A Rule-based Approach to Modeling of Semantically-enriched Web Services

Marko Ribarić[1], Dragan Gašević[2], Milan Milanović[3]

[1]Mihajlo Pupin Institute, Serbia
[2]School of Computing and Information Systems, Athabasca University, Canada
[3]FON-School of Business Administration, University of Belgrade, Serbia
marko.ribaric@institutepupin.com, dgasevic@acm.org, milan@milanovic.org

Abstract – **Web services are usually defined as autonomous, platform-independent computational elements that can be *described, published, discovered, orchestrated and programmed* using standard protocols for the purpose of building networks of collaborating applications distributed within and across organizational boundaries. Semantic Web services present the augmentation of Web service descriptions through Semantic Web annotations (e.g., references to ontologies), to facilitate higher automation of service discovery, composition, invocation, and monitoring on the Web. Today, there is a need for effective mechanisms for modeling Web services where as creation of these mechanisms for Semantic Web services (SWS) modeling is especially challenging, as SWS are relatively new technology. In this paper, we propose a modeling approach that enables one to model Semantic Web services from the perspective of the underlying business logic regulating how Web services are used regardless of the context where they are used. This is done by modeling Semantic Web services in terms of message-exchange patters, where each service is described by a (set of) rule(s) regulating how Web services' messages are exchanged. We show how our approach can be used with the recent W3C recommendation SAWSDL (Semantic Annotations for WSDL and XML Schema)**

I. INTRODUCTION

During the past few years, service oriented architecture (SOA) has become a dominant architecture style in industry. Combining smaller services into larger services is the core requirement in service-oriented architectures (SOAs). According to Papazoglou and Georgakopoulos [31] the process of *service composition* encompasses necessary roles and functionality for the consolidation of multiple services into a single composite service. The resulting composite services may be used by service aggregators as components in further compositions or may be utilized as applications by clients.

Web services proved to be the most mature framework toward achieving the SOA goal [4], since they are based on a set of XML based standards for description, publication, and invocation of services ([1], [2], [3]). However, researchers have found that the mere usage of basic Web service standards would not create scalable solutions [5]. A solution to search, integration, and meditation in large scale, open and heterogeneous environments was needed, and it was found in the area of Semantic Web. The use of semantics in Web services solves those problems by bringing several improvements, including [26]: better reuse (semantics improves finding of relevant services), better interoperability (semantics allows development of mappings of data that is being exchanged between the services) and easier composition of services.

Researchers proposed Semantic Web services that took approach of annotating service elements with terms from domain models, including industry standards, vocabularies, taxonomies, and ontologies [6]. In this paper, we will focus on the recent W3C recommendation SAWSDL (Semantic Annotations for WSDL and XML Schema) [7]. SAWSDL is built on existing Web standards using only extensibility elements, and thus taking evolutionary approach rather then revolutionary. By adopting this philosophy, SAWSDL offers an elegant mechanism that allows the externalization of the semantic domain models. This mechanism is fully agnostic to ontology representation languages; it allows the reuse of existing domain models; and also allows annotation using multiple ontologies [26].

In this paper we propose the use of a high-level modeling approach in the process of developing semantically annotated Web services. That is, our approach is based on the principles of Model Driven Engineering (MDE) [27]. By using MDE, we can first define a modeling language that is suited for modeling specific problem domains (in our case business logic that should be supported by Web services). Models created by such modeling languages can later be transformed (by using model transformation languages) into different implementation platforms [11].

Web services are usually jointly used to realize more complex functionality, typically a business process. A business process specifies the potential execution order of operations from a collection of Web services, the data shared between these Web services, which partners are involved and how they are involved in the business process, and other issues involving how multiple services and organizations participate [12]. So, there exists a need to support workflow-independent services modeling, and yet to consider some potential patterns or conditions under which a specific service can be used. In this paper, we propose focusing the design perspective from the question where (or in what context) to the question how a service is used. To do so, our proposal is to leverage message-exchange patterns (MEPs) as an underlying perspective integrated into a Web service modeling language.

Also, our solution involves the use of rules for addressing highly dynamic nature of business systems. By using rules, one can dynamically reflect business logic changes at run-time without the need to redesign the system. Since Web services are used for integration of business processes of

various stakeholders, it is important for them to reflect changes in business logic, or policies, as good as possible. This means if we can support the definitions of services based on rules, we can also reflect changes in service-oriented processes dynamically. This has already been recognized in the service community, where Charfi and Mezini [13] demonstrate how rules can be used to make more dynamic business processes based on service-oriented architectures. However, developers need development mechanisms that will allow them for building and later updating Web services based on such changes. Thus, we propose using rule-based approaches to modeling Web services [14]

This way of modeling Web services we thoroughly explained in [8]. In this paper, we further expand this process for the need of modeling Semantic Web services, specifically we concentrate on the use of SAWSDL. It is important to emphasize that this solution presented here, is made under the assumption that domain model and domain ontology are the same (see Sect. 3 for details), meaning that our SAWSDL annotations do reference just the classes we use to describe our domain model (i.e. our working Vocabulary).

The paper is structured as follows. Section 2 gives a brief introduction to technologies that we use in our approach, i.e. we give a brief definition of the languages on which our solution is based, including, UML-based Rule Language (URML), REWERSE Rule Markup Language (R2ML) [9], and SAWSDL (where SAWSDL subsection also summarizes the advantages we get by employing semantics to Web services). In Section 3, we describe the modeling of Semantic Web services from the perspective of MEPs (message exchange patterns). In Section 4, we give a process of transformation between rule-based models to the SAWSDL, as a part of the tooling support that we have implemented to support our approach [10]. And, finally in section 5 we conclude the paper.

II. BACKGROUND

A. R2ML

R2ML (REWERSE Rule Markup Language) is a rule language that addresses all the requests defined by the W3C working group for the standard rule interchange format [17]. The R2ML language is defined by MDE principles. The R2ML language can represent different types of rule constructs, that is, it can represent different types of rules [18], including, integrity rules, derivation rules, production rule, and reaction rules. Integrity rules in R2ML, also known as (integrity) constraints, consist of a constraint assertion, which is a sentence in a logical language such as first-order predicate logic or OCL. Derivation rules in R2ML are used to derive new knowledge (conclusion) if a condition holds. Production rules in R2ML produce actions if the conditions hold, while post-conditions must also hold after the execution of actions. A reaction rule is a statement of programming logic [19] 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 execution of the action(s), post-conditions may be made true. R2ML also allows one to define vocabularies by using the following constructs: basic content vocabulary, functional content vocabulary, and relational content vocabulary.

Here, we give short description of vocabulary constructs that we use in this paper. Vocabulary is a concept (class) that can have one or more VocabularyEntry concepts. VocabularyEntry is abstract concept (class) that is used for representing other concepts by its specialization. For example, one of the VocabularyEntry-s is an R2ML Class concept which represents the class element similar to the notion of the UML Class. An R2ML Class can have attributes (class Attribute), reference properties (class ReferenceProperty) and operations (class Operation).

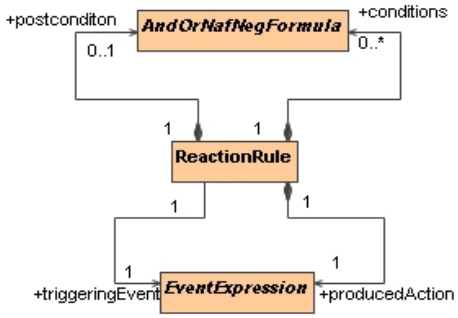

Fig. 1. The definition of reaction rules in the R2ML metamodel

Due to the importance for our Web service modeling approach, here we only describe the details of R2ML reaction rules. Reaction rules represent a flexible way for specifying control flows, as well as for integrating events/actions from a real life [19]. Reaction rules are represented in the R2ML metamodel as it is shown in Fig. 1: triggeringEvent is an R2ML EventExpression (the R2ML event metamodel defines basic concepts that are needed for dynamic rule behavior - each of those concepts is subclassed from the EventExpression class); conditions are represented as a collection of quantifier free logical formulas; producedAction is an R2ML EventExpression and represents a system state change; and (optional) postcondition must hold when the system state changes.

B. URML

UML-Based Rule Modeling Language (URML) is a graphical concrete syntax of R2ML. URML is developed as an extension of the UML metamodel to be used for rule modeling. In URML, modeling vocabularies is done by using UML class models. Rules are defined on top of such models. The URML reaction rule metamodel, which we use for modeling services, is shown in Fig. 2a. The figure shows components of a reaction rule: Condition, Postcondition, RuleAction and EventCondition. The figure also shows that reaction rules are contained inside the UML package which represents Web services operation. This means, that such packages have a stereotype <<operation>> in UML diagrams. An instance of the EventCondition class is represented on the URML diagram as incoming arrow (e.g., see Fig. 4), from a UML class that represents either an input message or an input fault message of the Web service

operations, to the circle that represents the reaction rule. The UML class that represents the input message (inputMessage in Fig. 2b) of the Web service operation is MessageEventType (a subclass of EventType) and it is represented by using the <<message event type>> stereotype on UML classes. The UML class that represents the input fault message (inFault in Fig. 2b) of the Web service operation is FaultMessageEventType in the URML metamodel. In URML diagrams, FaultMessageEventType is represented by the <<fault message event type>> stereotype on UML classes. EventCondition contains an object variable (ObjectVariable in Fig. 2c), which is a placeholder for an instance of the MessageEventType class.

An instance of the RuleAction class is represented as an outgoing arrow on the URML diagram, from the circle that represents the reaction rule to the class that represents either an output message or an output fault message of the Web service operation. The UML class that represents the output message (outputMessage in Fig. 2c) of the Web service operation is MessageEventType and it is represented with the <<message event type>> stereotype on UML classes. The UML class that represents the output fault message (outFault) of the Web service operation is FaultMessageEventType in the URML metamodel and it is represented with the <<fault message event type>> stereotype on UML classes. RuleAction also contains an object variable (ObjectVariable), which represents an instance of the MessageEventType class.

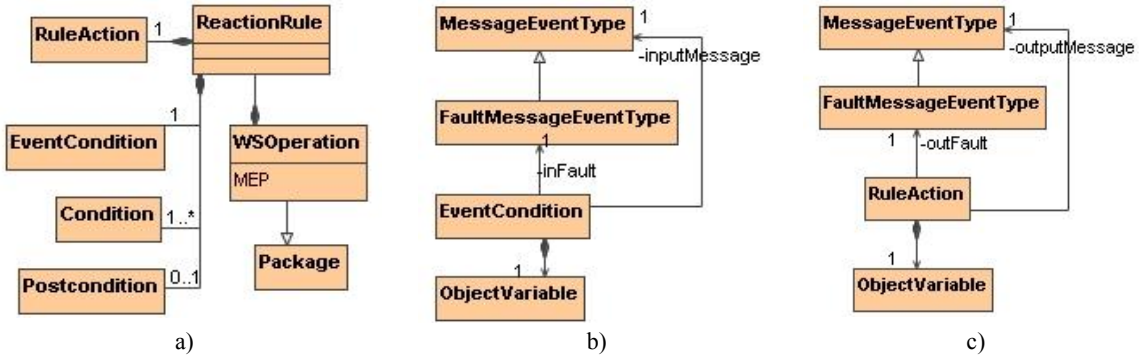

Fig. 2. a) Extension of the URML metamodel for reaction rules; b) Part of the URML metamodel for EventCondition; c) Extension of the URML metamodel for actions

C. SAWSDL

Web services are a key enabler for service-oriented architectures that focus on service reuse and interoperability [5]. Researchers found that for reaching this interoperability goal some form of semantics needed to be added to services. They offered many advantages of doing so: e.g. in [25] authors say that: 1) models that employ semantics promote reuse and interoperability among independently created and managed services, 2) ontology supported representations based on formal and explicit representation lead to more automation, and 3) explicit modeling of the entities and their relationships between them allows performing deep and insightful analysis. In [23], authors stress that issues of structural and semantic heterogeneity between messages exchanged by Web services are crucial to interoperability. In that paper, authors further classify structural and semantic message level heterogeneities as: (a) Domain level incompatibilities that arise when semantically similar attributes are modeled using different descriptions. (b) Entity definition incompatibilities that arise when semantically similar entities are modeled using different descriptions. (c) Abstraction level incompatibilities that arise when two semantically similar entities or attributes are represented at different levels of abstraction. Paper [24] points out that using ontologies not only brings user requirements and service advertisements to common conceptual space, but also helps to apply reasoning mechanism to find a better match.

Currently, Web services are described using WSDL descriptions [1]. WSDL document contains one root element called *Description*. *Description* component contains *Schema*, *Interface*, *Binding*, and *Service* components. *Schema* component's purpose is to define data types that are being used in messages (WSDL most frequently relies on the XML schema for this task). *Interface* component describes a sequence of messages the service sends and/or receives. This is achieved by grouping messages into operations. *Operation* component describes an operation a given interface supports. Operation presents service interaction that contains a set of messages that are being sent between service and other parties involved in interaction. The order and cardinality of messages participating in certain interaction is dictated by the message exchange pattern (MEP) that operation is using. Each operation can have input message, output message, or both of them, plus an optional fault message. *Interface* also contains *Fault* component: fault event can happen during the exchange of messages and can disrupt the normal flow of messages. *Binding* component describes binding of Interface component to the concrete message format and communicational protocol. (i.e. *Binding* component defines implementation details needed to access a service). *Service* component contains a collection of end points, where all end points implement one interface. Endpoint component describes where (on which address) a service can be found.

On the other hand, SAWSDL [7] is a W3C recommendation that defines mechanisms by which semantic annotations can be added to WSDL components.

As already mentioned, SAWSDL is an evolutionary approach built on existing Web standards (WSDL) using only extensibility elements. It defines extension attributes that we can apply to elements both in WSDL and in XML Schema to annotate WSDL interfaces, operations, and their input and output messages [5]. The SAWSDL extensions take two forms:

- model references (presented with the extension attribute *modelReference*) that can be applied to any WSDL or XML schema element in order to point to semantic concepts. SAWSDL is agnostic to the domain model and ontology representation language.
- schema mappings that specify data transformations between messages' XML data structure and the associated semantic model SAWSDL provides two attributes for attaching schema mappings: *liftingSchemaMapping* (lifting mappings transform XML data from a Web service message into a semantic model) and *loweringSchemaMapping* (lowering mappings transform data from a semantic model into an XML message). SAWSDL is agnostic to the mapping (and transformation) language.

As part of our approach described in [8], we presented a metamodel for WSDL 2.0. In order to support solution presented in this paper, we had to change our WSDL 2.0 metamodel, so that it reflects annotations SAWSDL introduces. In Fig. 3, we present a fragment from our SAWSDL metamodel that shows how we achieved this. We show here the SAWSDL metamodel in the KM3 format (KM3 is domain specific language for defining metamodels with syntax that is similar to the Java syntax [28]).

First, let us point out that we are using an XML Schema Definition (XSD) metamodel retrieved from Eclipse Model Development Tools (MDT) subproject [29]. The XSD metamodel is introduced into our solution through the package we named *Xs*. The XSD metamodel offers the *XsAnnotation* class as a means of inserting annotations to the XML elements (both complex and simple ones). So, what we did is that we first added a new package called SAWSDL. This package contains just three classes: *ModelReference*, *LiftingSchemaMapping*, and *LoweringSchemaMapping*, where each class extends *XSAnnotaion* class from the Xs package. These classes correspond to the annotations SAWSDL introduces. Then, we added two more abstract classes to the WSDL package: *MRAnnotation* and *AllAnnotations*. The *MRAnnotation* class has a reference to a *ModelReference* class from the SAWSDL package, and this class is supposed to be extended by all the WSDL classes that model reference can be applied to (i.e., *Interface*, *Outfault*, *Infault*, *Operation*, and *Fault* classes). The *AllAnnotations* class has a reference to a *XsAnnotations* class from the *Xs* package, and this class is supposed to be extended by all the WSDL classes to which schema mappings can be applied to (i.e. *Output* and *Input* classes).

III. MODELING APPROACH

As previously stated, the approach we take for Semantic Web services modeling is based on our approach to modeling "regular" Web services. In other words, because of the nature of our approach [8], and thanks to SAWSDL's non invasive mechanisms by which semantic annotations can be added to WSDL components, we could use the same approach here, of course with some changes applied, for the Semantic Web services modeling. In the next two, sections we discuss its use and necessary modifications.

```
package WSDL {

    class Interface extends MRAnnotation {
        reference fault[*] container : Fault;
        reference "operation"[*] container : Operation;
        attribute name : String;
    }

    class Operation extends MRAnnotation {
        reference infault[*] container : Infault;
        reference outfault[*] container : Outfault;
        reference output[*] container : Output;
        reference input[*] container : Input;
        attribute name : String;
        attribute pattern : MEP;
        attribute style[0-1] : OperationStyle;
        attribute wsdlx_safe[0-1] : Boolean;
    }

    abstract class MRAnnotation {
        reference annotation[0-1] : ModelReference;
    }

    abstract class ALLAnnotations {
        reference annotations[*] ordered : XsAnnotation;
    }

--  ........

    package Xs {
        class XsAnnotation extends XsRedefineContent {
        }

        class XsComplexTypeDefinition extends XsTypeDefinition {
            reference baseTypeDefinition : XsTypeDefinition;
            reference content[0-1] container : XsComplexTypeContent;
            reference contentType[0-1] : XsComplexTypeContent;
--          .....
        }

        class XsSimpleTypeDefinition extends XsTypeDefinition  {
            reference baseTypeDefinition : XsSimpleTypeDefinition;
            reference primitiveTypeDefinition[0-1] : XsSimpleTypeDefinition;
            reference itemTypeDefinition[0-1] : XsSimpleTypeDefinition;
            reference rootTypeDefinition : XsSimpleTypeDefinition;
--          .....
        }
    }

    package SAWSDL {
        class ModelReference extends XsAnnotation{
        }

        class LiftingSchemaMapping extends XsAnnotation{
        }

        class LoweringSchemaMapping extends XsAnnotation{
        }
    }
}
```

Fig. 3. Fragment from SAWSDL metamodel presented in KM3 format

Our approach to modeling Web services is taken from the perspective of the potential patterns of the use of services. That is, we model services from the perspective of MEPs. We first start from the definition of a business rule that corresponds to a MEP under study, but without considering the Web services that might be developed to support that rule.

We explain our modeling approach on an example of modeling in-out MEP in URML. The in-out MEP consists of exactly two messages: when a service receives an input message, it has to reply with an output message. The business rule that we use in this example is this: On a customer request for checking availability of a hotel room during some period of time, if the specified check-in date is before the specified check-out date, and if the room is

available, then return to the customer a response containing the information about availability of the room, or if this is not the case return a fault message. URML model of this rule is presented on the Fig. 4.

This business rule is modeled with the two reaction rules, after which those rules are mapped to Web services (i.e. SAWSDL descriptions). A triggering event of a reaction rule (CheckAvailability) maps to the input message of a Web service operation. The action of the reaction rule, which is triggered when a condition is true (CheckAvailabilityResponse), maps to the output message of the Web service operation. The action of the second reaction rule (InvalidDataError), triggered on a false condition, maps to the out-fault message of the Web service operation. To model condition constructs (checkinDate < checkoutDate) we use OCL filters [20]. OCL filters are based on a part of OCL that models logical expressions, which can be later translated to R2ML logical formulas, as parts of reaction rules. However, these OCL filters cannot be later translated to Web service descriptions (WSDL nor SAWSDL), as those languages cannot support such constructs. But, we can translate our URML models into rule-based languages (e.g., Jess or Drools). This means that for each Web service, we can generate a complementary rule, which fully regulates how its attributed service is used.

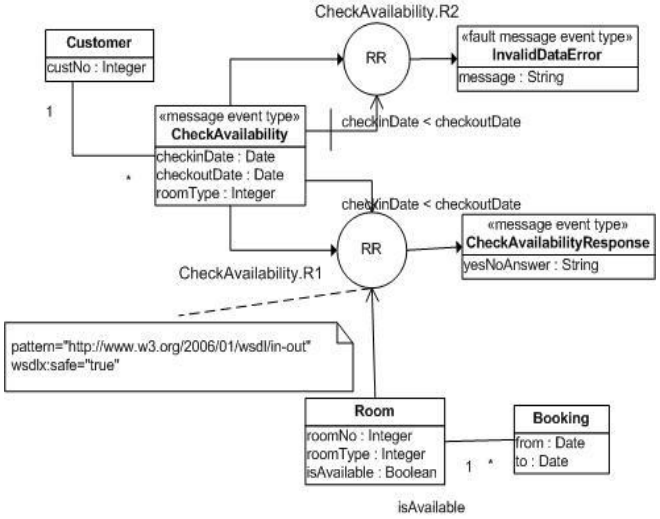

Fig. 4. URML model

We have developed a tool that supports this approach. The tool is called Strelka and it is developed as a plug-in for the Fujaba UML tool. Strelka fully supports the URML modeling previously described, plus it has the ability to call the transformations we use (that we describe in the next section) in order to automate the mappings between Web services (i.e. SAWSDL) and URML.

The modeling approach presented here can be equally used for the modeling of Semantic Web services (i.e., the URML diagram does not have to take any changes). The reason for this is the fact that we are making an assumption that domain model and domain ontology are the equivalent. What this means is that the ontology that we reference from the SAWSDL document is fully defined by our working Vocabulary. For example, in Fig. 4, our Vocabulary contains classes such as Customer, CheckAvailability, and InvalidDataError where Customer is of type Class, CheckAvailability is of type MessageType, and InvalidDataError is of type FaultMessageType). This equalization of domain model and domain ontology further leads to the following consequences: i) we do not need to change neither the URML nor R2ML metamodel, and ii) we need to use only one extensibility attribute that SAWSDL offers – *modelReference*; iii) the use of *liftingSchemaMapping* and *loweringSchemaMapping* is not necessary, as there are no mismatches between the semantic model and the XML structures we use within WSDL.

In the next section, we briefly describe model transformations as the key part of our solution, where we concentrate on the changes that needed to be employed, in order to fully support our approach for Semantic Web service modeling.

IV. MODEL TRANSFORMATIONS

In Fig. 7, we give the tools and transformations that we have developed to support our modeling framework. In the central part of the figure we have Strelka - the native serialization of URML models in Strelka is the R2ML XML concrete syntax. To support generation of SAWSDL-based Web services, we need to translate R2ML XML-based models to WSDL. In this transformation, we also need to annotate (with modelReference attribute) all the necessary WSDL elements, so that they reference the ontology that we generate from R2ML Vocabulary. As shown in Fig. 7, we have two files as the output of our system: SAWSDL XML file, and OWL file that presents our ontology. In the next two figures (Fig. 5 and Fig. 6), we give snippets of these files.

```
<?xml version="1.0" encoding="utf-8" ?>
<description xmlns="http://www.w3.org/2006/01/wsdl" .... >
  <types>
    <!-- ... -->
  </types>

  <interface name="tns:reservationInterface">

    <fault name="tns:InvalidDataFault"
    sawsdl:modelReference="http://example.org/Ontology#InvalidDataError"
    element="ex:InvalidDataError" />

    <operation name="tns:checkAvailability"
    pattern="http://www.w3.org/2006/01/wsdl/in-out">

    <input messageLabel="In"
    sawsdl:modelReference="http://example.org/Ontology#CheckAvailability"
    element="ex:CheckAvailability" />
    <output messageLabel="Out"
    sawsdl:modelReference="http://example.org/Ontology#CheckAvailabilityResponse"
    element="ex:CheckAvailabilityResponse" />
    <outfault ref="tns:InvalidDataFault"
    messageLabel="Out" />

    </operation>
  </interface>
  <!-- ... -->
</description>
```

Fig. 5. SAWSDL snippet we get at the end of our system

It is important to say that, we have developed bidirectional transformations, i.e. we also support transformation from WSDL documents to R2ML models, in order to enable reverse engineering of existing Web services, thus enabling an extraction of business rules that were already integrated into to the implementation of Web services.

We have decided to implement transformation from R2ML to WSDL at the level of metamodels by using the

model transformation language ATL [22]. To support our approach, we needed to implement a number of transformations between different languages and their representations (all of them are bidirectional):

```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
<rdf:RDF xmlns:rdf = 'http://www.w3.org/1999/02/22-rdf-syntax-ns#' ... >
  <owl:Ontology rdf:about = 'RoomAvailability'/>

  <owl:Class rdf:ID = 'CheckAvailability'>
    <rdfs:label>CheckAvailability</rdfs:label>
    <ns:msgType>MessageType</ns:msgType>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource = '#CheckAvailability.checkInDate'/>
        <owl:cardinality  rdf:datatype =
        'http://www.w3.org/2001/XMLSchema#nonNegativeInteger'>1</owl:cardinality >
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource = '#CheckAvailability.customer'/>
        <owl:cardinality  rdf:datatype =
        'http://www.w3.org/2001/XMLSchema#nonNegativeInteger'>1</owl:cardinality >
      </owl:Restriction>
    </rdfs:subClassOf>
    <!-- ... -->
  </owl:Class>

  <owl:DatatypeProperty rdf:ID = 'CheckAvailability.checkInDate'>
    <rdfs:domain rdf:resource = '#CheckAvailability'/>
    <rdfs:range rdf:resource = 'http://www.w3.org/2001/XMLSchema#dateTime'/>
  </owl:DatatypeProperty>

  <owl:ObjectProperty rdf:ID = 'CheckAvailability.customer'>
    <rdfs:domain rdf:resource = '#CheckAvailability'/>
    <rdfs:range rdf:resource = '#Customer'/>
  </owl:ObjectProperty>

<owl:AnnotationProperty rdf:about="&ns;msgType"/>
  <owl:oneOf rdf:parseType="Collection">
    <owl:Class rdf:about="#MessageType"/>
    <owl:Class rdf:about="#FaultMessageType"/>
  </owl:oneOf>
</owl:AnnotationProperty>

  <!-- ... -->
</rdf:RDF>
```

Fig. 6. OWL snippet we get at the end of our system

- URML and R2ML XML concrete syntax (transformation no. 1 on Fig. 7). This is the only transformation that is not implemented by using ATL, because Fujaba does not have explicitly defined metamodel in a metamodeling language such as MOF. We based this transformation on the use of JAXB (Java Architecture for XML Binding). JAXB guarantees that the R2ML XML rule sets comply with the R2ML XML schema.
- R2ML XML-based concrete syntax and R2ML metamodel (transformation 2 on Fig. 7). This transformation is important to bridge between the concrete (XML) and abstract (MOF) syntax of R2ML. This is done by using ATL and by leveraging ATL's XML injector and extractor for injecting/extracting XML models into/from the MOF-based representation of rules.
- R2ML metamodel and SAWSDL metamodel (transformation 3 on Fig. 7). This transformation is the core of our solution and presents mappings between R2ML and SAWSDL at the level of their abstract syntaxes.
- SAWSDL XML-based concrete syntax and SAWSDL metamodel (transformation 4 in Fig. 7). This transformation is important to bridge concrete (XML) and abstract (MOF) syntax of SAWSDL. This is also done by using ATL i.e. by leveraging ATL's XML injector and extractor.
- R2ML metamodel and ODM (Ontology Definition Metamodel) [21] (transformation 5 in Fig. 7). This transformation is necessary in order for us to get an OWL representation of our Vocabulary. It presents mappings between R2ML and OWL at the level of their abstract syntax. And finally the transformation between ODM and OWL (transformation 6 in Fig. 4) has already been developed as part of the use case presented in [30].

Due to the size constraints for this paper, we explain these mappings in the table form – these tables contain very small excerpts from the mappings between the metamodels presented in their column headers.

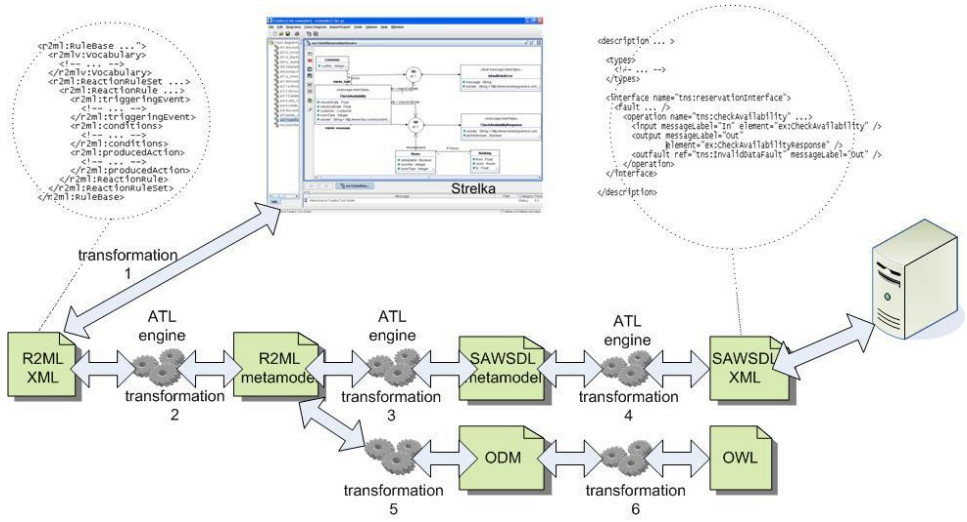

Fig. 7. Transformation chain for bidirectional mapping between URML and SAWSDL

Table 1 presents an excerpt of the mapping between the R2ML XML schema and R2ML metamodel. Actually, we did not have to make any change in this transformation comparing to our original approach, explained in [8], because the assumption of equality of the domain model and domain ontology did not require it. Under this assumption, there was no need to change the R2ML metamodel, because R2ML vocabulary matches our domain ontology (i.e. they are equivalent.

Table 2 presents an excerpt of the mapping between the SAWSDL XML schema and SAWSDL metamodel. Our original approach contained transformation between the WSDL XML schema and WSDL metamodel. This transformation had to be changed, so that it could take into considerations annotations (more precisely, in our case modelReference attribute) introduced by the SAWSDL (and adequately presented in the SAWSDL metamodel whose fragment is shown in Sect. 3.C)

Table 1. An excerpt of the mapping between the R2ML XML schema and R2ML metamodel

R2ML XML schema	**XML metamodel**	**R2ML metamodel**
RuleBase	Root name=`r2ml:RuleBase`	RuleBase
Description: This is a root element. It contains a collection of rules		
ReactionRuleSet	Element name=`r2ml:ReactionRuleSet`	ReactionRuleSet
Description: This element contains a collection of reaction rules		
ReactionRule	Element name=`r2ml:ReactionRule`	ReactionRule
Description: This element represents a reaction rule		

Table 2. An excerpt of the mapping between the SAWSDL XML schema and SAWSDL metamodel

SAWSDL XML schema	**XML metamodel**	**SAWSDL metamodel**
description	Root name=`description`	Description
Description: This is the root element. It contains these elements: types, binding, service and interface		
interface	Element name=`interface`	Interface
Description: This element contains operation and fault elements		
operation	Element name=`operation`	Operation
Description: This element contains in/outfault and in/output elements		

Table 3 presents an excerpt of the mapping between the R2ML metamodel and SAWSDL metamodel. As our original transformation was between the R2ML metamodel and WSDL metamodel, this transformation also had some minor changes in order to reflect *modelReference* attribute – i.e. we populate the value of this attribute with the appropriate URI that references classes from our referring ontology (or in our case, from R2ML vocabulary).

Finally, in table 4, we present an excerpt of the mapping between the R2ML metamodel and OWL metamodel (ODM). There was no need for this transformation in our original approach, i.e. this is newly developed transformations, and as such it is not yet completely finished, rather it is work in progress. R2ML and ODM have similar ways of presenting vocabulary – they both have properties defined as independent concepts with their own domain and range. This is for example, different from the UML way of defining properties where UML attributes and association ends are dependent on their enclosing classes.

Table 3. An excerpt of the mapping between the R2ML metamodel and SAWSDL metamodel

R2ML	**SAWSDL**
RuleBase	Description
Description: This is a root element.	
Vocabulary	ElementType
Description: R2ML Vocabulary is mapped to XML schema language (which SAWSDL uses for defining message types and vocabularies)	
ReactionRuleSet	Interface
Description: R2ML ReactionRuleSet maps to the SAWSDL Interface element (SAWSDL document can have just one Interface element)	
MessageEventExpression	Input
Description: R2ML MessageEventExpression actually maps to different SAWSDL elements (Input, Infault, Output and Outfault elements)	

Table 4. An excerpt of the mapping between the R2ML metamodel and ODM metamodel

R2ML	**ODM**
Class	OWLClass
Description: R2ML Class concept represents the class element similar to the notion of the UML Class.	
Attribute	OWLDatatypeProperty
Description: R2ML Class can have Attribute - Attribute maps to ODM OWLDatatypeProperty	
ReferenceProperty	OWLObjectProperty
Description: R2ML Class can have ReferenceProperty – ReferenceProperty maps to ODM OWLObjectProperty	

V. CONCLUSION

In this paper, we have shown one approach to modeling semantically-enriched Web services - the approach that leverages the use of both MDE principles and reaction rules. We have backed up this approach with a working example that consists, among other things (i.e. creating URML model), of bidirectional transformations between R2ML reaction rules and SAWSDL descriptions, allowing us an extraction of business rules that were already integrated into the implementation of Web services. The assumption we made in our approach, is that our domain model (presented as the URML diagram) and domain ontology (that we reference in order to identify some piece of semantics) are the same. As a result, we get a simple solution which is a good starting point for the more general one in which domain model and domain ontology are not the same and where we can leverage all the results obtained here (e.g. all the transformations can be reused with just a minor modifications). The approach presented here relies on our previous work described in [8], so it keeps all the advantages gained there: By using the MDE principles we have been able to develop a framework for modeling Web services from the perspective of how services are used in terms of message-exchange patterns (MEPs). Our approach enables developers to focus on the definition of business rules, which regulate MEPs, instead of focusing on low level Web service details or on contexts where services are used (i.e., workflows).

As rules are closer to the problem domain, rule-based models of services are much closer to business experts, and the process of knowledge/requirements elicitation is more reliable, as well as collaboration between service developers and business experts. The use of model transformations allows for transforming platform independent models of business logic to specific platforms such as Web services.

The process of modeling (semantic) Web services described in this paper can be used in the wider scope of integration of business rules and business processes. That is to say, it is widely acknowledged that business process management would greatly benefit from integration with business rule management. The key idea behind this integration is to extract some parts of a business logic contained implicitly in business process models into explicit definitions of business rules. But there is still no established solution to this integration problem, and the leading business process modeling language, BPMN [16], does not provide any explicit support for rules. In [15] the approach to integration of business rules (R2ML) and business processes (BPMN) has been proposed, by using the MDE principles. Authors have created a new rule-based process modeling language called rBPMN, by integrating metamodels of two languages (i.e., BPMN and R2ML). It is shown how our modeling approach can be practically implemented in the area of business process management. Our plan for the future is to further experiment in this area, and to find the best way for its semantically enrichment.

VI. REFERENCES

[1] Web Services Description Language (WSDL) Ver. 2.0 Part 1: Core Language. W3C Candidate Rec. http://www.w3.org/TR/2007/REC-wsdl20-20070626.

[2] UDDI Ver. 3.0.2. OASIS, 2004, http://uddi.org/pubs/uddi v3.htm.

[3] SOAP Ver. 1.2 Part 1: Messaging Framework. W3C Recommendation, http://www.w3.org/TR/soap12-part1/.

[4] Margaria, T., "Service Is in the Eyes of the Beholder," Computer, vol. 40, no. 11, pp. 33-37, Nov., 2007

[5] Kopecky, J., Vitvar, T., Bournez., C., Farrell, J., "SAWSDL: Semantic Annotations for WSDL and XML Schema", IEEE Internet Computing, vol 11, pp. 60-67, Nov-Dec., 2007

[6] Verma, K., Sheth, A., "Semantically Annotating a Web Service", IEEE Internet Computing, vol 11, pp. 83-85, March-April, 2007

[7] Semantic Annotations for WSDL and XML Schema, W3C Recommendation 2007, http://www.w3.org/TR/sawsdl/

[8] Ribaric, M., Gaševic, D., Milanovic, M., Guirca, A., Lukichev, S., Wagner, G., "Model-Driven Engineering of Rules for Web Services," In Lämmel, R., Saraiva, J., & Visser, J. (Eds.) Post-Proceedings of 2nd Summer School on Generative and Transformational Techniques, Springer, 2008, in press.

[9] S. Lukichev and G. Wagner. Visual rules modeling. Proceedings of the 6th International Conference Perspectives of Systems Informatics, pp. 467–673, 2006.

[10] S. Lukichev and G.Wagner. UML-based rule modeling with Fujaba. Proceedings of the 4th International Fujaba Days 2006, pp. 31–35, 2006.

[11] D.C. Schmidt. Model-Driven Engineering. Computer, 39(2):25–31, 2006.

[12] Leymann, F., Roller, D., "Business processes in a Web services world", http://www.ibm.com/developerworks/library/ws-bpelwp/

[13] A. Charfi and M. Mezini. Hybrid Web service composition: Business processes meet business rules. In Proc. of 2nd Int'l Conf. on Service Oriented Comp., pp. 30–38, 2004.

[14] R. G. Ross. Principles of the Business Rule Approach. Addison-Wesley, 2003.

[15] Milanovic, M., Gaševic, D., Wagner, G., "Combining Rules and Activities for Modeling Service-Based Business Processes", International Workshop on Models and Model-driven Methods for Enterprise Computing (3M4EC), in conjunction with The Twelfth IEEE International EDOC Conference (EDOC 2008), Munich, Germany, 2008. (to be published)

[16] Object Management Group, "Business Process Model and Notation (BPMN) Specification 2.0", initial submission, http://www.omg.org/cgi-bin/doc?bmi/08-02-06, 2008.

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

[18] Wagner, G., Giurca, A., Lukichev, S., R2ML: A General Approach for Marking up Rules, In Dagstuhl Seminar Proc. 05371, 2006

[19] Guirca, A., Lukichev, S., Wagner., G., Modeling Web Services with URML", In Proceedings of Workshop Semantics for Business Process Management, 2006

[20] Object Management Group, Object Constraint Language,OMG Specification, Version 2.0, formal/06-05-01, http://www.omg.org/docs/formal/06-05-01.pdf, 2006.

[21] OMG ODM (2005). "Ontology Definition Metamodel," Third Revised Submission

[22] Atlas Transformation Language - User Manual, ver. 0.7, ATLAS group, Nantes http://www.eclipse.org/gmt/atl/doc/ATL_User_Manual[v0.7].pdf

[23] Nagarajan, M., Verma, K., Sheth, A., Miller, J., Lathem, J., "Semantic Interoperability of Web Services – Challenges and Experiences", In Proc. of Int'l Conf. on Web Services, pp. 373–382, 2006.

[24] Sivashanmugam, K., Verma, K., Sheth, A., Miller, J., Adding Semantics to Web Services Standards, In Proc. of the 1st Int'l Conf. on Web Services (ICWS'03), Las Vegas, Nevada (June 2003) pp. 395-401

[25] Sheth, A., Verma, K., Gomadam, K., "Semantics to Energize the Full Services Spectrum," Comm. ACM, vol. 49, no. 7, 2006, pp. 55–61.

[26] Verma, K., Sheth, A., "Using SAWSDL for Semantic Service Interoperability", Tutorial at Semantic Technology Conference, San Jose, CA, May 21, 2007.

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

[28] Jouault, F., Bézivin, J., "KM3: a DSL for Metamodel Specification", In Proceedings of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems, Bologna, Italy, pp. 171-185, 2006.

[29] The Model Development Tools (MDT) project, http://www.eclipse.org/xsd/

[30] ATL Use Case - ODM Implementation (Bridging UML and OWL): http://www.eclipse.org/m2m/atl/usecases/ODMImplementation/

[31] Papazoglou, M. P. and Georgakopoulos, D. 2003. Introduction. Commun. ACM 46, 10 (Oct. 2003), 24-28. DOI= http://doi.acm.org/10.1145/944217.944233