I1-D10

Tool Improvements and Extensions 1

Project title:	Reasoning on the Web with Rules and Semantics
Project acronym:	REWERSE
Project number:	IST-2004-506779
Project instrument:	EU FP6 Network of Excellence (NoE)
Project thematic priority:	Priority 2: Information Society Technologies (IST)
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	IST506779/Cottbus/I1-D10/D/PU/b1
Responsible editors:	Sergey Lukichev
Reviewers:	Dragan Gasevic
Contributing participants:	Cottbus
Contributing workpackages:	I1
Contractual date of deliverable:	March 23, 2007
Actual submission date:	March 30, 2007

Abstract

In this report we describe a number of improvements to the rule modeling tool Strelka, developed in the Working Group I1. The main focus is on improvements of actions/events metamodel and using UML rules for modeling Web Services.

Keyword List

Rules, URML, Strelka, R2ML, Semantic Web, Web Services

Project co-funded by the European Commission and the Swiss Federal Office for Education and Science within the Sixth Framework Programme.

Tool Improvements and Extensions 1

Sergey Lukichev[1], **Adrian Giurca**[1], **Gerd Wagner**[1], **Marko Ribaric**[3]

[1]Institute of Informatics, Brandenburg Technical University at Cottbus, Germany
{lukichev, giurca, wagnerg}@tu-cottbus.de
[3]School of Electrical Engineering, University of Belgrade, Serbia
marko.ribaric@gmail.com

March 30, 2007

Abstract
In this report we describe a number of improvements to the rule modeling tool Strelka, developed in the Working Group I1. The main focus is on improvements of actions/events metamodel and using UML rules for modeling Web Services.

Keyword List
Rules, URML, Strelka, R2ML, Semantic Web, Web Services

Contents

Chapter 1

Modeling Web Services with Strelka

In this chapter, we present a UML- and rule-based approach to modeling Web services. The core of the solution is the UML-based Rule Model Language (URML) that allows for developing business vocabularies and rules independent of an implementation technology. This helps developers to focus on solving problems under study rather than on low-level platform-specific details. Here we demonstrate how several Web service message exchange patterns can be modeled by URML. To support the use of the proposed solution we: improve Strelka to support events and actions, employ the REWERSE I1 Rule Markup Language (R2ML) for encoding rules, and provide transformations between R2ML and WSDL, and thus round-trip engineering of Web services.

1.1 Introduction

Developing interoperable and loosely coupled components is one of the most important goals for Web-based software industry with the main goal to bridge the gaps between many different stakeholders. Web services seem as a promising attempt toward achieving that challenge, since they are based on a set of XML-defined standards for describing (WSDL [WSD]), publishing (UDDI [UDD04]), and messaging (SOAP [soa]). Although Web services offer many benefits, here we name two important factors that constraint their development and use:

1. There is no high-level approach to modeling systems under study, which should be supported by Web services. Instead, developers are manly focus on platform specific and low-level implementation details (e.g., elements of WSDL). The consequence is that one can hardly focus on modeling a system under study (e.g., a business process), and thus it decreases the productivity. On the other hand, developers also have to implement many things manually, which may lead to potential execution errors, especially when trying to extend the present Web services with new functionalities.

2. There is no/there are limited automatic mechanisms for updating Web services based on the business process changes. This is due to the fact that business systems are highly-dynamic and may change quite often. However, developers need ways that will allow them

for updating Web services based on such changes.

In order to address the first of the above constraints, we propose using an approach based on Model Driving Engineering (MDE). The main promise of MDE is to increase productivity of developers by switching their focus form platform specific implementation details to high-level (i.e. platform independent) modeling languages. Given such modeling languages, one can build models that can be later transformed into different implementation platforms [Sch06]. We should mention that there have been several solutions that are based on UML profiles and contain either elements of WSDL [BHLJ04], [VdCM05] or OWL-S [TG05], [GJH05]. Although, they are based on UML and use MDE principles, they are still very low-level oriented, as they again focus on implementation details covered either by WSDL or OWL-S.

To address the second issue, we propose using rule-based approaches [MdSM05]. When business processes are modeled by rules, such rules can be interpreted in run-time, and thus they can reflect the changes in the business process [Ros03]. However, currently there is no generally adopted Web rule language with the capacity to express either rules [rif] on which a Web service has been built or what events trigger the activations of a specific Web service. This further emphasizes the previous constraint by preventing developers to automatically develop Web services and their descriptions.

The solution that we propose is to use the UML-based Rule Modeling Language (URML) ([WGL06b], [LW06b]) that supports modeling domain vocabularies (i.e., ontologies) and integrity, derivation, production, reaction, and transformation rules. Considering the nature of Web services, we specifically propose using reaction rules (also known as Event-Condition-Action, ECA, rules) for describing business rules that can be later used for automatic generation of Web service descriptions. In the next section we give a brief overview of the URML language with the emphasis on reaction rules.

In Section 1.3, we describe how URML rules can be mapped to Web services, and thus using URML as a language for modeling Web services. In Section 1.4, we illustrate how Web service message exchange patterns are expressed by means of URML reaction rules. In Section 1.5, we define how we use the REWERSE Rule Markup Language (R2ML) for serializing URML rules. Before we conclude the paper, we describe process of transforming between URML and WSDL that is a part of our tool for URML (Strelka) [LW06a].

1.2 UML-based Rules Modeling

In order to provide UML-based rules modeling, the REWERSE Working Group I1 has developed a UML-based Rule Modeling Language (URML). The language has been designed having the following in mind:

- Rules are built on assertions, and assertions are built on concepts as expressed by terms [Ros03];
- MOF/UML can be used for defining a vocabulary of terms;
- UML metamodel is extended with additional rule concepts and visual notation for rules has been defined.

In this report we are concerned with reaction rules. Such rule formalizes event-condition-action behavioral model, where the action is executed on event with a condition satisfied. In this section URML visual notation is explained by means of a reaction rule example:

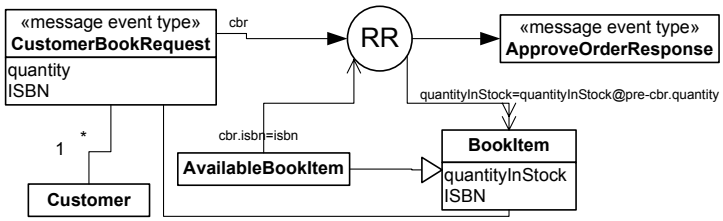

Figure 1.1: An example of a reaction rule expressed in URML

> On customer book item request, if the item is available, then approve order and decrease amount of items in stock.

In this rule *on customer book item request* is a triggering rule event, *if the item is available* is a rule condition, *approve order* is a rule action and *decrease amount of items in stock* is a postcondition, which can be expressed as a logical constraint, requiring less amount of items in stock, than before the rule execution. The rule diagram is depicted on Figure 1.1.

In order to model this rule in URML, first the rule vocabulary must be defined by means of UML classes. The vocabulary of this rule includes the following classes: *Customer*, *BookItem*, *AvailableBookItem*, *CustomerBookRequest* and *ApproveOrderResponse*. The event and the action are also a part of the vocabulary and are classes *CustomerBookRequest* and *ApproveOrderResponse* with the stereotype *<<message event type>>* (see Figure 1.1).

Rules in URML are depicted using circles with identifiers. Conditions are depicted as arrows from a conditioned model element to a rule circle (i.e. $\rightarrow$). A conditioned model element can be one of UML classifiers: *class*, *association*, *association end*. A negated condition, is depicted using a crossed arrow. A *message event type* model element is connected to the rule circle with the help of a *triggering event arrow* with a solid head. A rule event is an arrow from an event class to a rule circle. A *postcondition arrow* is depicted as an outgoing arrow with a double head (in order to denote a state change) from the rule circle to the postcondition classifier (class, association, or association end).

A *condition arrow* can be annotated with a *filter expression*, which is a boolean expression and is used to filter out instances of the condition classifier. For instance, an annotation *cbr.isbn=isbn* is a filter expression, which filters out only those available items, which correspond to the ISBN, specified in the request.

For more information on URML visual notation, examples, and the modeling tool Strelka we refer to [WGL06b] and [LW06a].

A *postcondition* arrow supports annotation with filter expressions, which specify the state change of the system (for example, the expression $quantityInStock = quantityInStock@pre - cbr.quantity$ in Figure 1.1).

1.3 Web Services Artifacts in the URML

Reaction rules can be considered as a Web Service interaction description. In this section, we define general web services artifacts as an extension of the URML metamodel for reaction rules. Figures 1.2, 1.3 and 1.4 depict an excerpt from the URML reaction rule metamodel, which is extended for the modeling of web service interfaces and operations.

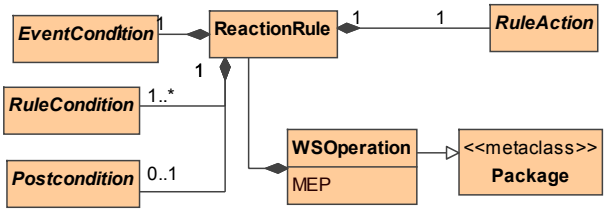

Figure 1.2: An excerpt of the URML metamodel for reaction rules

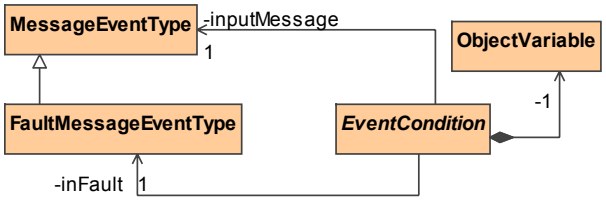

Figure 1.3: An excerpt of the URML metamodel for reaction rule events

An event condition has a message event type as a web service operation input message or fault message event type as an in-fault (See Figure 1.3). In the URML visual notation, an instance of the *MessageEventType* class is a user-defined class with a <<*message event type*>> stereotype, specifying a message.

An instance of the *EventCondition* class is depicted as an incoming arrow from the message type or fault message type class to the rule circle. An event condition refers to an object variable, which represents an instance of the message event class and corresponds to the annotation of the event condition arrow with the variable name (e.g., *cbr* in Fig. 1.1).

A rule action has a message event type as an output message or fault message event type as an out-fault. In the URML visual notation, the *RuleAction* class is depicted as an outgoing arrow from the rule circle to the message type or fault type class. A rule action refers to an object variable, which represents an instance of the *MessageEventType* class and corresponds to the annotation of the action arrow with the variable name. An instance of the class *FaultMessageEventType* is a class with the <<*fault message event type*>> stereotype.

A web service is represented as a UML package, which contains subpackages with types, interfaces, and bindings (see Figure 1.5). A web service operation may be modeled by using more than one rule (see Section 1.4.1.1). Rules, which define an operation, are grouped in one package by the name of the operation. Several operations may define a web service interface, and thus each operation package is a sub package of an interface package. The *Type* package contains classes, which define types of the web service messages.

1.4 Modeling WSDL MEPs with URML

In this section, we consider modeling of the *In-Out* and *Robust In-Only* WSDL Message Exchange Patterns (Section 2.4.1.1 of [WSD]) with URML. These patterns reflect widely-used

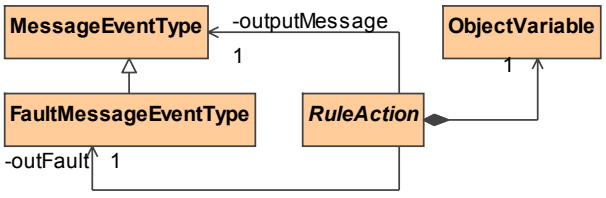

Figure 1.4: An excerpt of the URML metamodel for reaction rule actions

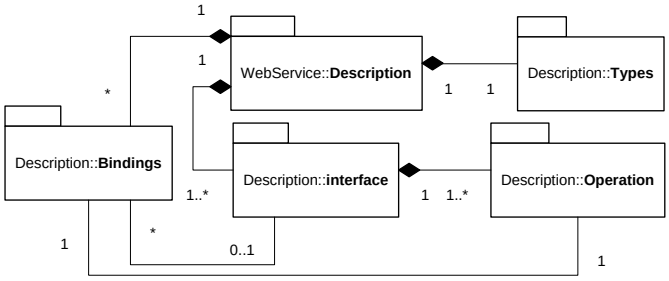

Figure 1.5: URML packages for a Web Service

request-response interaction of a user with a web service. URML models, presented in this section, are based on the metamodel, defined in Section 1.3. For each pattern, we give a web service operation example and corresponding URML model.

1.4.1 In-Out

This pattern consists of exactly two messages and uses the fault replaces message. Here we consider two examples of In-Out pattern: with an out-fault and with an in-fault.

1.4.1.1 In-Out pattern with out-fault

Let us consider a web service operation *CheckAvailability*, which has an input message *CheckAvailability* and an output message *CheckAvailabilityResponse*. The operation is performed successfully if the specified parametar *checkinDate* in the input message is before the *checkoutDate* and the room *isAvailable*. We follow the MDA modeling approach where the system is considered on the sequence of three levels: CIM (Computation-Independent Model), PIM (Platform-Independent Model) and PSM (Platform-Specific Model). On each level some additional information is added [MM03].

A web service Computation Independent Model (CIM) describes the domain and requirements of the service. The CIM consist of a model from the informational viewpoint, which captures information about the web service. The CIM corresponds to the conceptualization perspective's requirements model. The CIM level model of this operation, modeled with URML is depicted in Figure 1.6.

A Platform Independent Model (PIM) of a system describes the data and the behavior of the web service independent of any platform. A PIM might consist of a model from the

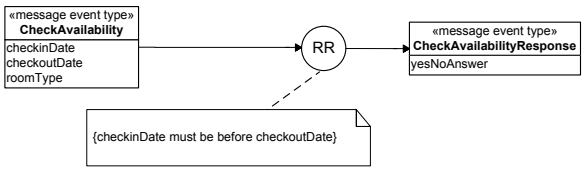

Figure 1.6: CIM level In-Out pattern in URML

informational viewpoint, which captures information about the data of a system, and a model from the computational viewpoint, which captures information about the processing of a system, independent of any platform. The PIM corresponds to the specification perspective's analysis model. The PIM level of this operation is depicted in Figure 1.7. Types of attributes are added and constraints are turned into URML conditions and filter expressions. At this level, the operation is modeled by means of two reaction rules, which are triggered by the same input message *CheckAvailability*.

```
ON CheckAvailability[input](checkinDate, checkoutDate)
IF checkinDate < checkoutDate AND isAvailable(Room)
THEN DO CheckAvailabilityResponse[output]("YES")

ON CheckAvailability[input](checkinDate, checkoutDate)
IF NOT checkinDate < checkoutDate THEN
DO InvalidDataError[outfault]("Check-in date
                    is more than check-out date")
```

The first rule with an identifier *CheckAvReqService.CheckAvailability.R1* captures the web service behavior if the data provided is correct and there is an available room. This rule is triggered by the triggering event *CheckAvailability* and it has a condition, which specifies the validity of the input date. The condition is represented as an arrow from the class *CheckAvailability* to the rule circle and is annotated with the filter expression $checkinDate < checkoutDate$. In addition to the condition, which checks validity of the input data, there is a second condition, requiring availability of the room and visualized as an arrow from the class *Room* to the *R1* rule circle.

The second rule with an identifier *CheckAvReqService.CheckAvailability.R2* captures the web service behavior if the data provided is not correct. An arrow with a bold arrow-head, connecting the class *CheckAvailability* and the rule circle, denotes a triggering event for the rule and specifies an input message of the operation. An input message is represented by the class *CheckAvailability* with the stereotype $<<message\ event\ type>>$. A crossed arrow from the class *CheckAvailability* to the rule circle, annotated with the filter expression $checkinDate < checkoutDate$, denotes a negated condition, which means that if the check-in date is more or equal to the check-out date, then the fault message *InvalidDataError* should be fired.

The condition part of the reaction rule set can be used in the generation of the executable code, while from WSDL only skeletons can be generated. Hence, URML models allows specifying a richer description of a web service than the WSDL.

These two rules define one operation of a web service interface. In order to denote its belongings to one operation, they are grouped in a package, named as the operation "Check-Availability". The operation pattern, style and some other parameters are specified in the annotation to a rule. Since both rules specify one operation, we annotate only one rule.

A PSM of a system describes the web service as it uses one or more specific platforms (i.e. WSDL). A PSM might consist of a model from the informational viewpoint, which captures

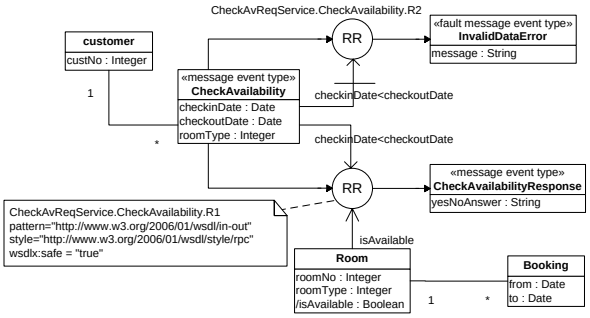

Figure 1.7: PIM level In-Out pattern in URML

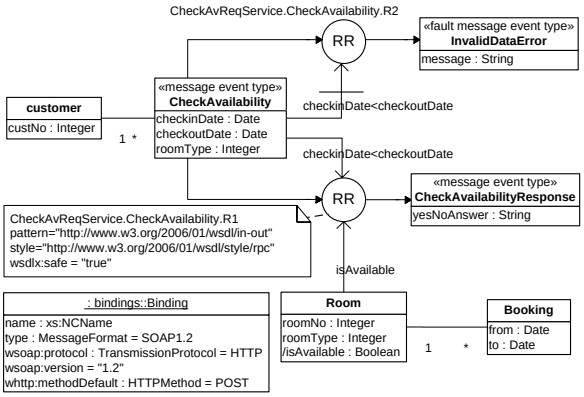

Figure 1.8: PSM level In-Out pattern in URML

information about the data of a web service, and a model from the computational viewpoint, which captures information about the processing of the web service, based on a specific platform. As a PSM targets a specific platform, it uses the features of the specific platform specified by a platform model. The PSM level model of this operation is depicted in Figure 1.8. On this level, the binding information is added and this model can be used to generate the WSDL XML documents.

Below is a part of the WSDL XML for this operation:

```
<wsdl:interface name="CheckAvReqService">
 <wsdl:operation name="CheckAvailability"
     pattern="http://www.w3.org/2006/01/wsdl/in-out"
  <wsdl:input element="tns:CheckAvailability"/>
  <wsdl:output element="tns:CheckAvailabilityResponse"/>
  <wsdl:outfault ref="tns:InvalidDataError"/>
 </wsdl:operation>
</wsdl:interface>
```

1.4.1.2 In-Out with in-fault

Let us consider a situation, when a service has an internal problem. The service forwards its incoming request to a mirror as an out-fault message. That mirror service has an operation

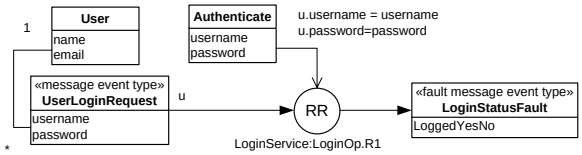

Figure 1.9: Robust In-Only pattern in URML

CheckAvailability with an in-fault *ServiceNotAvailable*, output message *CheckAvailabilityResponse*, and out-fault *InvalidDataError*. Similar to the In-Out pattern from Section 1.4.1.1, such an operation is modeled by means of two reaction rules, but these rules are triggered by the same in-fault message *ServiceNotAvailable*.

```
ON ServiceNotAvailable[infault](checkinDate, checkoutDate)
IF checkinDate < checkoutDate  AND isAvailable(Room)
THEN DO CheckAvailabilityResponse[output]("YES")

ON ServiceNotAvailable[infault](checkinDate, checkoutDate)
IF NOT checkinDate < checkoutDate THEN
DO InvalidDataError[outfault]("Check-in date is more
                than check-out date")
```

The corresponding URML diagram is similar to the one on Figure 1.7, but instead of the input message event type *CheckAvailability*, there is a fault message expression type (i.e., *<<fault message event type>>*) *ServiceNotAvailable*.

1.4.2 Robust In-Only

This pattern consists of exactly one input message and uses message triggers fault. Let us consider a login operation, which has an input message *LoginRequest* with a username and password and an out-fault *LoginFailed*, which may be sent back to the user when login fails. Below is a part of the WSDL XML document for this operation:

```
<wsdl:interface name="LoginService">
 <wsdl:operation name="LoginOp"
  pattern="http://www.w3.org/2006/01/wsdl/robust-in-only"
  <wsdl:input element="tns:LoginRequest"/>
  <wsdl:outfault ref="tns:LoginFailed"/>
 </wsdl:operation>
</wsdl:interface>
```

This operation is modeled by means of one reaction rule:

```
ON LoginRequest[input](username, password)
IF NOT Authenticate(username, password)
DO LoginFailed[outfault]("Wrong password")
```

The corresponding URML diagram is depicted on Figure 1.9.

The input of the operation is a *LoginRequest* class with a username and password. Event arrow from the message event type class to the rule circle is annotated with the rule variable u. The rule has a condition, which checks whether the username and password, provided in the event, correspond to the internal username and password. If username and password are not correct, then the outfault *LoginFailed* is sent back to the user. We should note that in this diagram, we show how parameters of the variable u are bound with attributes of the class *Authenticate*.

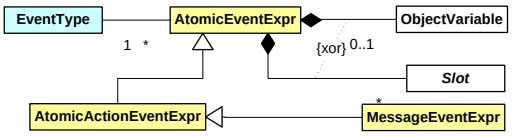

Figure 1.10: An excerpt from the R2ML event model

1.5 R2ML as an Interchange Format

The REWERSE I1 Rule Markup Language (R2ML) [WGL05], [WGL06a], [WGL06b] is an XML rule interchange format with the main purpose to perform rule loss-free interchange. Its abstract syntax is defined by a MOF-based metamodel, while its concrete syntax is defined by an XML schema [WGL06b]. R2ML v0.5 supports various rule types including reaction rules and it is a serialization format for the URML. An R2ML 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 its conditions are satisfied. Post-conditions may be optionally required to be satisfied after the action execution. The execution effect of reaction rules may depend on the rules order (note that the order is defined by the rule execution mechanism or by the rules representation). The R2ML Events Metamodel specifies the core concepts required for dynamic behavior of rules and provides the infrastructure for more detailed definition of this behavior. One of the event types supported by R2ML is the *message event expression* (see Figure 1.10) which is suitable for modeling Web service messages. A *message event expression* is an atomic event described by: *i)* a reference to an event type (from the domain vocabulary); and *ii)* a number of slots (i.e. property-value pairs), which describe event parameters or a global object variable, which encode all parameters. For instance, an instance of the *CheckAvailability* message event type of the reaction rule `R1` depicted in the Figure 1.7 is represented in R2ML as:

```
<r2ml:MessageEventExpr r2ml:eventType="CheckAvailability">
 <r2ml:DataSlot r2ml:attributeID="checkinDate">
  <r2ml:value>
   <r2ml:TypedLiteral r2ml:lexicalValue="2006-12-24"
                      r2ml:datatypeID="xs:date"/>
  </r2ml:value>
 </r2ml:DataSlot>
 <r2ml:DataSlot r2ml:attributeID="checkoutDate">
  <r2ml:value>
   <r2ml:TypedLiteral r2ml:lexicalValue="2007-01-03"
                      r2ml:datatypeID="xs:date"/>
  </r2ml:value>
 </r2ml:DataSlot>
 <r2ml:DataSlot r2ml:attributeID="roomType">
  <r2ml:value>
   <r2ml:TypedLiteral r2ml:lexicalValue="single"
                      r2ml:datatypeID="xs:string"/>
  </r2ml:value>
 </r2ml:DataSlot>
</r2ml:MessageEventExpr>
```

The URML `CheckAvailability` class (Figure 1.8) corresponds to the `r2ml:eventType` attribute value. According to its stereotype <<*message event type*>>, an R2ML *message event expression* is used (i.e., `r2ml:MessageEventExpr`).

The vocabulary class `CheckAvailability` directly corresponds to the WSDL XML element declaration from the `types` section, i.e.

```
<types>
 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.service.org/resSvc"
```

Table 1.1: An excerpt of the mappings between the R2ML and WSDL elements

R2ML construct	**WSDL construct**
EventType	type
triggeringEvent	input, infault
producedAction	output, outfault
ReactionRule	operation and bindings
RuleSet	service

```
   xmlns="http://www.service.org/resSvc">
  <xs:element name="checkAvailability">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="checkInDate" type="xs:date" />
     <xs:element name="checkOutDate" type="xs:date" />
     <xs:element name="roomType" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
 </xs:element>
 <!- ... -->
 </xs:schema>
</types>
```

This type corresponds to the type of the `<input>` of the web service operation, i.e.

```
<operation name="checkAvailability"
 pattern="http://www.w3.org/2006/01/wsdl/in-out">
 <input messageLabel="In" element="ghns:checkAvailability" />
 <!-- ... -->
</operation>
```

The filter expressions from the rule `R1` are represented as R2ML conditions. For instance, the boolean filter $checkinDate < checkoutDate$ is represented as a datatype predicate atom (used to encode the relational predicate $<$).

```
<r2ml:DatatypePredicateAtom
      r2ml:datatypePredicateID="swrlb:greaterThan">
 <r2ml:dataArguments>
  <r2ml:AttributeFunctionTerm
        r2ml:attributeID="checkinDate">
   <r2ml:contextArgument>
    <r2ml:ObjectVariable r2ml:name="checkAv"
           r2ml:classID="CheckAvailability"/>
   </r2ml:contextArgument>
  </r2ml:AttributeFunctionTerm>
  <r2ml:AttributeFunctionTerm
        r2ml:attributeID="checkoutDate">
   <r2ml:contextArgument>
    <r2ml:ObjectVariable r2ml:name="checkAv"
           r2ml:classID="CheckAvailability"/>
   </r2ml:contextArgument>
  </r2ml:AttributeFunctionTerm>
 </r2ml:dataArguments>
</r2ml:DatatypePredicateAtom>
```

The action part of the reaction rule `R1` corresponds to another message event expression, similar to the incoming event. Faults (*in* or *out*) are also message event expressions. The Table 1.1 describes an excerpt from the R2ML to WSDL mapping.

Each event type from the reaction rule is mapped onto a corresponding XML Schema element in the WSDL types section as we already described in the previous example. An R2ML `MessageEventExpr` is mapped into WSDL according to the following rules:

- The `triggeringEvent` of the rule is mapped to the WSDL `input` or WSDL `infault` according to the event type from the domain vocabulary (see also Section 1.4.1.2).

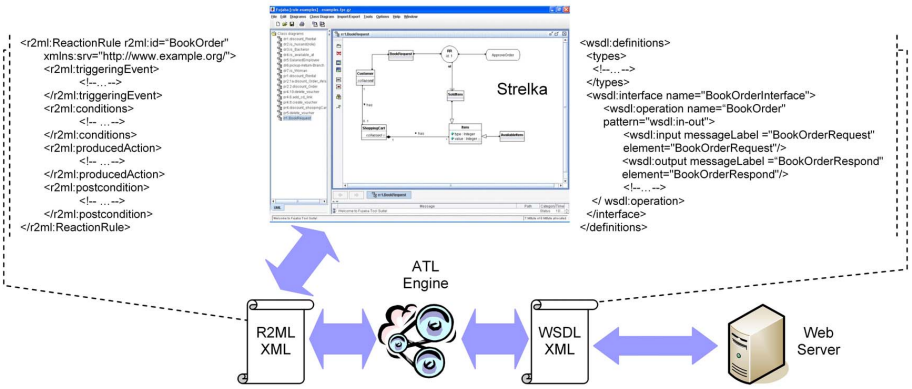

Figure 1.11: General procedure of getting a WSDL document from a URML tool (i.e., Strelka)

- The `producedAction` of the reaction rule is mapped into WSDL `output` or WSDL `outfault` according to the event type from the domain vocabulary and the condition part of the rule (see URML diagram from Figure 1.8).

The annotated R2ML `ReactionRule` element captures the entire information to describe the WSDL `operation` element including `name`, `pattern` and `style` attributes. The WSDL service element is captured by an annotated R2ML `RuleSet` element.

The next section describes the entire process of generating web services WSDL descriptions from URML models. This includes an R2ML model as a serialization format for URML models. From an R2ML model we obtain a WSDL model, compliant with the WSDL metamodel.

1.6 Generating Web service descriptions from URML models

In this section, we explain the transformation steps that need to be undertaken, to transform R2ML rules used for encoding URML models onto WSDL descriptions. Here we should point out that we are using a well-known UML tool called Fujaba for which we are developing a plug-in named Strelka that is used to support URML notation. Among other things, Strelka supports serialization of URML models into the R2ML XML concrete syntax. In Figure 1.11, we give a high level description of the applied transformation procedure.

We decided to implement this transformation in the MOF technical space by using model transformation languages such as ATL [atl]. This is an easier way to provide transformations of MOF-based models and to maintain a transformation rather than using XSLT [JG03]. The transformation process itself consists of four steps.

Step 1. This step consists of injecting R2ML rules encoded in the R2ML XML format into the MOF technical space, i.e., to the representation compliant to the R2ML metamodel. Such a process is shown in detail in [MGG+06]. We use the XML injector that transforms an R2ML XML document into a model conforming to the MOF-based XML metamodel that defines XML elements such as *XML Node*, *Element*, and *Attribute*. This XML injector is distributed as a tool along with the ATL engine. The result of this injection is an XML model that can

be represented in the XML XMI format, which can be later used as the input for the ATL transformation.

Step 2. In this step, we transform the XML model from Step 1 into a (R2ML) model compliant with the R2ML metamodel [WGL06b]. This transformation is done by using the ATL transformation named XML2R2ML.atl. Originally the transformation covered just production and derivation rules, and now it is expanded with reaction rules.

Step 3. The R2ML model (obtained in the previous step) is transformed onto a (WSDL) model compliant to the WSDL metamodel, which developed at the University of Cottbus (a similar one to those presented in [BHLJ04][VdCM05]). This step represents the transformation of the R2ML abstract syntax into the WSDL abstract syntax. This transformation step is fully based on the conceptual mappings between the elements of the R2ML and WSDL metamodel.

In Figure 1.12, we show the conceptual model of this transformation. The actual transformation between the R2ML model and elements of the WSDL metamodel are defined as a sequence of rules in the ATL language (R2ML2WSDL.atl). These rules use additional helpers in defining mappings. Each rule in the ATL has one input element (i.e., an instance of a metaclass from a MOF based metamodel) and one or more output elements. ATL, in fact, instantiate the WSDL metamodel (the M2 level), i.e. it creates WSDL models (the M1 level). It is important to point out that M1 models (both source and target ones) must be conformant to their M2 metamodels. This principle is well-known as metamodel-driven model transformations [Bez01].

Step 4. The step is the transformation from the WSDL model to the WSDL XML format. This transformation is done by using the ATL transformation named WSDL2WSDLxML.atl. The output file is a regular WSDL XML-based document that allows service authors to provide crucial information about the service so that others can use it.

We should also point out that all transformations mentioned are implemented in both directions, so that one can transform from R2ML XML to WSDL XML and way back from WSDL XML to R2ML XML, and thus we can reversely engineer existing Web services into reaction rules of the R2ML and URML.

1.7 Conclusion

In this report, we have proposed the use of a UML-based rule modeling language for modeling Web services. Building the URML closely related to the R2ML Web rule language, we have provided a representation completely independent of Web service and MEP models. This should enable for transformation of the business logics that Web services are built on onto other rule languages such as F-Logic or Jess. This means that rule-based business models can be shared and reused regardless of the target technology on which they will be implemented. Moreover, developers do not have to focus on low level Web service implementation details, but rather on the description of business process by using reaction rules. Once they develop their business models in the URML, they can generate Web services by using transformations we proposed between R2ML and WSDL. Since URML is based on the UML, it also allows for using UML classes for modeling business vocabularies, and thus generating XML Schema types based on vocabulary concepts as well as interoperating with business vocabularies or ontologies.

This approach can be used not only when modeling business processes from scratch and then generating new Web services, but it can also be applied to re-engineering the existing Web services and update them with new business rules. This is possible, since our approach contains two way transformations between WSDL and URML reaction rules. The novelty of

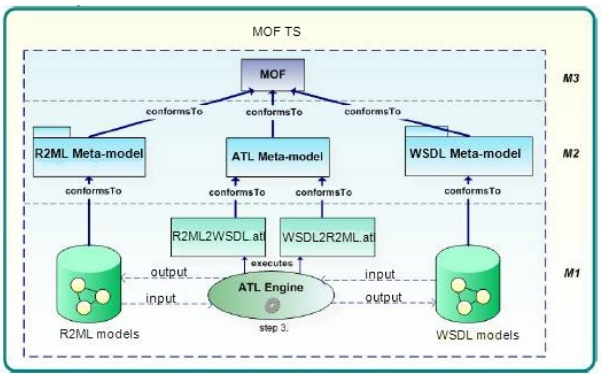

Figure 1.12: The model transformations between the R2ML metamodel and the WSDL metamodel

our approach over other ones is that we propose the use of business rules rather than workflow models and languages such as BPMN or BPEL4WS. Developing workflow models can be hard, since one has to grasp and predicts all the aspects that might appear in the business process execution. By using rules, workflows can automatically be generated based on the interpretation of business, which allows for easier updating business policies and consequently Web services.

In the future, we plan to finalize our implementation of transformation chain between R2ML and WSDL with the main emphasis on the full support of generation of XML schema types from UML class models. This means not just types captured by message types, but also all other types that are used in Web service message types. This is, of course, a natural point where we consider a further extension of XML types generated to be annotated with vocabulary concepts. That is, we will extend our approach on semantic Web services (e.g., WSDL-S or WSMO), where we will also additionally investigate the use of pre- and post-conditions of reaction rules.

Chapter 2

Other Improvements

Here is a list of minor improvements that have been made to Strelka since it was described in Deliverables [WL06a] and [WL06b]:

1. URML rules are built on top of UML vocabularies. Rules serialization into R2ML XML by Strelka has been described in [WL06b]. Current version of Strelka serializes UML vocabularies into R2ML vocabularies. Many rule languages (for instance, JBoss Rules, F-Logic) need vocabularies for their rule, therefore XML code generation for vocabularies is an important rule modeling issue for some rule platforms.

2. Improved support of OCL syntax in URML filter expressions (for instance, support of operation calls).

3. A number of bug fixes has been made in R2ML XML code generation and in parsing of filter expressions.

Our preliminary plan for further work includes:

1. Strelka as an Eclipse Plugin and, possibly, as an independent application, based on Eclipse Graphical Modeling Framework (GMF). Shift to the Eclipse platform will spread the use of URML among large community of Eclipse users and will make the tool more user-friendly, since Eclipse has more advanced GUI and a lot of additional features (for instance, more reliable OCL parser, then the one currently used in Strelka).

2. Further improvements of rule-based web services modeling.

3. Integration of Strelka with R2ML translators and the R2ML verbalizer.

Bibliography

[atl] ATLAS transformation language (ATL). http://www.sciences.univ-nantes.fr/lina/atl.

[Bez01] J Bezivin. From object composition to model transformation with the MDA. In *In Proc. of the 39th Int. Conf. and Exh. on Tech. of OO Lang. and Sys.*, pages 350–355, 2001.

[BHLJ04] J. Bezivin, S. Hammoudi, D. Lopes, and F. Jouault. Applying MDA approach for Web Service Platform. In *Proc. of the 8th IEEE Int. Conf. on Enterprise Distributed Object Computing Conf.*, pages 58–70, 2004.

[GJH05] R. Gronmo, M.C. Jaeger, and H. Hoff. Transformations between UML and OWL-S. In *Proc. of the 1st European Conf. on Model Driven Architecture - Foundations and Applications (ECMDA-FA), Nuremberg, Germany*, pages 269–283, 2005.

[JG03] J. Jovanovic and D. Gasevic. XML/XSLT-based knowledge sharing. *Expert Systems with Applications*, 29(3):535–553, 2003.

[LW06a] S. Lukichev and G. Wagner. Uml-based rule modeling with fujaba. In Holger Giese and Bernhard Westfechtel, editors, *Proceedings of the 4th International Fujaba Days 2006, University of Bayreuth, Germany*, pages 31–35, 2006.

[LW06b] S. Lukichev and G. Wagner. Visual rules modeling. In Irina Virbitskaite and Andrei Voronkov, editors, *Proceedings of the 6th International Conference Perspectives of Systems Informatics*, volume 4378 of *Lecture Notes in Computer Science*, pages 467–673. Springer, 2006.

[MdSM05] Colleen McClintock and Christian de Sainte Marie. ILOG's position on rule languages for interoperability. In *Rule Languages for Interoperability*. W3C, 2005.

[MGG+06] M. Milanovic, D. Gasevic, A. Giurca, G. Wagner, and V. Devedzic. On interchanging between owl/swrl and uml/ocl. In *Proceedings of the OCLApps Workshop*, pages 81–95, Genova, Italy, 1-6 October 2006 2006.

[MM03] J. Miller and J. Mukerji. MDA guide, Ver. 1.0. Technical report, OMG, 2003.

[rif] RIF Use Cases and Requirements. W3C Working Draft 10 July 2006. http://www.w3.org/TR/rif-ucr.

[Ros03] R. G. Ross. *Principles of the Business Rule Approach*. Addison-Wesley, 1nd edition edition, 2003.

[Sch06] D.C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, 2006.

[soa] SOAP Ver. 1.2 Part 1: Messaging Framework. W3C Recommendation 24 June 2003. http://www.w3.org/TR/soap12-part1/.

[TG05] J. T. Timm and G. C. Gannod. A Model-Driven Approach for Specifying Semantic Web Services. In *Proc. of the IEEE international Conf. on Web Services*, pages 313–320, 2005.

[UDD04] UDDI302. UDDI Ver. 3.0.2. Technical report, OASIS, 2004. http://uddi.org/pubs/uddi_v3.htm.

[VdCM05] Juan M. Vara, Valeria de Castro, and Esperanza Marcos. WSDL Automatic Generation from UML Models in a MDA Framework. *Int. Journal of Web Services Practices*, 1(1-2):1–12, 2005.

[WGL05] G. Wagner, A. Giurca, and S. Lukichev. R2ml: A general approach for marking up rules. In *Dagstuhl Seminar Proc. 05371*, 2005.

[WGL06a] G. Wagner, A. Giurca, and S. Lukichev. A usable interchange format for rich syntax rules. integrating ocl, ruleml and swrl. In *Proceedings of the Reasoning on the Web Workshop*, Edinburgh, Scotland, May 2006.

[WGL06b] Gerd Wagner, Adrian Giurca, and Sergey Lukichev. Language improvements and extensions. Deliverable i1-d8, REWERSE IST 506779, March 2006.

[WL06a] G. Wagner and S. Lukichev. Strelka - a visual rule modeling tool. Technical report, REWERSE IST 506779, March 2006.

[WL06b] G. Wagner and S. Lukichev. Xml code generation component for the i1 rule modeling tool. Technical report, REWERSE IST 506779, September 2006.

[WSD] WSDL20. *Web Services Description Language (WSDL) Ver. 2.0 Part 1: Core Language*. W3C Candidate Recommendation 27 March 2006. http://www.w3.org/TR/2006/CR-wsdl20-20060327.