is an intuitive and well-established language [5]. However, when it comes to the engineering practice, as well as its scientific aspect, the formal approach to the rule language specification has to be considered. In fact, there are number of specific rule languages based on different formal calculi, from simple propositional logic, through subsets of predicate calculus, to specific higher-order logics [9].

This paper presents advances in *Set Attributive Logic* and its application to develop tabular rule-based systems within the XTT framework. The primary goal is to extend the expressive power of simple attributive languages so that it becomes satisfactory for complex monitoring, control, decision support and business rules applications. A formal framework of extended Set Attributive Logic is presented and specific inference rules are provided. The practical representation and inference issues both at the logical and implementation level are tackled.

2 HEKATE RULE LANGUAGE

In the HEKATE project (`hekate.ia.agh.edu.pl`) an extended rule language is proposed. It is based on the XTT language described in [11]. The version used in the project is currently called XTT+.

The XTT+ rule language is based on the classic concepts of rule languages for rule-based systems [8], with certain important extensions and features, such as:

- strong formal foundation based on attributive logic,
- explicit rulebase structurization,
- extended rule semantics.

In this paper the XTT+ language will be simply referred to as XTT.

In XTT there is a strong assumption, that the rule base is explicitly structured. The rules with same sets of attributes are grouped within decision tables. On the rule level explicit inference control is allowed. In this way, a set of tables is interconnected using links, corresponding to inference control. This makes up a decision-tree like structure, with tables in the tree nodes. In a general case, the XTT is a directed graph, with cycles optionally allowed.

In RBS, a rule has a general format:

```
IF condition THEN decision
```

This format can be used in both forward and backward chaining systems. However, here we focus on the production rule systems, based on the forward chaining paradigm. The power of a rule language stems from the syntax and semantics of the conditional and decision expressions. Number of systems implicitly assume, that this rule format can be extended to the *conjunctive normal form (CNF)*, that is:

```
IF cond1 AND cond2 AND ... AND condN
THEN decision
```

which in fact corresponds to a *Horn* clause φ [1, 9], that is:

$$\varphi = \neg p_1 \vee \neg p_2 \vee \ldots \neg p_k \vee q,$$

Such a clause can be represented as an implication of the form:

$$\varphi = p_1 \wedge p_2 \wedge \ldots \wedge p_k \Rightarrow q.$$

which can be regarded as a rule in the above format, where ps correspond to conditions and q corresponds to the decision. In fact the PROLOG language uses a subset of predicate calculus, restricted to *Horn* clauses [3].

The decision expression can also be a compound one in the CNF. Now the question is what are the conditional and decision expressions. In number of systems these correspond to expressions in the propositional calculus, which makes the semantics somehow limited. Some systems try to use some subsets of predicate logic, which gives much more flexibility, but may complicate a RBS design and the inference process. This is the case of the PROLOG language [2]. In XTT these expressions are in the the *attributive logic* [9] described in more detail in Sect. 4. This gives much more power than the propositional logic, but does not introduce problems of the predicate logic-based inference. In XTT an extended rule semantics is used. These extensions were introduced in [13], and refined in [12].

Let us now move to attributive logic that provides a formal foundation for the rule language.

3 A MOTIVATIONAL EXAMPLE

Consider a simple piece of knowledge expressed with natural language as follows.

> The regular class hours are from 8:00 to 18:00. If all the teaching hours are located within regular class hours then the salary is regular. If the teaching hours goes beyond the regular class hours then the salary is special.

[1] Institute of Automatics, AGH – University of Science and Technology, Al. Mickiewicza 30, 30-059 Kraków, Poland email: `gjn@agh.edu.pl` `ligeza@agh.edu.pl`

The problem is to formalize these two rules with attributive logic. Let *RCH* stays for regular class hours, and *TH* for teaching hours. We can define a fact like:

$$RCH = \{8, 9, 10, 11, 12, 13, 14, 15, 16, 17\},$$

and

$$TH = \{10, 11, 12, 16, 19, 20\}$$

to specify a case of teaching hours. Note that teaching hours may form any subset of $\{0, 1, 2, 3, \ldots, 23\}$ (not necessarily a convex interval).

Now, to express the rules we need an extended attributive logic employing set values of attributes and some powerful relational symbols. For example, we can have:

$$R1: \; TH \subseteq RCH \longrightarrow Salary =' regular'$$

and

$$R2: \; TH \sim NRCH \longrightarrow Salary =' special'$$

where

$$NRCH = \{0, 1, 2, 3, 4, 5, 6, 7, 18, 19, 20, 21, 22, 23\}$$

is a specifications of non-regular class hours, and sim means a non-empty intersection. Note that an attempt to specify the rules with attribute logic based on atomic values (even if relational symbols such as $<$, $\leq$, $>$ and $\geq$ are allowed) would lead to a very long and at least clumsy set of hardly readable rules.

4 ATTRIBUTIVE LOGIC

Attributive logics constitute a simple but widely-used tool for knowledge specification and inference. In fact in a large variety of applications in various areas of Artificial Intelligence (AI) [14] and Knowledge Engineering (KE) attributive languages constitute the core knowledge representation formalism. The most typical areas of applications include rule-based systems [8, 9], expert systems (ones based on rule formalism) [6, 15] and advanced database and data warehouse systems with knowledge discovery applications [7] and contemporary business rules and business intelligence components (e.g. Jess, Drools).

However, it is symptomatic that although Propositional Logic and Predicate Logic (in the form of First-Order Predicate Calculus) have well-elaborated syntax and semantics, presented in details in numerous books covering logic for knowledge engineering [4, 6, 15], logic for computer science or Artificial Intelligence [1, 8], the discussion of syntax and semantics of attribute-based logic is omitted in such positions.

In a recent book [9] the discussion of attributive logic is much more thorough. The added value consist in allowing that attributes can take *set values* and providing formal syntax of the *Set Attributive Logic* (SAL) with respect to its syntax, semantics and selected inference rules.

The very basic idea is that attributes can take *atomic* or *set* values. After [9] it is assumed that an attribute A_i is a function (or partial function) of the form $A_i\colon O \to D_i$. A generalized attribute A_i is a function (or partial function) of the form $A_i\colon O \to 2^{D_i}$, where 2^{D_i} is the family of all the subsets of D_i. The atomic formulae of SAL can have the following three forms: $A_i = d$, $A_i = t$ or $A_i \in t$,

where $d \in D$ is an atomic value from the domain D of the attribute and $t = \{d_1, d_2, \ldots, t_k\}$, $t \subseteq D$ is a set of such values. The semantics of $A_i = d$ is straightforward – the attribute takes a single value. The semantics of $A_i = t$ is that the attribute takes *all* the values of t (the so-called *internal conjunction*) while the semantics of $A_i \in t$ is that it takes *some* of the values of t (the so-called *internal disjunction*).

The SAL as introduced in [9] seems to be an important step towards the study and extension of attributive logics towards practical applications. On the other hand it still suffers from lack of expressive power and the provided semantics of the atomic formulae is poor.

In this paper an improved and extended version of SAL is presented in brief. For simplicity no object notation is introduced. The formalism is oriented toward Finite Domains (FD) and its expressive power is increased through introduction of new relational symbols. The practical representation and inference issues both at the logical level and implementation level are tackled. The main extension consists of a proposal of extended set of relational symbols enabling definitions of atomic formulae. The values of attributes can take singular and set values over Finite Domains (FD).

4.1 ALSV(FD)

An extension of SAL was proposed in [10]. Both the syntax and semantics were extended and clarified. Here some further details to support set values of attributes over finite domains are discussed.

The basic element of the language of *Attribute Logic with Set Values over Finite Domains* (ALSV(FD) for short) are attribute names and attribute values. Let us consider:

$\mathbf{A}$ – a finite set of attribute names,
$\mathbf{D}$ – a set of possible attribute values (the *domains*).

Let $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$ be all the attributes such that their values define the state of the system under consideration. It is assumed that the overall domain $\mathbf{D}$ is divided into n sets (disjoint or not), $\mathbf{D} = D_1 \cup D_2 \cup \ldots \cup D_n$, where D_i is the domain related to attribute A_i, $i = 1, 2, \ldots, n$. Any domain D_i is assumed to be a finite (discrete) set.

As we consider dynamic systems, the values of attributes can change over time (or state of the system). We consider both *simple* attributes of the form $A_i\colon T \to D_i$ (i.e. taking a single value at any instant of time) and *generalized* ones of the form $A_i\colon T \to 2^{D_i}$ (i.e. taking a set of values at a time); here T denotes the time domain of discourse.

Let A_i be an attribute of $\mathbf{A}$ and D_i the sub-domain related to it. Let V_i denote an arbitrary subset of D_i and let $d \in D_i$ be a single element of the domain. The atomic formulae of ALSV(FD) are defined as follows.

Definition 1 *The legal atomic formulae of ALSV for simple attributes are:*

$$A_i = d, \tag{1}$$

$$A_i \neq d, \tag{2}$$

$$A_i \in V_i, \tag{3}$$

$$A_i \notin V_i. \tag{4}$$

Definition 2 *The legal atomic formulae of ALSV for generalized attributes are:*

$$A_i = V_i, \tag{5}$$

$$A_i \neq V_i, \tag{6}$$

$$A_i \subseteq V_i, \tag{7}$$

$$A_i \supseteq V_i \tag{8}$$

$$A \sim V, \tag{9}$$

$$A_i \not\sim V_i. \tag{10}$$

In case V_i is an empty set (the attribute takes in fact no value) we shall write $A_i = \{\}$. In case the value of A_i is unspecified we shall write $A_i =$ NULL (a database convention). If we do not care about the current value of the attribute we shall write $A = \_$ (a PROLOG convention).

The semantics of the atomic formulae as above is straightforward and intuitive. In case of the first three possibilities given by (1), (2), (3) and (4) we consider A_i to be a simple attribute taking exactly one value. In case of (1) the value is precisely defined, while in case of (3) any of the values $d \in V_i$ satisfies the formula. In other words, $A_i \in V_i$ is equivalent to $(A_i = d_1) \otimes (A_i = d_2) \otimes \ldots \otimes (A_i = d_k)$, where $V_i = \{d_1, d_2, \ldots, d_k\}$ and $\otimes$ stays for exclusive-or. Here (2) is a shorthand for $A_i \in D_i \setminus \{d\}$. Similarly, (4) is a shorthand for $A_i \in D_i \setminus V_i$.

The semantics of (5), (2) (7),(8), (9), and (10) is that A_i is a generalized attribute taking a set of values equal to V_i (and nothing more), different from V_i (at at least one element), being a subset of V_i, being a superset of V_i, having a non-empty intersection with V_i or disjoint to V_i, respectively.

More complex formulae can be constructed with *conjunction* ($\wedge$) and *disjunction* ($\vee$); both the symbols have classical meaning and interpretation.

There is no explicit use of negation. The proposed set of relations is selected for convenience and as such is not completely independent. For example, $A_i = V_i$ can perhaps be defined as $A_i \subseteq V_i \wedge A_i \supseteq V_i$; but it is much more concise and convenient to use "=" directly. Various notational conventions extending the basic notation can be used. For example, in case of domains being ordered sets symbols such as $>$, $>=$, $<$, $=<$ can be used.

4.2 BASIC INFERENCE RULES FOR ALSV(FD)

Since the presented language is an extension of the SAL (Set Attributive Logic) presented in [9], its simple and intuitive semantics is consistent with SAL and clears up some points of it. For example, the *upward* and *downward consistency rules* do hold and can be formulated in a more elegant way. Let V and W be two sets of values such that $V \subseteq W$. We have the following straightforward inference rules for atomic formulae:

$$\frac{A \supseteq W}{A \supseteq V} \tag{11}$$

i.e. if an attribute takes all the values of a certain set it must take all the values of any subset of it (downward consistency). Similarly

$$\frac{A \subseteq V}{A \subseteq W} \tag{12}$$

i.e. if the values of an attribute takes values located within a certain set they must also belong to any superset of it (upward consistency). These rules seem a bit trivial, but they must be implemented for enabling inference, e.g they are used in the rule precondition checking.

The summary of the inference rules for atomic formulae with simple attributes (where an atomic formula is the logical consequence of another atomic formula) is presented in Table. 1. In Table 1 and

Table 1. Inference rules for atomic formulae for simple attributes

$\models$	$A = d_j$	$A \neq d_j$	$A \in V_j$	$A \notin V_j$
$A = d_i$	$d_i = d_j$	$d_i \neq d_j$	$d_i \in V_j$	$d_i \notin V_j$
$A \neq d_i$	$\_$	$d_i = d_j$	$V_j = D \setminus \{d_i\}$	$V_j = \{d_i\}$
$A \in V_i$	$V_i = \{d_j\}$	$d_j \notin V_i$	$V_i \subseteq V_j$	$V_i \cap V_j = \emptyset$
$A \notin V_i$	$D \setminus V_i = \{d_j\}$	$V_i = \{d_j\}$	$V_j = D \setminus V_i$	$V_j \subseteq V_i$

Table 2 the conditions are *satisfactory* ones. However, it is important to note that in case of the first rows of the tables (the cases of $A = d_i$ and $A = V$, respectively) all the conditions are also *necessary* ones. The interpretation of the tables is straightforward: if an atomic formula in the leftmost column in some row i is true, then the atomic formula in the topmost row in some column j is also true, provided that the relation indicated on intersection of row i and column j is true. The rules of Table 1 and Table 2 can be used for checking if preconditions of a formula hold or verifying subsumption among rules.

4.3 RULES IN ALSV(FD)

ALSV(FD) has been introduced with practical applications for rule languages in mind. In fact, the primary aim of the presented language is to extend the notational possibilities and expressive power of the XTT-based tabular rule-based systems [9]. An important extension consist in allowing for explicit specification of one of the symbols $=, \neq, \in, \notin, \subseteq, \supseteq$, sim and $\not\sim$ with an argument in the table.

Consider a set of n attributes $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$. Any rule is assumed to be of the form:

$$(A_1 \propto_1 V_1) \wedge (A_2 \propto_2 V_2) \wedge \ldots (A_n \propto_n V_n) \longrightarrow RHS$$

where $\propto_i$ is one of the admissible relational symbols in ALSV(FD), and RHS is the right-hand side of the rule covering conclusion and perhaps the retract and assert definitions if necessary; for details see [9].

Knowledge representation with eXtended Tabular Trees (XTT) incorporates extended attributive table format. Further, similar rules are grouped within separated tables, and the whole system is split into such tables linked by arrows representing the control strategy. Consider a set of m rules incorporating the same attributes $A_1, A_2, \ldots, A_n$. In such a case the preconditions can be grouped together and form a regular matrix. Together with the conclusion part this can be expressed as in Tab. 3

Table 3. A general scheme of an XTT table

Rule	A_1	A_2	$\ldots$	A_n	H
1	$\propto_{11}\ t_{11}$	$\propto_{12}\ t_{12}$	$\ldots$	$\propto_{1n}\ t_{1n}$	h_1
2	$\propto_{21}\ t_{21}$	$\propto_{22}\ t_{22}$	$\ldots$	$\propto_{2n}\ t_{2n}$	h_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
m	$\propto_{m1}\ t_{m1}$	$\propto_{m2}\ t_{m2}$	$\ldots$	$\propto_{mn}\ t_{mn}$	h_m

In Table 3 the symbol $\propto_{ij} \in \{=, \neq, \in, \notin\}$ for simple attributes and $\propto_{ij} \in \{=, \neq, \subseteq, \supseteq, \sim, \not\sim\}$ for the generalized ones. In practical

Table 2. Inference rules for atomic formulae for generalized attributes

$\models$	$A = W$	$A \neq W$	$A \subseteq W$	$A \supseteq W$	$A \sim W$	$A \not\sim W$
$A = V$	$V = W$	$V \neq W$	$V \subseteq W$	$V \supseteq W$	$V \cap W \neq \emptyset$	$V \cap W = \emptyset$
$A \neq V$	_	$V = W$	$W = D$	_	$W = D$	_
$A \subseteq V$	_	$V \subset W$	$V \subseteq W$	_	$W = D$	$V \cap W = \emptyset$
$A \supseteq V$	_	$W \subset V$	$W = D$	$V \supseteq W$	$V \cap W \neq \emptyset$	_
$A \sim V$	_	$V \cap W = \emptyset$	$W = D$	_	$V = W$	_
$A \not\sim V$	_	$V \cap W \neq \emptyset$	$W = D$	_	$W = D$	$V = W$

applications, however, the most frequent relation are $=$, $\in$, and $\subseteq$, i.e. the current values of attributes are *restricted* to belong to some specific subsets of the domain. If this is the case, the relation symbol can be omitted (i.e. it constitutes the default relation which can be identified by type of the attribute and the value).

The current values of all the attributes are specified with the contents of the knowledge-base (including current sensor readings, measurements, inputs examination, etc.). From logical point of view it is a formula of the form:

$$(A_1 = S_1) \wedge (A_2 = S_2) \wedge \ldots \wedge (A_n = S_n), \qquad (13)$$

where $S_i = d_i$ ($d_i \in D_i$) for simple attributes and $S_i = V_i$, ($V_i \subseteq D_i$) for complex.

Having a table with defined rules the execution mechanism searches for ones with satisfied preconditions. The satisfaction of preconditions is verified in an algebraic mode, using the dependencies specified in the first row of Table 1 for simple attributes and the first row of Table 2 for the complex ones.

The rules having all the preconditions satisfied can be fired. In general, rules can be fired in parallel (at least in theory) or sequentially. For the following analysis we assume the classical, sequential model, i.e. the rules are examined in turn in the top-down order and fired if the preconditions are satisfied. Various mechanisms can be used to provide a finer inference control mechanism [9].

4.4 ATTRIBUTE DOMAINS

It is assumed that for each XTT attribute a *type* has to be stated. A type is named and it specifies: the base type and the domain. In the design of the XTT+ attributive language the set of base types was introduced in a way that simplifies the low-level interpreter integration with Prolog, Java, and RDBMS.

An example definition could be as follows:

- suppose we have a natural language specification: "some temperature",
- create attribute type,
  - pick a attribute type name, e.g. "Temperature"
  - decide what base type to use, in this case it could be a float,
  - define the domain by specifying constraints, e.g. -100, 100 depending on the conditions,
  - decide whether the domain is ordered – in case of symbolic base type, in this case numbers are ordered,
- create new attribute, with given,
  - attribute type, in this case of "Temperature",
  - name, e.g. sensor_temperature,
  - decide whether the attribute can take only single values, or also multiple values.

Generalized attributes are unordered or ordered sets (lists). This means attributes are in fact *multi-valued*. Some applications for this features are e.g.: a set of languages a person speaks, or storing subsequent values (changing in time).

4.5 RULE FIRING

The XTT+ rule firing process is coherent with the regular RBS semantics. It involves: condition checking and decision execution.

The condition checking can be described as a pattern matching process, where the condition evaluates true or false. The condition is an expression in the CNF build of expressions in the ALSV(FD).

The decision execution is where actions are possible. In a general case, the XTT+ rule decision involves: attribute value change context switching through inference control links event triggering. In XTT it is assumed, that the *whole* system state is described by the means of attributes.

5 PROTOTYPE IMPLEMENTATION EXAMPLE

In the prototype implementation of the knowledge base, rules and the interpreter are developed in PROLOG. A meta-programming approach is followed. This allows for encoding virtually any structured information. Note that in such a case the built-in PROLOG inference facilities cannot be used directly, there is a need for a meta-interpreter (however, this gives more flexibility in terms of rule processing).

Example domains and attributes specification in PROLOG follows:

```
domain(d7,[1,2,3,4,5,6,7]).
attribute(aDN,atomic,d7).
attribute(sDN,set,d7).
```

The atomic formulae (facts) are represented as terms of the type `fact/4` with four arguments; here are some examples:

```
%%% fact(<attribute-type>,<attribute-name>,
%   <relation>,<attribute-domain>)
fact(atomic,aDN,eq,7).
fact(atomic,aDD,in,[monday,wednesday,friday]).
fact(set,sDD,sim,[monday,wednesday,friday]).
fact(set,sSE,subseteq,[spring,summer,autumn]).
```

Facts are used mostly in rule preconditions. The meaning of the above facts is as follows: `f1: sDN=7`, `f2: aDD`$\in$`[monday,wednesday,friday]`, `f3: sDD`$\sim$`[monday, wednesday,friday]`, and `f4: sSE`$\subseteq$`[spring,summer,autumn]`. PROLOG list are used to represent set values.

The state of the system is represented by all the facts true in that state. Recall that the form $A = d$ and $A = V$ are allowed for state specification.

```
%%% state(<state-identifier>,
%   <attribute>,<value>,<type>).
state(s17,aDD,atomic,friday).
state(s17,aSE,atomic,spring).
state(s17,sDN,set,[1,3,5,7]).
```

Note that using set values in state specification increases drastically the expressive power. This is a bit similar to the Cartesian Product: in state `s17` the attribute `sDN` takes all the values from `[1,3,5,7]`.

Inference, i.e. checking logical consequence defined by first rows of Table 1 and Table 2 is performed with the `valid/s` predicate defined as follows:

```
valid(f(atomic,A,eq,Value),State):-
   state(State,A,atomic,StateValue),
   Value ==  StateValue,!.
valid(f(atomic,A,neq,Value),State):-
   state(State,A,atomic,StateValue),
   Value =\= StateValue,!.
valid(f(atomic,A,in,SetValue),State):-
   state(State,A,atomic,StateValue),
   member(StateValue,SetValue),!.
valid(f(atomic,A,notin,SetValue),State):-
   state(State,A,atomic,StateValue),
   \+member(StateValue,SetValue),!.
valid(f(set,A,eq,SetValue),State):-
   state(State,A,set,StateValue),
   eqset(SetValue,StateValue),!.
valid(f(set,A,neq,SetValue),State):-
   state(State,A,set,StateValue),
   neqset(SetValue,StateValue),!.
valid(f(set,A,subseteq,SetValue),State):-
   state(State,A,set,StateValue),
   subset(SetValue,StateValue),!.
valid(f(set,A,supseteq,SetValue),State):-
   state(State,A,set,StateValue),
   subset(StateValue,SetValue),!.
valid(f(set,A,sim,SetValue),State):-
   state(State,A,set,StateValue),
   intersect(SetValue,StateValue,[_|_]),!.
valid(f(set,A,notsim,SetValue),State):-
   state(State,A,set,StateValue),
   intersect(SetValue,StateValue,[]),!.
```

The excerpt of the implementation code presented above includes only symbolic domains. The definitions for the remaining domains are similar to the ones presented here. Currently the use of CLP (*Constraint Logic Programming*) PROLOG extensions are being investigated.

6 CONCLUDING REMARKS

Providing an expressive yet formally described rule language is of a high importance for practical rule design and implementation. This paper presents extensions of Set Attributive Logic as presented in [9]. In the proposed logic both atomic and set values are allowed and various relational symbols are used to form atomic formulae. The proposed language provides a concise and elegant tool of significantly higher expressive power than in case of classical attribute logic. It can be applied for design, implementation and verification of rule-based systems.

In the paper new inference rules specific for the introduced logic are presented and examined. New inference possibilities constitute a challenge for efficient precondition matching algorithm. Algebraic solutions are proposed. Knowledge representation and some excerpt from inference engine implemented in PROLOG is described. Components of a rule-based system in form of extended attributive decision tables (the so-called XTT paradigm) are presented and their characteristics and applications are outlined.

Future work includes a more robust implementation of the type system, tighter integration with a Java-based runtime, as well as an interface do RDBMS.

ACKNOWLEDGEMENTS

The paper is supported by the Hekate Project funded from 2007–2009 resources for science as a research project.

References

[1] Mordechai Ben-Ari, *Mathematical Logic for Computer Science*, Springer-Verlag, London, 2001.

[2] Ivan Bratko, *Prolog Programming for Artificial Intelligence*, Addison Wesley, 3rd edn., 2000.

[3] Michael A. Covington, Donald Nute, and André Vellino, *Prolog programming in depth*, Prentice-Hall, 1996.

[4] Michael R. Genesereth and Nils J. Nilsson, *Logical Foundations for Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., Los Altos, California, 1987.

[5] Adrain A. Hopgood, *Intelligent Systems for Engineers and Scientists*, CRC Press, Boca Raton London New York Washington, D.C., 2nd edn., 2001.

[6] Peter Jackson, *Introduction to Expert Systems*, Addison–Wesley, 3rd edn., 1999. ISBN 0-201-87686-8.

[7] *Handbook of Data Mining and Knowledge Discovery*, eds., Willi Klösgen and Jan M. Żytkow, Oxford University Press, New York, 2002.

[8] *The Handbook of Applied Expert Systems*, ed., Jay Liebowitz, CRC Press, Boca Raton, 1998.

[9] Antoni Ligęza, *Logical Foundations for Rule-Based Systems*, Springer-Verlag, Berlin, Heidelberg, 2006.

[10] Antoni Ligęza and Grzegorz J. Nalepa, 'Knowledge representation with granular attributive logic for XTT-based expert systems', in *FLAIRS-20 : Proceedings of the 20th International Florida Artificial Intelligence Research Society Conference : Key West, Florida, May 7-9, 2007*, eds., David C. Wilson, Geoffrey C. J. Sutcliffe, and FLAIRS, pp. 530–535, Menlo Park, California, (may 2007). Florida Artificial Intelligence Research Society, AAAI Press.

[11] Grzegorz J. Nalepa and Antoni Ligęza, 'A graphical tabular model for rule-based logic programming and verification', *Systems Science*, **31**(2), 89–95, (2005).

[12] Grzegorz J. Nalepa and Igor Wojnicki, 'Proposal of visual generalized rule programming model for Prolog', in *17th International conference on Applications of declarative programming and knowledge management (INAP 2007) and 21st Workshop on (Constraint) Logic Programming (WLP 2007) : Wurzburg, Germany, October 4–6, 2007 : proceedings : Technical Report 434*, eds., Dietmar Seipel and et al., pp. 195–204, Wurzburg : Bayerische Julius-Maximilians-Universitat. Institut für Informatik, (september 2007). Bayerische Julius-Maximilians-Universitat Wurzburg. Institut für Informatik.

[13] Grzegorz J. Nalepa and Igor Wojnicki, 'Visual software modelling with extended rule-based model : a knowledge-based programming solution for general software design', in *ENASE 2007 : proceedings of the second international conference on Evaluation of Novel Approaches to Software Engineering : Barcelona, Spain, July 23–25, 2007*, eds., Cesar Gonzalez-Perez and Leszek A. Maciaszek, pp. 41–47. INSTICC Press, (july 2007).

[14] Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Prentice-Hall, 2nd edn., 2003.

[15] I. S. Torsun, *Foundations of Intelligent Knowledge-Based Systems*, Academic Press, London, San Diego, New York, Boston, Sydney, Tokyo, Toronto, 1995.

Defining a subset of OCL for expressing SWRL rules

Sergey Lukichev [1]

Abstract. OCL is a rich-syntax language for expressing integrity rules and many business rules can be expressed using OCL. On the other hand, UML/OCL is a mainstream modeling technology and adopting it for expressing rules in the upcoming Semantic Web is a good practical issue: rules can be initially expressed in UML/OCL and then mapped into OWL/SWRL. Since UML/OCL is more expressive than OWL/SWRL it is good to define a subset of OCL, which can easily be mapped into the SWRL. In this paper we define this subset of OCL, called OCL-Lite, by building a mapping from SWRL into OCL. In addition, we briefly sketch the correctness problem of this mapping in terms of language interpretations.

1 Introduction

There are two issues we have to motivate: *i*). Why the subset of OCL, which can be mapped into SWRL is needed? *ii*). Why we build the mapping from SWRL to OCL in order to define this subset?

UML/OCL [1] is widely used among software engineers, while the area of the Semantic Web lacks for modeling methodologies and tools, comparable in popularity and maturity with UML/OCL. Therefore, it is natural to reuse existing UML/OCL methodologies and tools for modeling Semantic Web rules, which are expressed in the Semantic Web Rule Language (SWRL) [3]. The first arising problem with mapping OCL constraints into SWRL rules is a rich OCL syntax. It allows variety of expressions and not all of them can be mapped into SWRL. There are a number of works on sharing rules between OWL/SWRL and UML/OCL [7], [5], [6]. However, these works are mainly focused on the technical side of the mapping, in particular on the implementation of the metamodel transformations using Query View Transformations (QWT). The main question, which remains open in these works is what is exact subset of OCL, which can be mapped into SWRL? In this work we bridge this gap by defining a subset of OCL, called OCL-Lite, which can be mapped into SWRL.

In order to define OCL-Lite, we build a mapping from SWRL to OCL. The expressivity of OCL is higher than the first order logic since closures over object relations can be expressed in OCL but not in FOL [4]. Moreover, the syntax of SWRL is simpler than OCL, therefore we build the mapping from the less expressive language to the more expressive one, which is a natural method in determining the mappable subset of OCL.

The paper is structured as follows: section 2 contains the mapping from SWRL to OCL, in section 3 the syntax of OCL-Lite is specified and section 4 defines the problem of the semantical correctness of SWRL to OCL mapping and lists open issues for the further research.

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

2 Translating SWRL rules into OCL invariants

First, we remind the SWRL abstract syntax, which is originally defined in [3]. By `I = Imp(Antec(A), Conseq(C))` we denote a SWRL rule where `A` and `C` are conjunctions of SWRL atoms. By `classAtom(CD, t)` we denote a SWRL class atom, where `CD` is a class description, as defined in [2] (Section 2.3.2.3. OWL DL Restrictions) and `t` is an object variable or an individual. By `individualPropertyAtom(p, o1, o2)` we denote a SWRL individual property atom, where `p` is a reference to a property, and `o1` and `o2` are object variables. By `A = datavaluedPropertyAtom(p, obj, val)` we denote a SWRL datatype property atom, where `p` is a reference to an datatype property, `obj` is an object variable and `val` is a data value or a data variable.

According to the semantics of SWRL [3], SWRL rules are integrity constraints in the form of logical implications.

Let `I = Imp(Antec(A), Conseq(C))` be a SWRL rule, then the mapping T transforms it into an OCL invariant as follows:

```
T(I) := context class(X) inv:
T(A) implies T(C)
```

The context class `X` of the resulting OCL invariant is class of a universally quantified variable in the rule. This variable then is renamed to **self**. If the rule has more than one universally quantified variable, then `X.allInstances()->forAll(y|e(y))` expression is used recursively, where `X` is a class of the universally quantified variable and `e(y)` is the OCL boolean expression, resulted from the SWRL rule.

In fact, using the SWRL syntax, it is possible to express a rule, which cannot be mapped into a syntactically correct OCL invariant. In particular, this is because SWRL supports data variables, which are not explicitly supported in OCL. For instance, if we have a conjunction of two SWRL atoms: `individualPropertyAtom(age, o, x)` and `builtinAtom(>, x, 18)`, where `x` is a data variable, then this corresponds to the OCL expression `o.age>18`, i.e. we have one OCL expression from the conjunction of two SWRL atoms. If the built-in atom is dropped, then the SWRL rule still remains syntactically correct, while the resulting OCL expression will have unknown symbol `x`. We do not define such possible mappings in this paper and focus on the direct mappings of SWRL atoms.

We do not describe how URIs are mapped into names of UML classifiers since this task is about the mapping from OWL to UML while we define the mapping of SWRL rules into OCL invariants.

The body and the head of a SWRL rule is a conjunction of SWRL atoms, therefore if `A = A1, ..., An`, then

```
T(A) := T(A1) and ... and T(An)
```

Let us consider the mapping of SWRL atoms.

- If `A = classAtom(id, t)` is a SWRL class atom, where `id` is an URI reference of the class and `t` is an object variable, then

```
T(A) := t.oclIsKindOf(id)
```

- If `A = individualPropertyAtom(p, o1, o2)` where `p` is a reference to an individual property, `o1` and `o2` are object variables, then

  ```
  T(A) := o1.p=o2
  ```

- If `A = datavaluedPropertyAtom(p, obj, val)` where `p` is a reference to an individual property, `obj` is an object variable and `val` is a data value or a data variable, then

  ```
  T(A) := obj.p=val
  ```

- If `A = builtinAtom(builtin, {obj})` where `builtin` is a SWRL builtin and `{obj}` is a list of datatype terms, then the resulting OCL expression is of type Boolean and depends on the built-in, for instance, if `builtin` is '+' then `A = builtinAtom('+', a,b,c)` and `T(A) := a=b+c`.

If `A = classAtom(CD, t)` is a SWRL class atom with an OWL description `CD`, then we define the mapping function case wise for each possible OWL class description `CD` (the syntax of OWL class descriptions is defined in [2]):

- If `CD = UnionOf(CD1, CD2)` where `CD1` and `CD2` are OWL class descriptions, then

  ```
  T(classAtom(UnionOf(CD1, CD2), t)) :=
  T(classAtom(CD1, t)) or T(classAtom(CD2,
  t))
  ```

  For instance, if `CD = UnionOf(Winery, Brewery)` then

  ```
  T(classAtom(CD, t)) := t.oclIsKindOf(Winery)
  or t.oclIsKindOf(Brewery)
  ```

- If `CD = IntersectionOf(CD1, CD2)` where `CD1` and `CD2` are OWL class descriptions, then

  ```
  T(classAtom(CD, t)) := T(classAtom(CD1, t))
  and T(classAtom(CD2, t))
  ```

- If `CD = ComplementOf(CD1)` where `CD1` is an OWL class descriptions, then

  ```
  T(classAtom(CD, t)) := not T(classAtom(CD1,
  t))
  ```

- If `CD = Restriction(oProp, allValuesFrom(CD1))` where `oProp` is a property and `CD1` is a class description, then

  ```
  T(classAtom(CD, t)) :=
  t.oProp->forAll(x|T(classAtom(CD1, x)))
  ```

  The variable `x` is unified with instances of the property collection `oProp` . For instance, if

  ```
  CD = Restriction(hasMaker,
                   allValuesFrom(Winery))
  ```

  then

  ```
  T(classAtom(CD, t)) :=
   t.hasMaker->forAll(x|
             T(classAtom(Winery, x))) :=
   t.hasMaker->forAll(x|
             x.oclIsKindOf(Winery))
  ```

- If `CD` is

  ```
  Restriction(oProp,
              someValuesFrom(CD1))
  ```

  where `oProp` is a property and `CD1` is a class description, then

  ```
  T(classAtom(CD, t)) :=
    t.oProp->exists(x|
             T(classAtom(CD1, x)))
  ```

- If `CD` is

  ```
  Restriction(dProp, value(d))
  ```

  where `dProp` is a datatype property and `d` is data value, then

  ```
  T(classAtom(CD, t)) := t.dProp=d
  ```

- If `CD` is

  ```
  Restriction(objProp, mincardinality(n))
  ```

  where `objProp` is a property and `n` is minimal cardinality, then

  ```
  T(classAtom(CD, t)) := t.objProp->size()>=n
  ```

  The mapping of restrictions with min- and max- cardinalities is similar.

- If `CD = OneOf({obj1,...,objn})` where `obj1,...,objn` are object URIs, then

  ```
  T(classAtom(CD, t)) := t=obj1 or ... or
  t=objn
  ```

  An example of the class atom with `oneOf` description

  ```
  CA := classAtom(oneOf(White, Red), t)
  ```

  Then

  ```
  T(CA) := t=White or t=Red
  ```

  An OWL enumerated class, which is specified by the direct enumeration of its instances, can be represented in UML by means of a UML enumeration class. Some additional mapping from OWL enumerated classes into UML enumeration classes may be needed in order to produce a syntactically correct OCL expression. However, this is the problem of UML to OWL mapping, which is beyond our rules mapping problem.

In order to show how the mapping works, let us consider a rule "*If the maker of a liquid is a winery, then the liquid is either a wine or a cognac*". The rule in SWRL is:

```
R=Implies(Antec(classAtom(Liquid, t)
  classAtom(Restriction(hasMaker,
               allValuesFrom(Winery)), t))
          Conseq(classAtom(
               UnionOf(Wine, Cognac), t)))
```

The mapping of `R` into OCL:

```
T(R):= context Liquid inv:
    T(classAtom(Restriction(hasMaker,
       allValuesFrom(Winery)), t))
    implies
    T(classAtom(UnionOf(Wine, Cognac), t)))
```

Which is finally:

```
context Liquid inv:
  self.hasMaker->forAll(x|
        x.oclIsKindOf(Winery))
implies
  self.oclIsKindOf(Wine) or
  self.oclIsKindOf(Congnac)
```

3 The syntax of OCL-Lite

The syntax of OCL-Lite expressions is defined recursively so that more complex expressions are built from simple ones. The definition below is based on the result of the mapping, defined in the previous section. In the syntax definition below we refer to OCL expressions, which are based on navigation operations, as *navigation expressions*. For instance, `self.father.brothers` is a navigation expression, which is not a new concept on top of OCL, but just a denotation of the existing OCL expression, also used in the OCL specification [1]. A navigation expression may result into a collection of instances. We introduce this denotation in order to simplify the OCL-Lite syntax by eliminating various OCL operations, which are not supported in SWRL.

1. **self** is a special variable, which type is given by the invariant context;
2. For every OCL model type t there is an unlimited number of variables v_t^i, which are OCL expressions of type t;

3. If f is an OCL operation symbol with argument types $t_1, ..., t_n$ and result type t_r, $e_1, ..., e_n$ are OCL expressions and type of e_i is t_i for all $1 \leq i \leq n$ then $f(e_1, ..., e_n)$ is OCL expression of type t_r. The set of operation symbols includes:
   - some of predefined data operations: $+, -, *$;
   - attribute operations, for instance, `self.age`, `e.salary`;
   - side-effect free operations defined by a class, for instance, `b.rentalDiscount();`
   - navigation by role names, for instance, `self.pickupBranch`.
4. If e_1 and e_2 are OCL expressions of an appropriate type (which provides an order relation, for instance, integer) then $e_1 > e_2$, $e_1 < e_2$, $e_1 >= e_2$, and $e_1 <= e_2$ are Boolean OCL expressions. Note, that symbols $>, <, >=$, and $<=$ are OCL operations in the OCL metamodel, however here we define OCL expressions with these symbols separately since logically they are predicate symbols, while expressions, defined in the previous item are different types of terms;
5. If e is an object variable, then $e.oclIsKindOf(C)$, where C is a class name, is a Boolean expression;
6. If e_1, e_2 are Boolean expressions then e_1 *and* e_2, e_1 *or* e_2, *not* e_1, e_1 *implies* e_2 are Boolean expressions;
7. If e_1, e_2 are OCL expressions of the same type, then $e_1 = e_2$, $e_1 <> e_2$ are Boolean expressions.
8. If $e(x)$ is a navigation expression, where variable x is bound by the quantifier, for instance, `self.employee`, then $e(x) \rightarrow size()$ *op* n is Boolean expression, where *op* is either $>, <, =, >=, <=$, or $<>$. We do not define an expression of the form $e(x).size()$, where e is of any OCL collection type, for instance, resulted from some operation on collections like *includesAll()* since they are not defined in SWRL. The defined expression of the form $e(x) \rightarrow size()$ *op* n is obtained via the mapping from the SWRL class atom with cardinality restriction.
9. If e is a navigation expression, for instance, `self.employee`, then $e \rightarrow forAll(x|e_1(x))$ and $e \rightarrow exists(x|e_1(x))$ are Boolean expressions, where $e_1(x)$ is a Boolean OCL expression from variable x, which is bound by universal quantifier or existential quantifier.

4 Conclusion and future work

In this paper we described the syntactic mapping from SWRL to OCL and on the result of this mapping defined the syntax of OCL-Lite, a simplified subset of the OCL syntax. OCL-Lite can be used for expressing rules, which later can be mapped into the SWRL. Some practical experiments with the implementation of this mapping are available at the REWERSE I1 website[2].

The main open issue in this work is the semantic correctness of the defined mapping. Having such mapping it is important to make sure that the set of models of a SWRL rule f coincides with the set of models of the OCL invariant $t(f)$. We describe the problem formally.

Let $\mathcal{I}$ be an interpretation of SWRL rules and $\models$ is a satisfaction relation between interpretations and SWRL rules and there are the following mappings:

- $t : S \rightarrow O$, which maps SWRL rules into OCL invariants. It is a syntactic transformation, which we have defined in this paper.
- $T : S_I \rightarrow O_I$, which maps SWRL interpretations into OCL interpretations.

The task is to prove that for any SWRL interpretation $\mathcal{J}$ and SWRL rule G if

$$\mathcal{J} \models G$$

then

$$T(\mathcal{J}) \models t(G)$$

The solution to this task is currently in our agenda and results will be published soon.

[2] REWERSE I1 Rule Translators: http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/15

REFERENCES

[1] Object Constraint Language (OCL), v2.0. OMG Final Adopted Specification.

[2] OWL Web Ontology Language Semantic and Abstract Syntax. W3C Recommendation 10 February 2004. http://www.w3.org/2004/OWL.

[3] SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission 21 May 2004. http://www.w3.org/Submission/SWRL/.

[4] Luis Mandel and Maria Victoria Cengarle. On the expressive power of ocl. In *FM '99: Proceedings of the Wold Congress on Formal Methods in the Development of Computing Systems-Volume I*, pages 854–874, London, UK, 1999. Springer-Verlag.

[5] Milan Milanovic, Dragan Gasevic, Adrian Giurca, Gerd Wagner, and Vladan Devedzic. On interchanging between owl/swrl and uml/ocl. In *Proceedings of the OCLApps Workshop*, pages 81–95, Genova, Italy, 2 October 2006.

[6] Milan Milanovic, Dragan Gasevic, Adrian Giurca, Gerd Wagner, and Vladan Devedzic. Sharing owl/swrl and uml/ocl rules. In *Proceedings of ACM/IEEE 9th International Conference on Model-Driven Engineering Languages and Systems (MODELS 2006)*, Genova, Italy, 1-6 October 2006.

[7] Milan Milanovic, Dragan Gasevic, Adrian Giurca, Gerd Wagner, and Vladan Devedzic. Towards sharing rules between owl/swrl and uml/ocl. In *Electronic Communications of European Association of Software Science and Technology*, 2007.

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

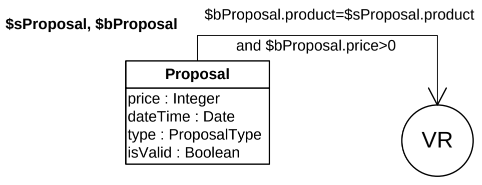


Figure 1. A rule condition

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`.

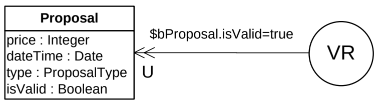
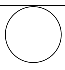
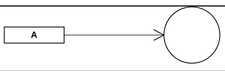
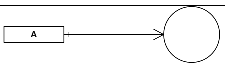
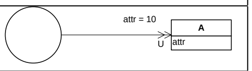


Figure 2. A rule action

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].

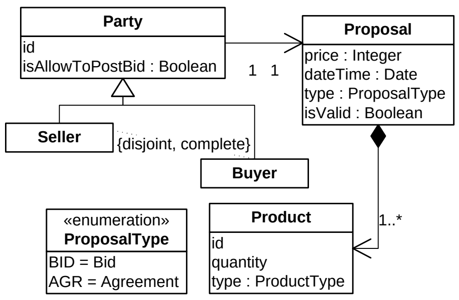


Figure 3. Negotiation Vocabulary

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.

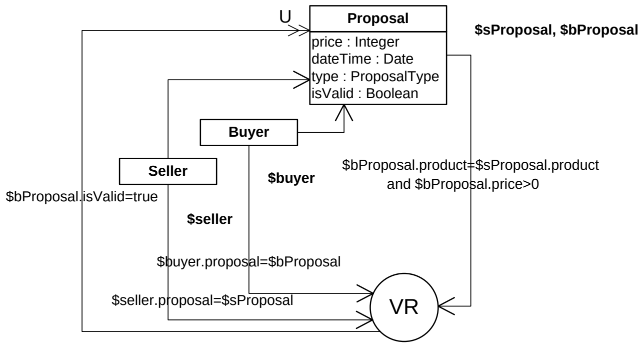


Figure 4. Validation Rule: *"A valid proposal is a proposal that is about the product from seller proposal and the submitted proposal price is greater than 0."*

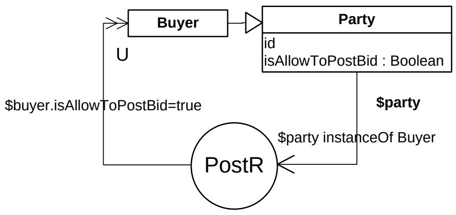
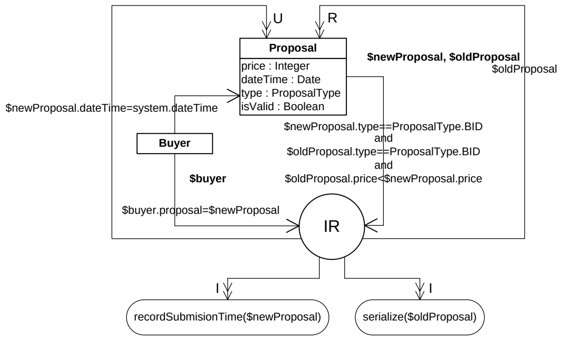


Figure 5. Posting Rule: *"Only buyers parties are allowed to post bids."*

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."*

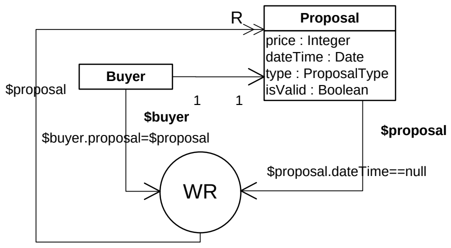


Figure 7. Withdraw Proposal Rule: *"Buyers proposals that are not best proposal have to be withdrawn."*

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.

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

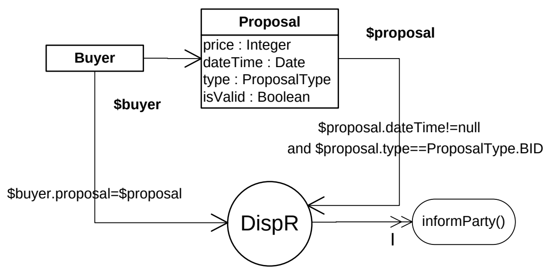


Figure 8. Display Proposal Rule: *"If a new proposal has been posted into the system then all parties must be informed about this."*

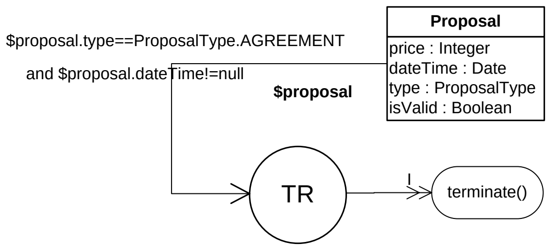


Figure 9. Termination Rule: *"If type of the current proposal from working memory is $AGREEMENT$ and $dateTime$ of proposal is not null then the auction can be terminated."*

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

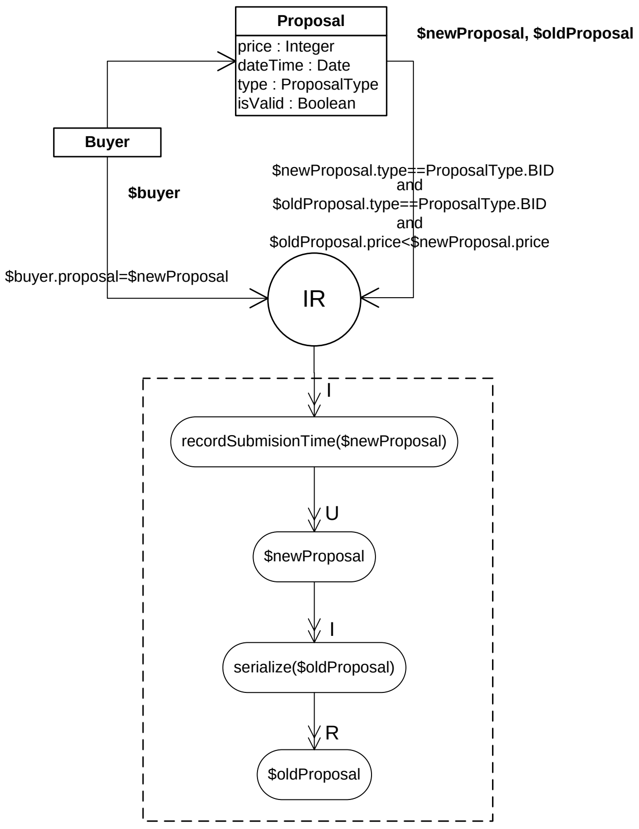
While the translation of all of these actions to Drools code is straightforward by using DRL functions one major disadvantage of the actual URML language is that it does not offer a way to specify the order of actions in the rule action part. For example, looking to the rule diagram from Figure 6 it 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`.