Modeling Web Services with URML

Adrian Giurca[1], Sergey Lukichev[1], and Gerd Wagner[1]

Institute of Informatics, Brandenburg University of Technology at Cottbus
{Giurca, Lukichev, G.Wagner}@tu-cottbus.de

Abstract. A Web service can be specified on the basis of a business vocabulary, including a business event model, and of a rule-based behavior model. We use the general rule language R2ML and the UML-based rule modeling language URML for modeling the behavior of Semantic Web services.

1 Introduction

A Web Service is a software application identified by a URI, whose interfaces and bindings are capable of being described in XML, e.g. by means of the Web Service Description Language (WSDL)[9] and which can directly interact with other software agents using XML-based messages (represented in the SOAP format [4]) exchanged via Internet protocols (HTTP or SMTP).

In this paper we discuss the modeling of Web Services with the help of the UML-Based Rule Modeling Language (URML)[6] developed by the REWERSE Working Group I1[1] for the semantic business process management on the Web. In order to support rule interchange between different rule engines, we have developed a general rule language, called R2ML ([10], [11]), with an XML serialization format. The metamodels of URML and R2ML largely overlap. URML can be considered as a language that is derived from R2ML in order to provide UML-based rule modeling.

We envision the following business process modeling scenario:

- A business process modeler models Web services based on a vocabulary and rules with a rule modeling tool, f.e. Strelka ([3]);
- The entire Web service specification can be serialized using WSDL and the rule description language R2ML.

A rule-based Web service may be implemented using a reaction rule engine and a SOAP listener. The rule engine receives SOAP messages, executes triggered rules and performs actions. The SOAP listener is a Web application, which captures SOAP messages via HTTP(S) and passes them to the rule engine.

In Section 2 we describe R2ML reaction rules, in Section 3 we describe the R2ML event metamodel with the focus on atomic events, represented by SOAP messages, in Section 4 we describe R2ML actions, and in Section 5 we give an example of a reaction rule, modeled with URML and serialized into R2ML. In Section 6 we give an outline of the future work and conclusions.

[1] Working Group I1 http://www.rewerse.net/I1

2 Reaction Rules

The main goal of R2ML is to provide a representation of rules, targeted to the rule engine platform that is independent of a vendor specific engine (PIM Level). This allows R2ML to be a rule interchange language between different PSM level specific rule engines. This section presents R2ML *reaction rules* also known as *Event-Condition-Action rules* (*ECA rules*) and their usage in describing business process modeling.

There are several advantages of using REACTION rules for specifying business processes: business requirements are often captured in the form of rules in a natural language, formulated by business people; REACTION rules are easier to maintain and integrate with other kinds of rules, used in business applications (integrity rules, which specify constraints the data must fulfill, derivation rules, which explain how a model element can be derived); the topic of rules validation and verification is well-studied; REACTION rules emphasis on events gives a flexible way to specify control flow.

The R2ML metamodel for reaction rules is depicted in Figure 1.

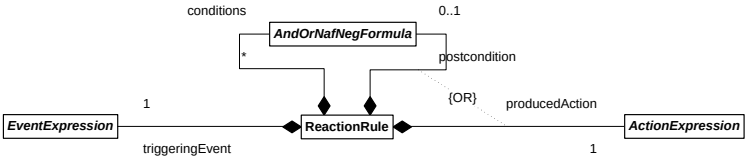

Fig. 1. Reaction rule metamodel

A *reaction rule* is a statement of programming logic that specifies the execution of one or more actions in the case of a *triggering event* occurrence and if rule *conditions* are satisfied. Optionally, after the action execution *post-conditions* may be made true.

Reaction rules therefore have an operational semantics (formalizing state changes, e.g., on the basis of a state transition system formalism).

A reaction rule has the following components:

- `triggeringEvent` is an R2ML EventExpression, which is either atomic or composite (Figure 2);
- `conditions` are represented as a collection of quantifier free logical formulas;
- `producedAction` is an R2ML action, which represents the state change of the system. The latest version of R2ML defines composite actions, which are, for instance, sequential actions and parallel actions.
- an optional `postcondition` specifies a state change in a declarative manner.

All components of a reaction rule contain expressions that refer to rule variables. The R2ML distinguishes between object variables, which are instantiated

with objects, and data variables, which are instantiated with data values. The rule variables are bound to specified classes/datatypes. An object variable in R2ML can be bound to a specific class either using an `ObjectClassificationAtom` or the optional attribute `classID` of the variable. Similarly, a data variable can be bound using a `DataClassificationAtom` or the optional attribute `datatypeID` of the variable.

3 R2ML Events Metamodel

The R2ML Events metamodel (see Figure 2) specifies the core concepts, which are necessary for dynamic behavior of rules and provides the infrastructure for the support of more detailed behavior definition.

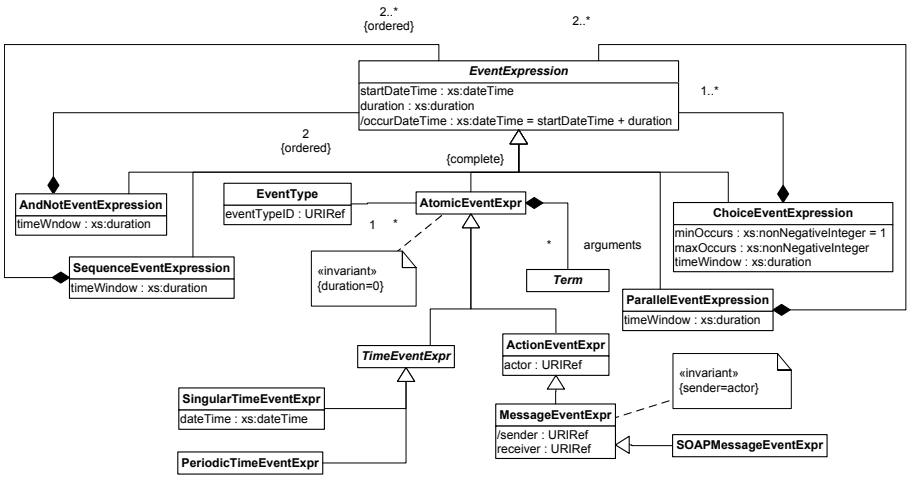

Fig. 2. R2ML Event Expressions

The basic properties of an *R2ML event expression* are:

- `startDateTime` is an event start date and time;
- `duration` is a value specification that specifies the temporal distance between two time expressions, which define time instants;
- `occurDateTime` is a derived property, which is given by the addition of duration to the existent start date time.

All R2ML Events are subclasses of `EventExpression`. `EventExpression` is either a composite event or an atomic event.

3.1 Composite event

Composite event in R2ML is either an AndNotEventExpression, a SequenceEventExpression, a ParallelEventExpression and a ChoiceEventExpression. Each event expression has a property `timeWindow`, which represents the duration of the corresponding event observation. The event expression metamodel is depicted in Figure 2.

AndNotEventExpression has two event expressions as arguments (`EvtExpr1` and `EvtExpr2`). It describes a complex event where an instance of `EvtExpr1` but no instance of `EvtExpr2` occurs.

SequenceEventExpression refers to an ordered list of event expressions, which are processed in a sequence of events, following the existent order and considering a finite value of `timeWindow` observation.

ParallelEventExpression refers to a collection of events that are concurrently processed inside of the corresponding `timeWindow`.

ChoiceEventExpression refers to a collection of events that requires processing of at least one event expression from the collection inside of the corresponding `timeWindow`.

3.2 Atomic event

Atomic event in R2ML is an *AtomicEventExpression*, which main characteristic is that it has no duration (`duration = 0`). As a consequence, the occurrence date time is the same as the start date time.

An atomic event expression:

- Refers to an EventType, which is its classifier;
- Is composed from an ordered, possible empty, list of terms as arguments.

The R2ML distinguishes between two main classes of atomic events: `MessageEventExpression` and `TimeEventExpression`.

Message event expression has a property `sender`. In the discussing approach for business process modeling in web services, a *sender* may be HTTP_REFERER. One category of message event is a SOAP message event.

SOAP Messages Events. SOAP is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics[4].

We use SOAP messages as transport containers for events, which are expressed in R2ML as `SOAPMessageEventExpr`'s (see Figure 3). SOAP is typically used for Remote Procedure Calls (RPC). The SOAP specification[4] defines two special message formats: a SOAP RPC Request Message, represented in R2ML by `SOAP-RPC-RequestMsgEvtExpr` and a SOAP RPC Response Message, represented in R2ML by `SOAP-RPC-ResponseMsgEvtExpr`.

The following example shows a sample SOAP message, which contains an R2ML SOAP RPC request. The message contains two pieces of application-defined data not defined by the SOAP specification: a SOAP header block and a body element with a local name of `ref`. In general, SOAP header blocks contain information which might be of use to SOAP intermediaries as well as the ultimate destination of the message.

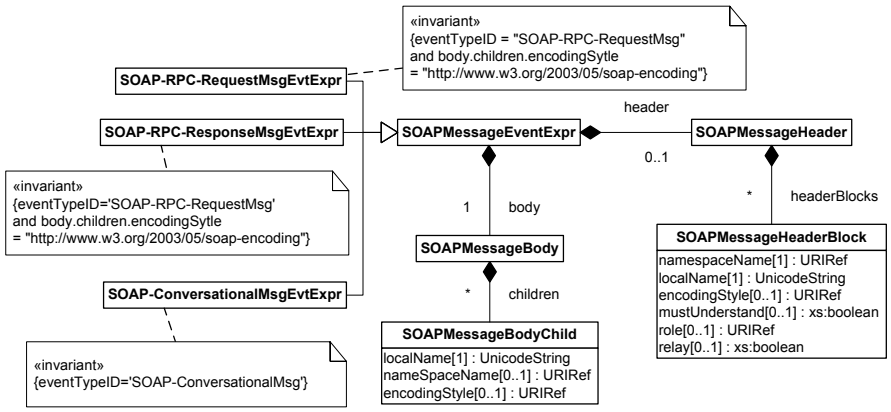

Fig. 3. SOAP message event expression in R2ML

In this example an intermediary might prioritize the delivery of the message based on the priority and expiration information in the SOAP header block. The body contains the actual event payload, in this case the customer's request for a car .

Example 1 (SOAP RPC Request).

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Body>
 <r2ml:SOAP-RPC-RequestMsgEvtExpr xmlns:rew="http://www.rewerse.net/I1/R2ML"
      r2ml:sender="eshop.com"
      r2ml:startTime="2006-03-21T09:00:00"
      r2ml:duration="P0Y0M0DT0H0M0S"
      r2ml:eventTypeID="productOrder">
1    <r2ml:arguments>
```

```
2     <r2ml:ObjectVariable r2ml:name="car" r2ml:classID="srv:Car"/>
3    </r2ml:arguments>
4    </r2ml:SOAP-RPC-RequestMsgEvtExpr>
   </env:Body>
  </env:Envelope>
```

The body of the SOAP is an R2ML SOAP-RPC-RequestMsgEvtExpr. Lines 1-3 define a list of arguments for the event: the ObjectVariable in line 2 is a variable `car` of type `srv:Car`, which is a particular car, requested by the customer.

4 Actions

The R2ML supports both production rules and reaction rules. With this respect it defines the concept of an *action*. Following the OMG Production Rule Representation submission[2], an action (Fig. 4) is either an *InvokeActionExpression* or an *AssignActionExpression* or a *CreateActionExpression* or a *DeleteActionExpression*. The R2ML provides also message actions in the form of a concrete *SOAPMessageEventExpr*.

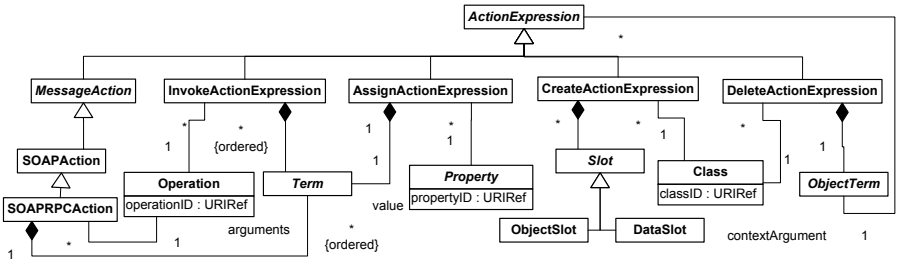

Fig. 4. Actions

All actions refer to a *context* which is an R2ML object term.

InvokeActionExpression models an object operation invocation. It refers to a UML *Operation* and contains an ordered, possible empty list of arguments represented as R2ML terms. The execution of this action is done by the corresponding operation-call.

Example 2 (InvokeActionExpression).

"Calculate the total payment of the purchase order."

```
<InvokeActionExpression r2ml:operationID="totalPayment">
 <contextArgument>
  <ObjectVariable r2ml:name="purchaseOrder" r2ml:classID="Order"/>
 </contextArgument>
</InvokeActionExpression>
```

In this example the operation `totalPayment` has no arguments.

AssignActionExpression refers to a UML *Property* and contains a DataTerm as a *value*. This action assigns a value to a property.

Example 3 (AssignActionExpression).

> "Set to `10` the property `discount` of the object variable `purchaseOrder` (`purchaseOrder.discount = 10`)."

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

CreateAction refers to a UML *Class* and contains a list of slots (object slots and/or data slots). The execution of this action consist in a constructor-call for creation of a new object in the system.

Example 4 (CreateActionExpression).

> "Create purchase order for one book named 'Harry Potter' with the price 11.25 and discount 10

```
<CreateActionExpression r2ml:classID="Order">
 <contextArgument>
   <ObjectVariable r2ml:name="purchaseOrder"/>
 </contextArgument>
 <DataSlot r2ml:attributeID="title">
  <TypedLiteral r2ml:lexicalValue="Harry Potter"
                r2ml:type="xs:string"/>
 </DataSlot>
 <DataSlot r2ml:attributeID="price">
   <TypedLiteral r2ml:lexicalValue="11.25"
                 r2ml:type="xs:float"/>
 </DataSlot>
 <DataSlot r2ml:attributeID="discount">
   <TypedLiteral r2ml:lexicalValue="10"
                 r2ml:type="xs:positiveInteger"/>
 </DataSlot>
</CreateActionExpression>
```

DeleteActionExpression refers to an UML *Class* and contains an *ObjectTerm*. This action removes an instance of the *Class*.

Example 5 (DeleteActionExpression).

Delete order `puchaseOrder`.

```
<DeleteActionExpression r2ml:classID="">
 <contextArgument>
   <ObjectVariable r2ml:name="purchaseOrder" r2ml:classID="Order"/>
 </contextArgument>
</DeleteActionExpression>
```

5 Business Process Modeling Example

Let's consider a part of a business process when a customer makes a request for a book from a web site. The customer fires an event, which is captured by the server. The server searches for an appropriate rule for this event and checks rule condition: whether the requested book is available or not. If the condition holds, i.e. the book is available, then it performs an action: approve order. The rule postcondition is that the amount of books in stock must be less by a requested quantity than before the rule execution. A part of the business vocabulary is depicted on Figure 5. The rule is modeled using a URML[6].

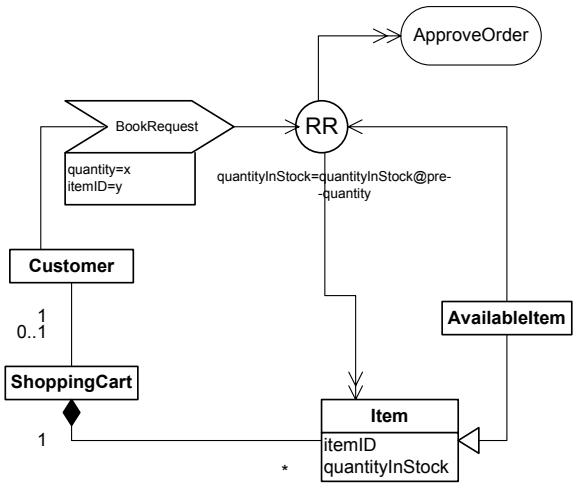

Fig. 5. On customer book request, if the book is available, then approve order and decrease amount of books in stock.

The business vocabulary consists of a customer, which may have a shopping cart. A shopping cart consists of items. An item has an *itemID* and *quantityIn-Stock*. There is an item category *AvailableItem*, which contains items, available

for the order and delivery. In the URML a rule is represented as a circle with a label "RR". Incoming arrow from AvailableItem is a rule condition. SOAP RPC request message is represented with a UML signal sign and connects a customer, who fires the event and a rule circle. The event contains list of parameters: *itemID* and *quantity*. Outgoing double-head arrow to an activity *ApproveOrder* is an action to approve the order. This action is defined in the WSDL interface of the service. Outgoing double-head arrow to an *Item* class represents rule postcondition with an OCL expression `quantityInStock=quantityInStock@pre-quantity`, that states that the amount of items in stock must be less by 1.

In order to be processed by the ECA engine, this rule should be serialized into rule interchange format R2ML. Corresponding R2ML syntax is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<r2ml:ReactionRuleSet xmlns:dc="http://purl.org/dc/elements/1.1/"
                      xmlns:rew="http://www.rewerse.net/I1/R2ML"
                      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                      xsi:schemaLocation="http://www.rewerse.net/I1/R2ML"
                      r2ml:id="ID000000">
    <r2ml:ReactionRule r2ml:id="rr1111" xmlns:srv="http://www.example.org/">
        <r2ml:RuleText r2ml:ruleDiagram="" r2ml:textFormat="text/xml"
                                          r2ml:ruleVocabularyDiagram=""/>
        <r2ml:SourceCode r2ml:language="R2ML"/>
        <dc:subject>Reaction rules, R2ML, Markup Languages</dc:subject>
        <r2ml:triggeringEvent>
            <r2ml:SOAPMessage r2ml:sender="" r2ml:startTime="2006-03-21T09:00:00"
                              r2ml:duration="P0Y0M0DT0H0M0S"
                              r2ml:eventTypeID="productOrder">
                <r2ml:arguments>
                    <r2ml:ObjectVariable r2ml:name="x"
                                         r2ml:classID="srv:Item"/>
                    <r2ml:DataVariable r2ml:name="quantity"
                                       r2ml:dataTypeID="xsd:integer"/>
                    <r2ml:ObjectVariable r2ml:name="customre1"
                                         r2ml:classID="srv:Customer"/>
                </r2ml:arguments>
            </r2ml:SOAPMessage>
        </r2ml:triggeringEvent>
        <r2ml:conditions>
            <r2ml:ObjectClassificationAtom r2ml:classID="srv:AvailableItem">
              <r2ml:ObjectVariable r2ml:name="x"/>
            </r2ml:ObjectClassificationAtom>
1           <r2ml:EqualityAtom>
              <r2ml:AttributeFunctionTerm r2ml:attributeID="srv:quantityInStock">
                <r2ml:contextArgument>
                  <r2ml:ObjectVariable r2ml:name="i" r2ml:classID="srv:Item"/>
                </r2ml:contextArgument>
              </r2ml:AttributeFunctionTerm>
              <r2ml:DataVariable r2ml:name="q" r2ml:dataTypeID="xsd:integer"/>
            </r2ml:EqualityAtom>
        </r2ml:conditions>
        <r2ml:producedAction>
            <r2ml:SOAPRPCAction r2ml:operationID="approveOrder">
                <r2ml:contextArgument>
                    <r2ml:ObjectVariable r2ml:name="x"/>
                </r2ml:contextArgument>
                <r2ml:arguments>
                    <r2ml:ObjectName r2ml:objectID="customer1"
                         r2ml:classID="srv:Customer"/>
                </r2ml:arguments>
            </r2ml:SOAPRPCAction>
        </r2ml:producedAction>
        <r2ml:postcondition>
2         <r2ml:EqualityAtom>
3           <r2ml:AttributeFunctionTerm r2ml:attributeID="srv:quantityInStock">
```

```
            <r2ml:contextArgument>
              <r2ml:ObjectVariable r2ml:name="i" r2ml:classID="srv:Item"/>
            </r2ml:contextArgument>
          </r2ml:AttributeFunctionTerm>
4         <r2ml:DataOperationTerm r2ml:operationID="minus">
            <r2ml:contextArgument>
              <r2ml:ObjectVariable r2ml:name="i" r2ml:classID="srv:Item"/>
            </r2ml:contextArgument>
            <r2ml:arguments>
              <r2ml:DataVariable r2ml:name="q"/>
              <r2ml:DataVariable r2ml:name="quantity"/>
            <r2ml:arguments>
          </r2ml:DataOperationTerm>
        </r2ml:EqualityAtom>
      </r2ml:postcondition>
  </r2ml:ReactionRule>
</r2ml:ReactionRuleSet>
```

It is important to note, that the postcondition expression `quantityInStock=quantityInStock@pre-quantity` is represented as combination of 2 atoms: equality atom in the condition part of a rule (line 1), which is considered as a variable `q` initialization with initial value of the attribute `quantityInStock` and equality atom in the postcondition part of the rule (line 2), which is considered as an assignment of a new value for the attribute `quantityInStock`. The new value is specified by the DataOperationTerm (line 4) with "minus" operation on old attribute value `q` and quantity `quantity`. For more rule examples in URML and R2ML we refer to the Working Group I1 web site and, in particular, to the EU-Rent case study[2], which contains a domain model and rules, modeled using URML and to the R2ML example rule set[3].

The transformation of URML model into R2ML is implemented in the Strelka tool. Metamodels of R2ML and URML largely overlap and URML concepts like rule, condition and conclusion can be directly mapped into R2ML. So called OCL filter expressions, used in URML conditions and postconditions to filter instances of a conditioned classifier (f.e. class or association), can be represented in the R2ML since it has corresponding functional atoms for representing OCL expressions. Since R2ML does not support collections yet, not all OCL expressions can be serialized into R2ML.

6 Conclusion

In this paper we have shown how a UML-Based Rule Modeling Language can be used for the modeling of Web Services, based on reaction rules. We have also presented a part of the R2ML language, related to reaction rules and gave examples of rule modeling in XML syntax of R2ML.

Concerning the future work on this topic we consider the following issues:

[2] EU-Rent Case Study in URML, using Strelka tool: `http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/12`

[3] R2ML project page: `http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/6`

– Give a focus on rule sets and respective control flow modeling in URML. Control flow modeling by means of reaction rules has been already introduced in [5] and we are going to adopt it in URML;
– Analyze the suitability of R2ML for expressing control flow patters, identified, for instance, by Van der Aslst et al. [7]. Control flow patterns representation by means of reaction rules has been specified in [8] and we have to investigate how they can be captured in reactive rules part of R2ML;
– R2ML needs a mechanism to specify exceptions.
– The issue of web service composition is currently under consideration.

References

1. Gelfond, M., Lifschitz, V., The stable model semantics for logic programming, In Proc. of ICLP-88, pp. 1070-1080.
2. W3C Workgroup on RIF Charter, http://www.w3.org/2005/rules/wg/charter
3. Strelka - A UML-Based Visual Rule Modeling Tool. http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/10
4. SOAP Version 1.2 Part 1: Messaging Framework W3C Recommendation 24 June 2003, http://www.w3.org/TR/soap12-part1/
5. Wagner G., The Agent-Object-Relationship Meta-Model: Towards a Unified View of State and Behavior. Information Systems 28:5 (2003), pp. 475-504.
6. A UML-Based Rule Modeling Language (URML) on REWERSE Working Group I1 website: http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=node/7
7. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and Parallel Databases 14(1) (2003)
8. Taveter, K.: A multi-perspective methodology for agent-oriented business modelling and simulation. PhD thesis, Tallinn University of Technology, Estonia, 2004 (ISBN 9985-59-439-8)
9. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C Candidate Recommendation 27 March 2006 http://www.w3.org/TR/wsdl20
10. G. Wagner, A.Giurca, S. Lukichev (2005). R2ML: A General Approach for Marking up Rules, Dagstuhl Seminar Proceedings 05371, in F. Bry, F. Fages, M. Marchiori, H. Ohlbach (Eds.) Principles and Practices of Semantic Web Reasoning, http://drops.dagstuhl.de/opus/volltexte/2006/479/
11. Wagner, G., Giurca, A., Lukichev, S. (2006). A Usable Interchange Format for Rich Syntax Rules. Integrating OCL, RuleML and SWRL. Will appear in proceedings of Reasoning on the Web Workshop at WWW2006, May 2006